Patrick Ding, James Dole, Naveed Merchant

Abstract: ¶

Hamiltonian Monte Carlo (HMC) is a MCMC method used to sample from intractable priors. Rate of convergence to the stationary distribution and mixing is compared to Gibbs sampling and Metropolis-Hastings algorithms. Tuning parameters of HMC are explored.

Introduction:

Hamiltonian Monte Carlo (HMC, Hybrid Monte Carlo) is an MCMC method for sampling from an intractable posterior. HMC improves upon the Metropolis-Hastings algorithm by reducing the autocorrelation of the Markov chain generated. As such, convergence is attained more quickly, reducing necessary computational time.

The Hamiltonian of a physical system is the following:

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

Where K is the kinetic energy and U is the potential energy of the system, written in terms of momentum(p) and position (q). The following two first order differential equations are satisfied:

$$\frac{dp}{dt} = -\frac{\partial H}{\partial q} \qquad \frac{dq}{dt} = \frac{\partial H}{\partial p}$$

Now consider the following joint density

$$h(p,q) = k * exp(-H)$$

Where k is a normalization constant. It can easily be shown that p and q are independent by noting that the kinetic energy depends only on p, while the potential energy depends only on q.

$$\begin{aligned} H(q,p) &= K(p,q) + U(q) \\ H(q,p) &= K(p) + U(q) \\ \pi(q,p) &= e^{-K(p)-U(q)} \\ &= e^{-K(p)}e^{-U(q)} \end{aligned}$$

The obvious choice for potential energy is performing a negative log transform of the density f that we wish to sample from.

$$U = -log(f)$$

As p and q are independent variables the marginal density of h is ,

$$h(q) = exp(-U) = exp(-(-log(f)) = f$$

Note that intractable posteriors are simply probability densities multiplied by an unknown normalization constant. Therefore the log transform has no domain issues and the potential energy approaches infinity as the density approaches zero. If the density is exactly 0 at some value, the potential energy is considered infinite).

Our choice of kinetic energy is the translational, non-relativistic kinetic energy,

$$K = 0.5m * v^2$$

Other choices of kinetic energy may be used, for instance we could add a rotational component to the kinetic energy as well (so the physical system equivalent would be analogous to a ball rolling up and down hills as opposed to a frictionless mass sliding. However, the additional term is unnecessary and makes calculations more complicated for no gain.

Now, given a starting location and momentum, it is possible to calculate the time evolution of the system (using the two first order differential equations). The system is deterministic, and after some time, we can calculate the new q and the new p. First we update momentum using half of a time step. Then we alternate updating position and momentum using full time steps. On the final update of momentum, we update with a final half time step. This guarantees that updating $q(t)$ to $q(t+1)$, we will use $p(t+0.5)$, a rough estimate of the average of p between times t and t+1. This is known as the leapfrog algorithm.

According to Hamilton's equations, the change of momentum depends on position, and the change of position depends on momentum. If we iterate this system using a certain number of time steps, an error is induced and the total energy may increase or decrease. Using the leapfrog algorithm helps reduce the magnitude of this error, but will not entirely remove it. If we make some minor adjustments (in the next paragraph we learn nothing actually has to change), we may make a Metropolis-Hastings step and accept or reject the new position as another sample from the intractable prior.

In order to ensure that the Metropolis-Hastings step is legitimate and our Markov chain converges to a stationary distribution, it is sufficient to show our Markov chain is reversible. To guarantee time reversibility of a physical system, the transition operator must reverse the momentum after the last step of the leapfrog algorithm (Figure WHATEVER THE FIGURE IS THAT HAS A PHASE SPACE DIAGRAM). However, since our kinetic energy is a function of the square of p, rather than just p, no changes are necessary in the code.

ADD SOME STUFF ABOUT MH STEP. WHY DO WE ACCEPT BASED ON ENERGY?

To convince yourself that changing the sign of momentum at the end of the time transition guarantees time reversibility, imagine throwing a perfectly elastic ball from a height such that the ball hits the floor at the end of the time transition. After a second time transition, the ball is where it was originally thrown from, but travelling upwards with the initial speed of the throw. Negating the momentum again brings us back to the starting state.

Methods:

So our goal is to try to implement different MCMC methods on different data, and see how fast HMC, Gibbs, and MH (Metropolis-Hastings) can get samples, and the quality of the samples they obtain.

