Hamiltonian Monte Carlo for (Physics) Dummies

Dootika Vats

February 11, 2023

Introduction

This document is meant for researchers (like myself) working in Markov chain Monte Carlo (MCMC) or Bayesian inference, who are unfamiliar with physics vernacular, but want to understand Hamiltonian Monte Carlo a little more closely than most references allow.

Throughout this document, I assume a working knowledge of the general Metropolis-Hastings (MH) algorithm. If you are unfamiliar with this, I suggest to first read into these algorithms before proceeding further. My guiding reference is still the popular book chapter by Neal et al. (2011), but I try to fill in the details where I can.

Metropolis-Hastings-Green Algorithm

Suppose π is a target distribution with density function $\pi(x)$; interest is in obtaining samples from π. As is typical in many Bayesian problems, obtaining independent and identically distributed samples from π is challenging, and one resorts to obtaining correlated samples via a Metropolis-Hastings algorithm.

The popular Metropolis-Hastings algorithm has the following form: for a given state x_k, the next iteration is obtained by

1. Draw $x^* \sim Q(\cdot|x_k)$, where this Q is a proposal distribution with density $q(\cdot|x_k)$ that often depends on the current state.
2. Independently from the above, draw $U \sim U(0,1)$.
3. Calculate
$$r(x_k, y) = \frac{\pi(x^*)}{\pi(x_k)} \frac{q(x_k|x^*)}{q(x^*|x_k)} \,.$$
4. If $U \leq \min\{1, r(x_k, x^*)\}$, then set $x_{k+1} = x^*$
5. Else, set $x_{k+1} = x_k$.

The MH algorithm above is relatively simple to understand, and easy to implement. The quality of the resulting samples depends on the complexity of π and the choice of proposal Q. Various choices of Q have been explored in the literature, each having its own merits and demerits.

A variant of the Metropolis-Hastings algorithm is the Metropolis-Hastings-Green algorithm with Jacobians (MHGJ), (see Geyer (2003)). Studying this variant is particularly useful in the context of Hamiltonian Monte Carlo (HMC). Essentially, consider the idea of proposing x^* in the M-H agorithm to be a *deterministic jump* from x_k. That is, the proposal distribution from x_k is a Dirac mass at x^*. Naturally, such proposal distributions will yield samples that do not move freely on the state space, and are likely going to yield reducible/periodic Markov chains.

The MHGJ algorithm augments the state space so that an auxilliary variable (y) is sampled randomly from its conditional distribution, and then the augmented state (x_k, y) is mapped deterministically to a proposal. Introducing this auxilliary variable enables randomness that ensures ergodicity features, while also allowing for deterministic jumps. Deterministic jumps by itself are not superior, but *informed* deterministic jumps will prove to be useful. Let's be more specific.

Let $g(x, y)$ be a function such that $g(g(x, y)) = (x, y)$. That is, g is an involution. Given x_k, the algorithm proceeds as follows:

1. Draw $y \sim S(\cdot|x_k)$ (the auxiliary variable), that has density $s(y|x_k)$.
2. Set $(x^*, y^*) = g(x_k, y)$.
3. Calculate
$$r(x_k, x^*) = \frac{\pi(x^*)s(y^*|x^*)}{\pi(x_k)s(y|x_k)}\det(\nabla g(x_k, y))$$
4. If $U \leq \min\{1, r(x_k, x^*)\}$, then $x_{k+1} = x^*$
5. Else $x_{k+1} = x_k$.

Intuitively, compared to the MH algorithm, the ratio without the Jacobian emulates the ratio joint density of (x, y); since the proposal is a Dirac mass, the density function disappears, and what remains is the Jacobian of the change-of-variables transformation. The fact that g should be an involution is required to ensure that the "proposal distribution" is symmetric.

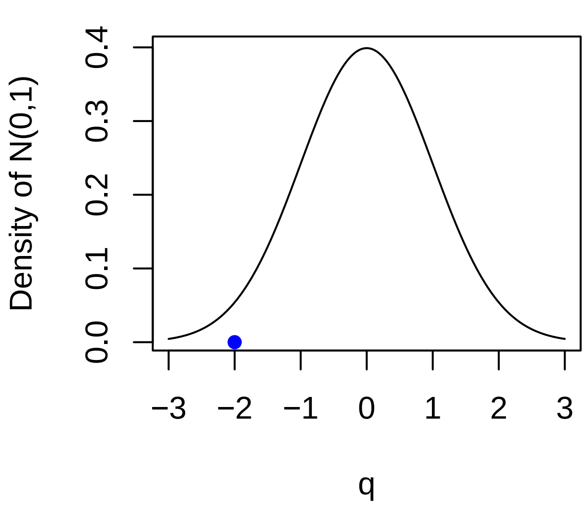
In order to implement the MHGJ algorithm, one requires choosing an appropriate distribution S and the deterministic map, g. Hamiltonian Monte Carlo corresponds to a particular choice of S and g.