To examine the “quality of the sample” that a MCMC algorithm produces, we look to examine their effective sample size. The effective sample size indicates how many independent draws we’ve made, using our MCMC algorithm, which uses dependent draws. To first make sure our algorithm works soundly, we examine how well our chains do on simulated data.

We first do this on the conjugate normal-normal model. The model is conjugate so we know what the posterior mean and variance are, and we can check our chains to make sure they’re all behaving appropriately.

Normal-normal model

$$
\begin{aligned}
x_1 \dots x_n | \mu &\sim N(\mu, 1) \\
\mu &\propto 1 \\
\mu | x_1 \dots x_n &\sim N(\bar{x}, 1/n)
\end{aligned}
$$

Our effective size for the normal normal model when we used HMC was around 824 samples, while our effective size when we used MH was around 330 sample sizes.

The amount of time it took to run the HMC algorithm on average is 1381.025 milliseconds. The MH algorithm took on average around 306.7682 milliseconds

We're not surprised HMC had higher effective size and took longer, but in this case, it feels like MH is more efficient at getting effective samples. If we had run MH for 3 times longer, we would have a higher effective size then HMC while taking less time to run.

Multivariate Normal:

We wanted to see if moving into higher dimensions would change the effectiveness of any of the samplers. So to do this we simulate our data so that they are drawn from a 21 dimensional multivariate normal distribution. We then try to infer the parameters of the original simulated data. We have 20 different means changing constantly so we're monitoring 20 chains in some sense.

Our effective sample sizes are as follows for the different samplers:

Method	Effective Sample Size[1]
MH	115
Gibbs	10013
HMC	93305

The median time it took to draw from MH 10000 times is 263 milliseconds. The median time it took to draw 10000 times from HMC is 1.48 seconds. Gibbs sampling 10000 samples had a median time of 334 milliseconds.

We only have 10000 samples, but HMC seems to give an effective sample size that is much greater than that. We suspect this is because our variables are dependent but have negative autocorrelation. As a result, sampling the dependent variable multiple times is better than sampling an independent variable the same number of times.

Our effective sample sizes normalized with respect to time are as follows for the different samplers:

Method	Effective Sample Size / Time (s)
MH	436
Gibbs	29954
HMC	63006

HMC is the clear best choice for this model.

Normal-Inverse Gamma

We examine a Normal - inverse gamma model next. It has less parameters then our multivariate normal example, but our concern was that our original normal-normal model was too simple, and wanted to see if changing how we draw our variances affects which sampler is optimal.

The effective size results are as follows:

Method	Effective Sample Size
MH	1380.9
Gibbs	8999.0
HMC	4698.4

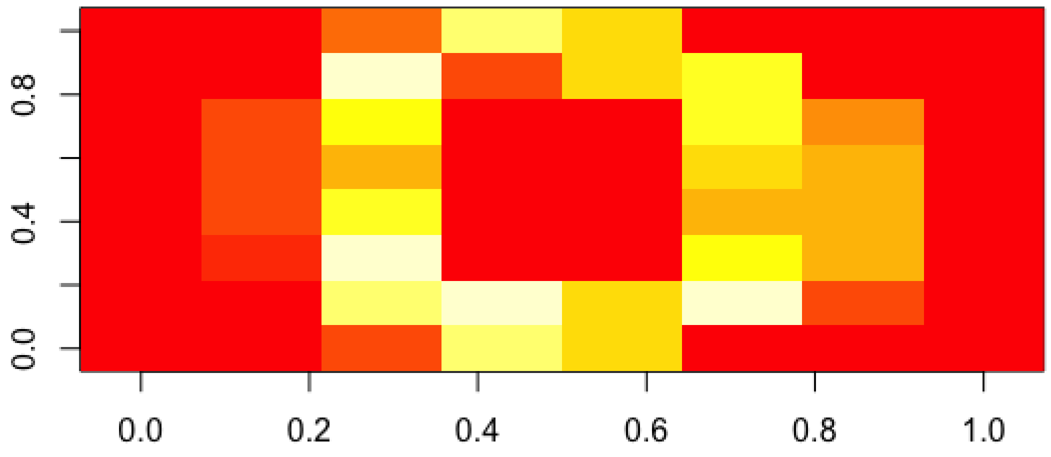
The time it took for each sampler to run is:

So the Gibbs is more efficient then either MH or HMC in this case. It takes less time

Digit Data

So after observing how Gibbs, MH, and HMC interact with simulated data, we thought it was a good idea to try this with real data.