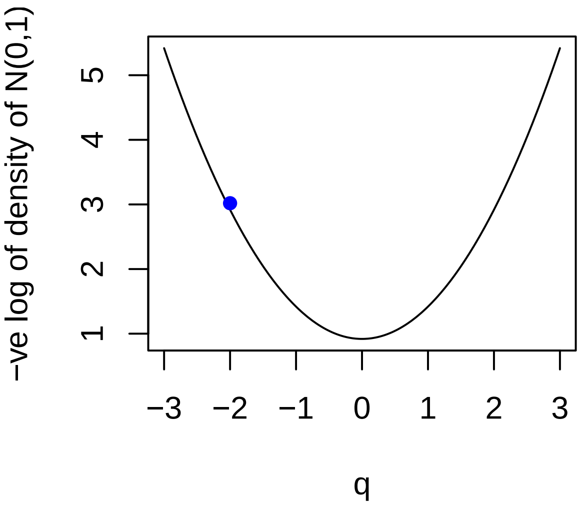
Hamiltonian Dynamics

Consider an object moving on a friction-less surface. At any moment in time, the object has both *potential energy* and *kinetic energy*. The potential energy is determined by the height of the object from the ground, and the kinetic energy is determined by its momentum. What does this have to do with our problem statement of sampling from π? Well, consider function

$$U(q) = -\log(\pi(q)).$$

Then a draw $q \sim \pi$ is thought to have potential energy $U(q)$. This is an assumed setup. Below is a pictorial demonstration of $\pi = N(0, 1)$. Points in the tail have large potential energy and points in areas of high probability are given low potential energy.





Now, let's assume the object is moving on the surface described by $U(q)$, with a momentum p. That is the object of mass m has kinetic energy

$$K(p) = \frac{p^2}{2m}\,.$$

If left undisturbed, this object moves on the surface while conserving its total energy:

$$H(q,p) = U(q) + K(p)\,.$$

The total energy of the system is the Hamiltonian, $H(q,p)$. The Hamiltonian is also incredibly useful in determining how the object will move in a given system. So, suppose you are given information on U and K for an object, and need to determine where the object will be at a given time, t, then Hamilton's equations help us:

$$\frac{dq}{dt} = \frac{\partial H(q,p)}{\partial p} \qquad \text{and} \qquad \frac{dp}{dt} = -\frac{\partial H(q,p)}{\partial q}\,.$$

The solutions to these system of equations will yield $q(t)$ and $p(t)$, the position and momentum at time t, respectively, and will depend on the initial condition: starting position at $t = 0$ and starting momentum at $t = 0$.

Example: Gaussian Potential Energy

Let's see a more concrete example of this. As before, consider $\pi = N(0,1)$ and $m = 1$, so that

$$U(q) = \frac{q^2}{2} \quad \text{and} \quad K(p) = \frac{p^2}{2} \quad \Rightarrow \quad H(q,p) = \frac{q^2}{2} + \frac{p^2}{2}\,.$$

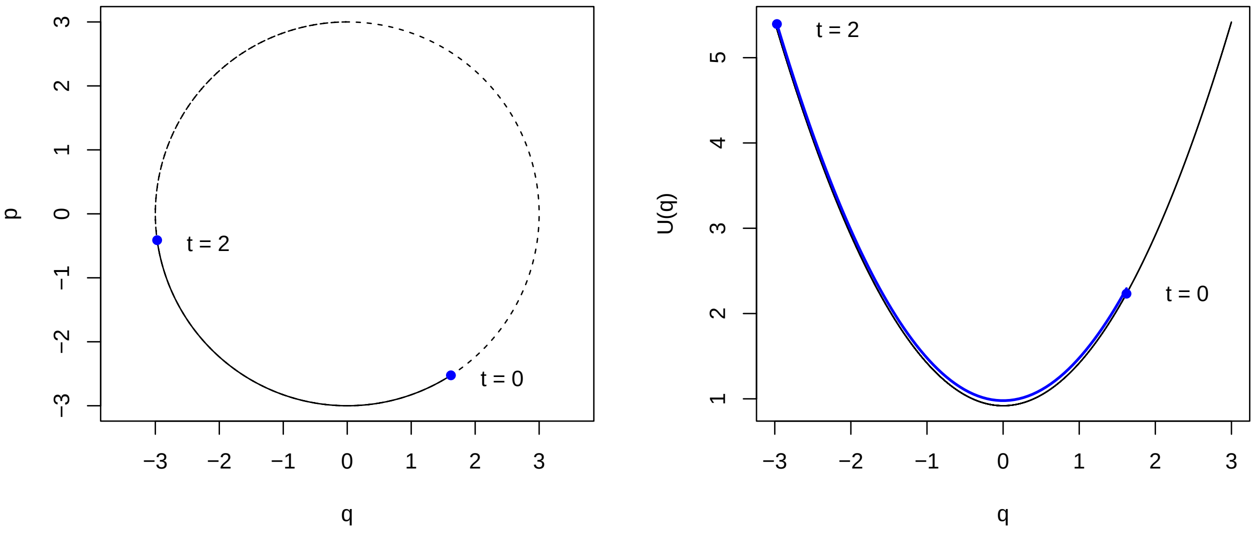
Using Hamilton's equations we obtain

$$\frac{dq}{dt} = \frac{\partial H(q,p)}{\partial p} = p \qquad \frac{dp}{dt} = -\frac{\partial H(q,p)}{\partial q} = -q\,.$$

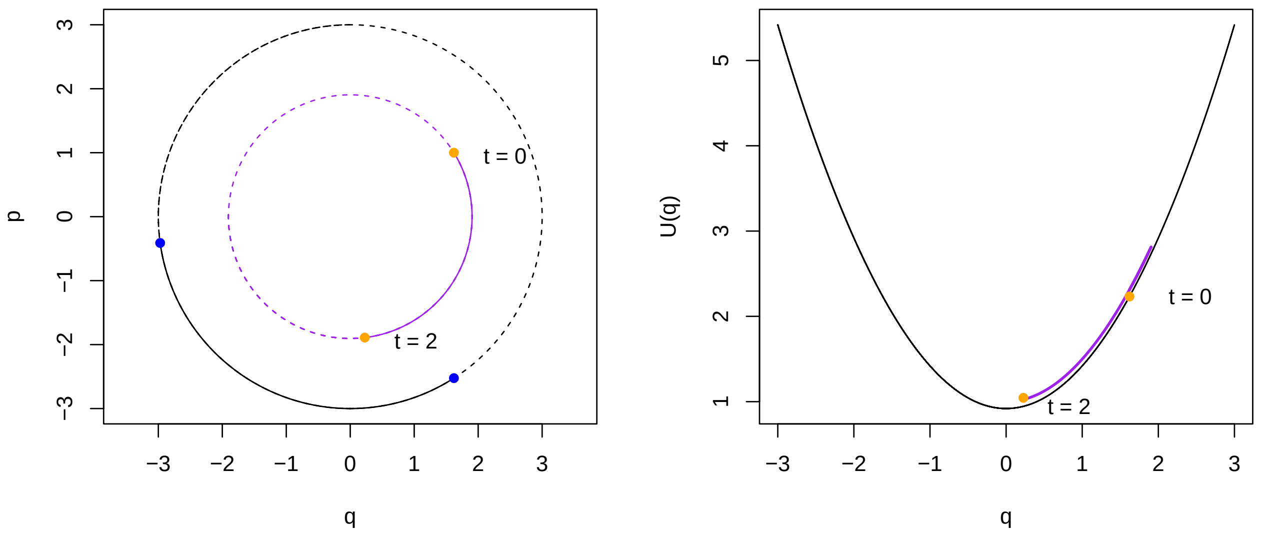
The solutions to these system of PDEs is available analytically. For constants a, r denoting initial conditions, one can obtain the solutions below:

$$q(t) = r\cos(a+t) \qquad \text{and} \qquad p(t) = -r\sin(a+t)$$

which indicates the position at time t is $(q(t), p(t))$, when the object starts its motion at $(q(0), p(0)) = (r\cos(a), -r\sin(a))$. In the figure below (left), I set $r = 3$ and $a = 1$, so that $q(0) = 1.621$ and $p(0) = -2.52$. This is the point where the particle starts — from the location 1.621 with momentum -2.52. Then I draw the joint path of $(q(t), p(t))$ as time evolves (which just describes a circle here). Since the initial energy is negative, when I progress the particle for $t = 2$ time units, it decreases the location component q. In the left is the corresponding potential energy function, $U(q) = -\log\pi(q)$.



It is crucial to highlight here that although the total energy is preserved for a given initial condition (along one given circle), changing the initial condition leads to a different energy level of the system. So for instance, if we were to restart with a different momentum, it will then proceed traversing the bowl with a different $H(q, p)$. In the figure below, I reset the initial condition so that $q(0)$ is the same but now $p(0) = 1$. For this momentum, the object starts going to the right, until the kinetic energy is 0, and then the momentum flips direction, and the balls starts falling.



In this way, "resampling" the momentum (which means, drawing a new momentum from its own distribution) can create different progressions of the position of the particle, in the same time unit ($t = 2$, in this case). Energy is conserved within each circle, but naturally, not conserved jumping from one circle to another. In this way, each circle corresponds to an energy level. If high momentum is sampled, then this large momentum allows the particle to farther away (the radius of the circle is bigger). This feature of resampling the momentum is what HMC is based on.