Using the sklearn 8 x 8 MNIST dataset, we obtain very small pictures (they are of size 8 x 8 pixels) of images that are labeled as an integer from 0 to 9. For each image, we observe “how dark” a pixel is, which is a number from 0 to 16, with 0 being white and 16 being black. There are 1797 total images. Here is an example of a zero:

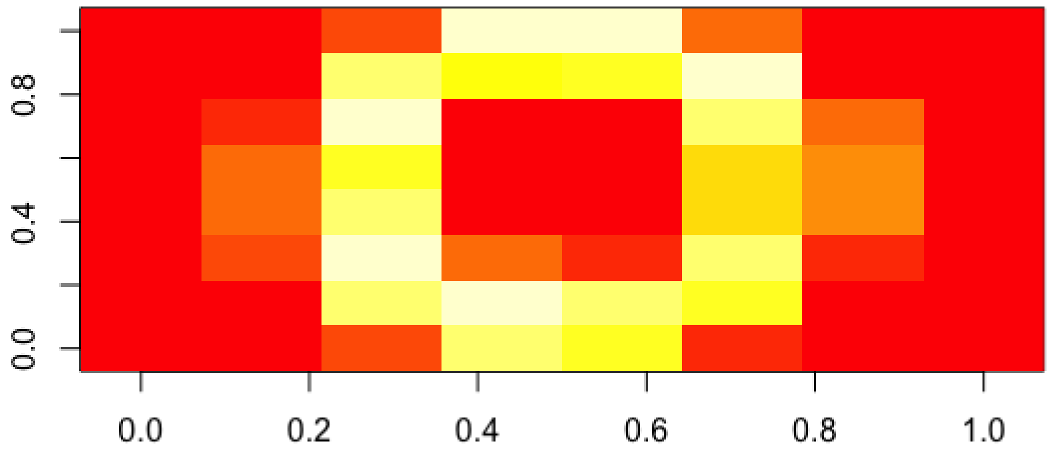


We decide to fit a naive-bayes model. Each image can be represented equivalently as an 8 x 8 matrix that takes some values from 0 to 16. Each image is an integer from 0 to 9, so we imagine that for each image, there are 9 different multivariate normal distributions that they could be like, one for each integer. We don't know the mean vector or covariance matrix, so we use our MCMC methods to try to get an idea for what they are.

In trying to compute the variance matrix, we realize that the number of parameters might be too large to compute, and in an attempt to speed up our code, we decide to change our model so that we no longer assume a general covariance matrix, but rather assume that the variance of pixels for an image are independent and identically distributed. So all the pixels in a particular class or that are a particular integer should have common variance and be independent, and each integer has a different variance. We realize this may not be entirely true, but doing this allows us to go from observing ten matrices that are of size 64 x 64 to observing only 10 variances overall, which speeds up computation heavily.

We give a training dataset, and have the computer get estimates for the means and variances for each class by using Gibbs, MH, and HMC. We then use our estimates with a testing set. We compute the log likelihood of obtaining any class using our estimates, then we have the computer guess the digit its tested on as the one that has the highest likelihood.

This is the posterior mean for the zero class, fit through HMC:



One way to test how well our samplers do, is to examine how accurate we were under different MCMC methods.

We run each method for roughly a thousand iterations and examine accuracy and run times to get an idea how each MCMC method did at producing estimates. We noticed while running HMC for the previous simulations that it could be very difficult to tune to get good mixing, but when it was that it gave great results. Tuning HMC felt much more tedious and difficult then tuning MH, so we used greta to perform HMC.

The accuracy for each sampler was:

	Accuracy
MH.Accuracy	0.86
Gibbs.Accuracy	0.89
HMC.Accuracy	0.89

The run time for each sampler was:

	Seconds
MH	7.31
Gibbs	5.96
HMC	143.71

So the Gibbs sampler and the HMC sampler have similar accuracy, but the HMC sampler take much longer to run. The MH sampler is very fast, but is less accurate then both the HMC and the Gibbs sampler.

We acknowledge that the HMC took a very long time to run, but we're using greta to tune and to create graphs

We suspect this is because our model is very simple and the conditionals are easy to sample from. If the conditionals are more complicated or are harder to draw from we imagine the the time it takes to run Gibbs to increase.

Concluding Remarks

The benefits of using HMC instead of MH in efficiently gaining effective size over time is more visible in higher dimensions than lower ones. For our Normal-Normal base model, MH was more efficient in gaining effective size over time then HMC, but for our 20 normal mixture model, HMC was much better.

```
In [ ]:
```