Hamiltonian Monte Carlo

Alright then, so the Hamiltonian dynamics are nice and all, but why do we care? Hopefully, you can already imagine that the potential energy function $U(q)$ allows us to track the position of our particle in our domain. We will *augment* an artificial *kinetic energy*, which will serve as the auxilliary variable. For a new kinetic energy sampled, we will restart our motion from the previous location q_k, along with the new kinetic energy, and let the *deterministic* Hamiltonian dynamic tell us where it has reached after some s time units. That will be the new proposed state of motion which we shall accept or reject. Since Hamiltonian dynamics are deterministic, this falls intuitively under the MHGJ paradigm. Special appreciation to Andrew Holbrook's video ("Dr. Andrew Holbrook's Lecture on Hamiltonian Monte Carlo (HMC)" (n.d.)).

Let $T_s(q, p)$ denote the progression of the particle after time unit s using $H(q, p) = U(q) + K(p)$ as defined above. That is $T_s(q_t, p_t) = (q_{(s+t)}, p_{(s+t)})$. It is well known that the Hamiltonian update is volume-preserving, that is $|\nabla T_s(q, p)| = 1$.

The mapping T_s, however is not an involution. But, consider a function $i(q, p) = (q, -p)$ and define $P_s(q, p) = i(T_s(p, q))$. That is, P_s follows the Hamiltonian for time s and then flips the sign of the momentum. The function P_s is an involution, so that $P_s(P_s(q, p)) = (q, p)$! P_s is also volume-preserving. Also notice that since $K(p)$ is an even function of p,

$$H(q, -p) = U(q) + K(-p) = U(q) + K(p) = H(q, p)\,.$$

Now, we have the ingredients to describe the Hamiltonian Monte Carlo algorithm. In the MHGJ algorithm framework, $x_k = q_k$ and $y = p$. Further $S(\cdot|q_k) = N(0, m)$ for some $m > 0$ and function $g(q_k, p) = P_s(q_k, p)$. The exact HMC algorithm, given state q_k, proceeds with q_{k+1} as:

1. Draw $p \sim N(0, m)$, the density of which I denote as $\phi(p)$.
2. Calculate $(q^*, p^*) = P_s(q_k, p)$, determined by progressing the particle s time units using the Hamiltonian dynamics, and then flipping the sign of the momentum.
3. Calculate ratio:

$$r(q_k, q^*) = \frac{\pi(q^*)\phi(p^*)}{\pi(q_k)\phi(p)} |\nabla P_s(q_k, p)| \,.$$

4. If $U \leq \min\{1, r(q_k, q^*)\}$ then $q_{k+1} = q^*$
5. Else, $q_{k+1} = q_k$.

Let's look at $r(q_k, q^*)$ a little closely now. The Jacobian term is 1 due to the volume-preserving feature. Recall that $U(q) = -\log \pi(q)$ and since $\phi(p) \propto e^{-p^2/2m}$, we have that $K(p) = -\log \phi(p)$. This implies that

$$r(q_k, q^*) = \frac{\pi(q^*)\phi(p^*)}{\pi(q_k)\phi(p)} = \frac{e^{-U(q^*)-K(p^*)}}{e^{-U(q_k)-K(p)}} = e^{-H(q^*,p^*)+H(q_k,p)} \,.$$

Now (q^*, p^*) was obtained by pushing the particle s steps from initial position (q_k, p) and flipping the sign. Since $H(q^*, -p^*) = H(q^*, p^*)$, and the total energy is preserved, we obtain

$$r(q_k, q^*) = 1 \,.$$

This makes steps 4 and 5 redundant and we can rewrite the algorithm as:

1. Draw $p \sim N(0, m)$.
2. Calculate $(q^*, p^*) = P_s(q_k, p)$, determined by progressing the particle s time units using the Hamiltonian dynamics, and then flipping the sign of the momentum.
3. Set $q_{k+1} = q^*$.

In practice, flipping the sign is not needed since a new p is sampled at the next iteration, independent of current momentum, p.

Example: $N(0, 1)$ target

Let's continue the Gaussian example. Recall $\pi = N(0, 1)$ and $m = 1$, so that

$$U(q) = \frac{q^2}{2} \quad \text{and} \quad K(p) = \frac{p^2}{2} \quad \Rightarrow \quad H(q, p) = \frac{q^2}{2} + \frac{p^2}{2} \,.$$

Using Hamilton's equations we obtained

$$\frac{dq}{dt} = \frac{\partial H(q, p)}{\partial p} = p \qquad \frac{dp}{dt} = -\frac{\partial H(q, p)}{\partial q} = -q \,.$$

The solutions to these system of PDEs were

$$q(t) = r\cos(a+t) \qquad \text{and} \qquad p(t) = -r\sin(a+t)$$

Recall that in the HMC algorithm, when we resample the momentum, p, we change the energy level of the system, and reset the initial time to zero, and thus require recalculating the initial conditions.

That is, at the iteration index k, given $q(0) = q_k, p(0) = p$, we first need to find the r and a corresponding to these initial conditions:

$$r_k^2 = q_k^2 + p^2 \Rightarrow r_k = \pm\sqrt{q_k^2 + p^2} \qquad \text{and } a_k = \cos^{-1}(q_k/r_k)\ .$$

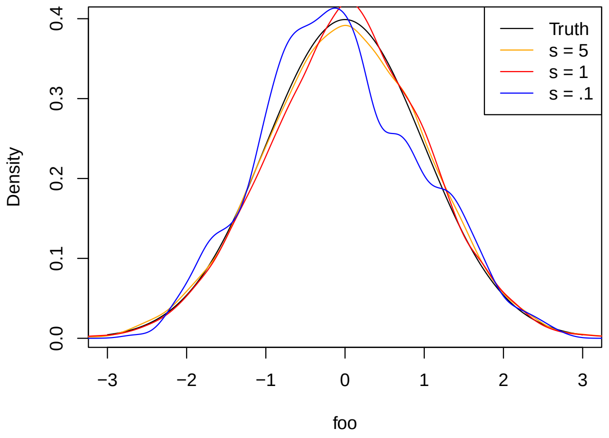
And then we obtain $q^* = q(s) = r_k\cos(a_k + s)$ and $-p* = -r_k\sin(a_k + s)$. Since flipping the sign is not practically necessary, I will just call this p^*. This gives us the full recipe to simulate this HMC.

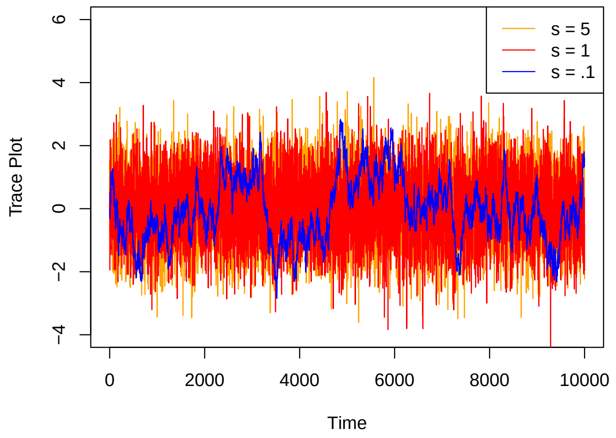
```r
normalHMC <- function(s = 1, n = 1e4)
{
  qt <- numeric(length = n)
  qt[1] <- 0 # starting here
  # don't need a starting value of p

  for(k in 2:n)
  {
    # new momentum
    p <- rnorm(1)

    #initial conditions for new H
    r2 <- qt[k-1]^2 + p^2
    # choosing +- with probability 1/2
    r <- sample(c(sqrt(r2), -sqrt(r2)), size = 1)
    a <- acos(qt[k-1]/r)

    # simulating Hamiltonian forward s time units
    qt[k] <- r*cos(a + s)
    p <- -r * sin(a + s)
  }
  return(qt)
}
```

The function `normalHMC` runs the HMC algorithm (without any accept reject). The quality of the samples depends on s. If s is too small, we are not simulating the particle too much in time, and thus, it will not move enough distance. If we set s to be too large, it might be redundant, as it will circle back to the same spot often.

```r
chain1 <- normalHMC(s = .1)
chain2 <- normalHMC(s = 1)
chain3 <- normalHMC(s = 5)
```

The following are the density and trace plots from the chains above. Notice that $s = 1$ and $s = 5$ are similar, for the reason explained before. $s = .1$ is too short of a time progression for the particle to move too far away from its current location.





Of course, all this seems too good to be true, and it is. The reality is, for most real π, the solutions to Hamilton's equations are not available analytically. And if they are, they may not be available to be inverted to understand the initial conditions. In these situations, the true PDEs underlying Hamilton's equations can be approximated using numerical integrators. The most popular of these is the Leap Frog Integrator, due to its simplicity and reasonable performance.

Leap Frog Integrator

Recall that Hamilton's equations are:

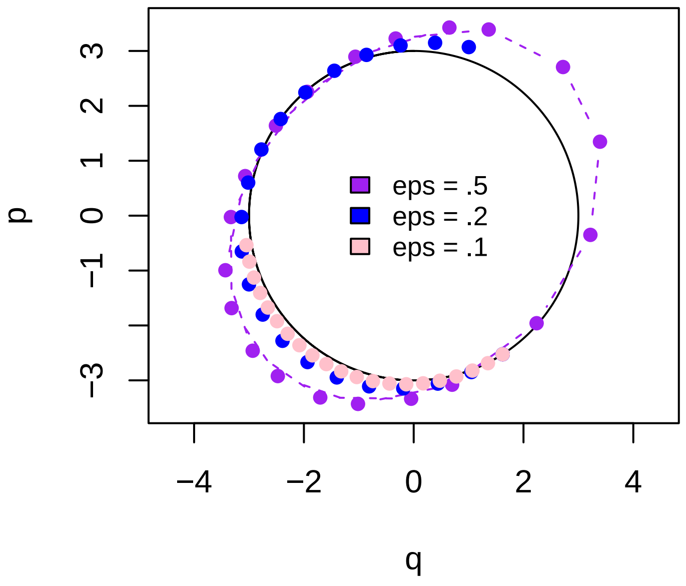
$$\frac{dq}{dt} = \frac{\partial H(q,p)}{\partial p} \qquad \text{and} \qquad \frac{dp}{dt} = -\frac{\partial H(q,p)}{\partial q} .$$

A discrete linear approximation for time unit ϵ to these will yield:

$$p(t+\epsilon) = p(t) + \epsilon\frac{dp(t)}{dt} = p(t) - \epsilon\frac{\partial U(q(t))}{\partial q}$$

$$q(t+\epsilon) = q(t) + \epsilon\frac{dq(t+\epsilon)}{dt} = q(t) + \epsilon\frac{\partial K(p(t+\epsilon))}{\partial p} = q(t) + \epsilon p(t+\epsilon)$$

These are the standard Eurler's approximations and can be unstable since the discrete update for p and q at time $(t+\epsilon)$ depends on the current state of both q and p at the same time. Let's simulate this to see what the approximation is like for our running example. Recall that here we know the true solution. In the figure below, I simulate 20 steps of the discretization for different values of ϵ. We see that the dicrete approximate gets worse as ϵ increases and for small ϵ, although the discretization is accurate, the particle does not progress much farther in time.



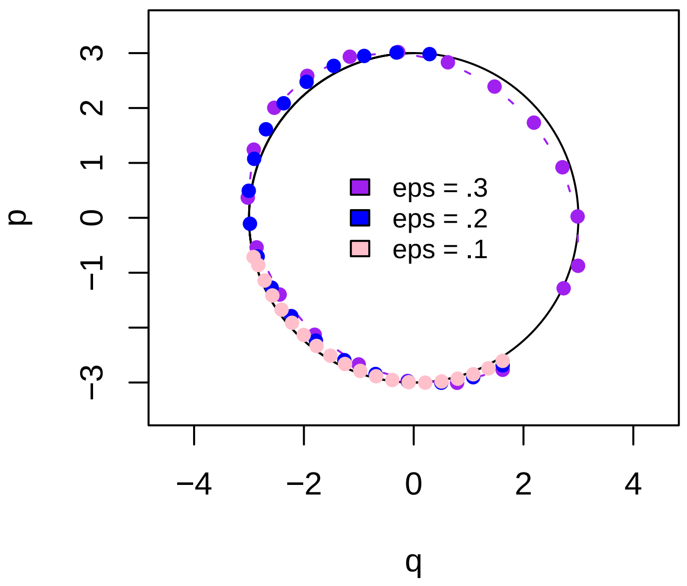
The leap frog integrator scatters the update sequence so that information on p is $\epsilon/2$ time units ahead of information on q. This ensure some stability since more information is being fed into the position at time $(t+\epsilon)$. Specifically, one jump of the leap frog integrator first does an $\epsilon/2$-Euler jump for p, followed by an ϵ-Euler jump of q using the *most updated state of* p, with one last $\epsilon/2$-Euler jump of p.

1. $p(t+\epsilon/2) = p(t) - \dfrac{\epsilon}{2}\dfrac{\partial U(q(t))}{dq}$
2. $q(t+\epsilon) = q(t) + \epsilon\dfrac{p(t+\epsilon/2)}{m}$
3. $p(t+\epsilon) = p(t+\epsilon/2) - \dfrac{\epsilon}{2}\dfrac{\partial U(q(t+\epsilon))}{dq}$.

The above does an ϵ time unit update of the particle. Naturally, we want the time update of the particle (s) to be large, but the linear approximation breaks down for large ϵ. Thus instead, one simulates multiple ϵ jumps, particularly L jumps of size ϵ, so that $s = L\epsilon$. This is done particularly with the following steps:

1. $p(t+\epsilon/2) = p(t) - \dfrac{\epsilon}{2}\dfrac{\partial U(q(t))}{dq}$
2. $q(t+\epsilon) = q(t) + \epsilon\dfrac{p(t+\epsilon/2)}{m}$
3. $p(t+3\epsilon/2) = p(t+\epsilon/2) - \epsilon\dfrac{\partial U(q(t+\epsilon))}{dq}$
4. $\vdots$
5. $q(t+L\epsilon) = q(t+(L-1)\epsilon) + \epsilon\dfrac{p(t+(2L-1)\epsilon/2)}{m}$
6. $p(t+L\epsilon) = p(t+(2L-1)\epsilon/2) - \dfrac{\epsilon}{2}\dfrac{\partial U(q(t+L\epsilon))}{dq}$.

I simulate the above Leap Frog integrator for 20 steps and you will notice that for the same values of ϵ as above, the leap frog approximation to the (q,p) curve is much better than Euler's approximation.



Using the above Leap Frog integrator, yields a mapping $\tilde{T}_{L,\epsilon}(q,p)$. Define $\tilde{P}_{L,\epsilon}(q,p) = i(\tilde{T}_{L,\epsilon}(p,q))$. Then$\tilde{P}_{L,\epsilon}$ is an involution and it is volume preserving. Essentially, using the leap frog integrator, we are able to simulate the Hamiltonian dynamics approximately, when analytical solutions are not available. The resulting $(q^*, p^*) = \tilde{P}_{L,\epsilon}(q_k, p)$ does not conserve energy *exactly*, and this implies that the acceptance ratio is not 1. As before, flipping the sign is not important in practice.

The final in-practice algorithm is then:

1. Draw $p \sim N(0, m)$, the density of which I denote as $\phi(p)$.
2. Calculate $(q^*, p^*) = \tilde{P}_{L,\epsilon}(q_k, p)$, determined by progressing the particle L jumps of size ϵ of the leap frog integrator. Flipping the sign is optional and irrelveant in practice.
3. Calculate ratio:

$$r(q_k, q^*) = \exp\{-H(q^*, p^*) - H(q_k, p)\}$$

4. If $U \leq \min\{1, r(q_k, q^*)\}$ then $q_{k+1} = q^*$
5. Else, $q_{k+1} = q_k$.

As long as the gradient of the log-target is available to us, this can be simulated simply, just like the MALA algorithm

A Complete Run for $N(0,1)$

For the $N(0,1)$ example, we will implement the final above algorithm, with the leap frog integrator. Notice that now, there are three tuning parameters: L, ϵ, m. For now, we set $m = 1$, and play around with L and ϵ.

First note that we require the gradient of the negative log likelihood, which in this case is q.

```r
normalLF_HMC <- function(L = 10, eps = .1, n = 1e4)
{
  qt <- numeric(length = n)
  qt[1] <- 0 # starting here
  accept <- 1

  # vectorizing this outside to save time
  momentums <- rnorm(n)

  for(k in 2:n)
  {
    # new momentum
    p <- momentums[k]
    q_prop <- qt[k-1]

    # one half-Euler step
    p_prop <- p - eps/2 * q_prop
    for(l in 1:L) # let the frog leap!
    {
      q_prop <- q_prop + eps * p_prop
      if(l != L) p_prop <- p_prop - eps*q_prop
    }
    # one last half-Euler step
    p_prop <- p_prop - eps/2 * q_prop

    # Accept-reject
    log.ratio <-  (-q_prop^2 - p_prop^2 + qt[k-1]^2 + p^2)/2
    if(log(runif(1)) < log.ratio)
    {
      qt[k] <- q_prop
      accept <- accept + 1
    } else{
      qt[k] <- qt[k-1]
    }
  }
  print(paste("Acceptance = ", accept/n))
  return(qt)
}
```

We will now implement this keeping the value of $s = L\epsilon$ steady at 1. This allows us to see the impact of ϵ.

```r
#Keeping L * epsilon = 1
chain1 <- normalLF_HMC(L = 10, eps = .1)
```

```
[1] "Acceptance =  0.9995"
```

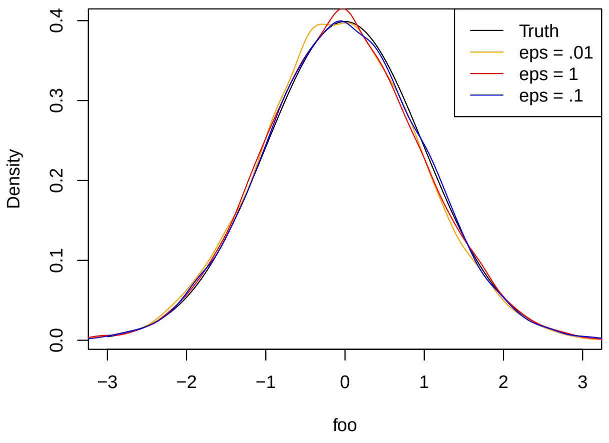
```
chain2 <- normalLF_HMC(L = 1, eps = 1)
```

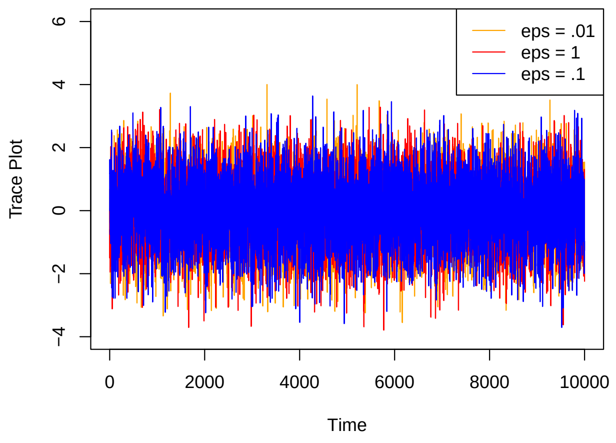
```
[1] "Acceptance =  0.9209"
```

```
chain3 <- normalLF_HMC(L = 100, eps = .01) # more time consuming
```

```
[1] "Acceptance =  0.9999"
```

Usually, one chooses L and ϵ so as to get a desired $s = L\epsilon$. However, if in this choice ϵ is small, we get a high acceptance rate (since the linear approximation is more accurate), at the cost of a high L, which requires more time. If ϵ is large, this is more time-efficient, at the cost of the linear approximation, reducing the acceptance rate. Since s is kept small in all the above runs, the chains are of reasonable quality. This of course, is a simple problem, and in most situations, it would be unclear what s should be, and how good a typical ϵ-linear approximation will be.





Setting s to be slightly larger so that we can break this down:

```
#Keeping L * epsilon = 10
chain1 <- normalLF_HMC(L = 100, eps = .1) # more time consuming
```

```
[1] "Acceptance =  0.9997"
```

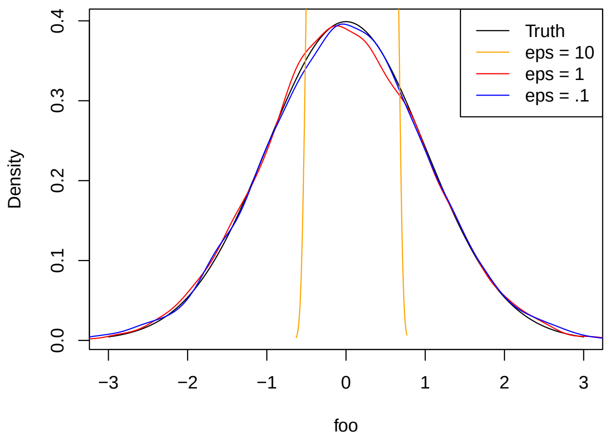
```
chain2 <- normalLF_HMC(L = 10, eps = 1)
```

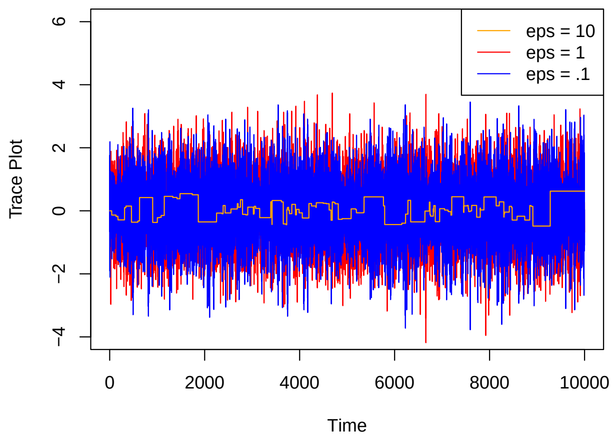
```
[1] "Acceptance =  0.9216"
```

```
chain3 <- normalLF_HMC(L = 1, eps = 10) #cheap but inaccurate
```

```
[1] "Acceptance =  0.0116"
```

Now since ϵ is allowed to be really large, the linear approximation is horrid, and the acceptance rate is abysmal.





Final Thoughts

For a few years now, I had been unable to quite understand the specifics of the HMC algorithm. The last time I had a physics course was when I was 17 years old. The last time I saw a course on PDEs, I was skipping college classes. A roadblock had formed in my head, which I was unable to get over.

The real insight came from understanding HMC as an instance of the MHGJ algorithm. Andrew Holbrook's online talk was what prompted this connection. Further, I had a Math graduate student explain Hamiltonian Dynamics to me like I was a kindergartener. Both these things together brought about a significant change in my understanding.

In this document I have omitted the theoretical underpinnings of the HMC algorithm. Concepts like invariance and ergodicity essentially follow from the MHGJ algorithm (although one has to be a little careful sometimes to obtain ergodicity). Details on this are available in Radford Neal's Handbook chapter.

Tuning HMC algorithms can be a pain sometimes, since one needs to choose L, ϵ, m. Of course, in higher dimensions, this m is a covariance matrix, and presents its own challenges. I hope having read this "introduction", the various resources on tuning and studying HMC algorithms are now more palatable and less fear-inducing. At least I walk away slightly more confident in my understanding of HMC.

References

"Dr. Andrew Holbrook's Lecture on Hamiltonian Monte Carlo (HMC)." n.d. JSB UCLA. https://www.youtube.com/watch?v=Byr9JVBI9cU.

Geyer, Charles J. 2003. "The Metropolis-Hastings-Green Algorithm." Technical Report.

Neal, Radford M et al. 2011. "MCMC Using Hamiltonian Dynamics." *Handbook of Markov Chain Monte Carlo* 2 (11): 2.