28 November 2017 by Richard

Markov Chains: Why Walk When You Can Flow?

***Abstract**: If you are still using a Gibbs sampler, you are working too hard for too little result. Newer, better algorithms trade random walks for frictionless flow.*

In 1989, Depeche Mode was popular, the first version of Microsoft Office was released, large demonstrations brought down the wall separating East and West Germany, and a group of statisticians in the United Kingdom dreamed of Markov chains on the desktop. In 1997, they succeeded, with the first public release of a Windows implementation of **Bayesian inference Using Gibbs Sampling**, **BUGS**.

David Hasselhoff celebrating, we must assume, BUGS

BUGS started a revolution in efficient, desktop Bayesian computation. New algorithms now make BUGS obsolete. It retires with honors. It has done more than any other initiative to promote and advance applied Bayesian data analysis.

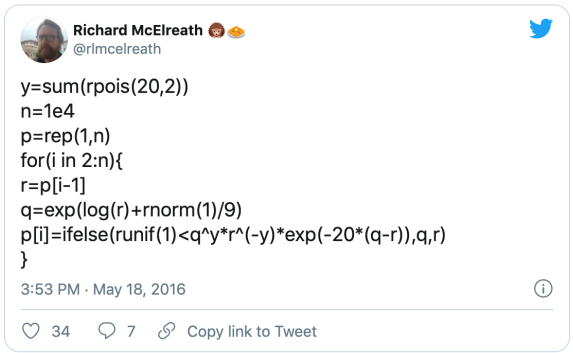
The point of this post isn't to introduce Markov chains. Markov chains don't really need an introduction. They are commonplace inference engines. Instead, the point is to nudge everyone away from Gibbs Sampling and other first-generation samplers. The serious scientific problem is that inferior samplers—custom Metropolis-Hastings monsters—still walk the corridors of scientific journals, like zombies. Better algorithms are already available. We are living in the future now, and continued use of older, fragile algorithms means that we are working too hard to produce inferior inferences.

This is not just something for quants to worry about. Every practicing scientist and analyst must take inference seriously. Whether your work is funded by the public or answers to stock holders, it isn't okay to use the second-best approach.

Here's the outline of the argument. First, I'll re-introduce Metropolis-Hastings, the algorithm behind Gibbs Sampling and similar Markov chain algorithms. I assume most readers have at least heard of it. Instead of mathematical rigor, I'll animate the approach so that the reader can appreciate both why it works and why it cannot scale with our inferential ambitions. Second, I'll introduce Hamiltonian Monte Carlo, a very different approach to constructing Markov chains. Again, the goal is not to be mathematically precise, but to animate the algorithm and show why it works and why it still has limits. I provide links to more mathematical treatments at the end.

Metropolis, Hastings, and the Random Walk

The simplest and least reliable way of building a Markov chain is the **Metropolis-Hastings algorithm**. This is the algorithm that I always teach first, because it is so simple that it can fit inside a single (old school 140 character) tweet:

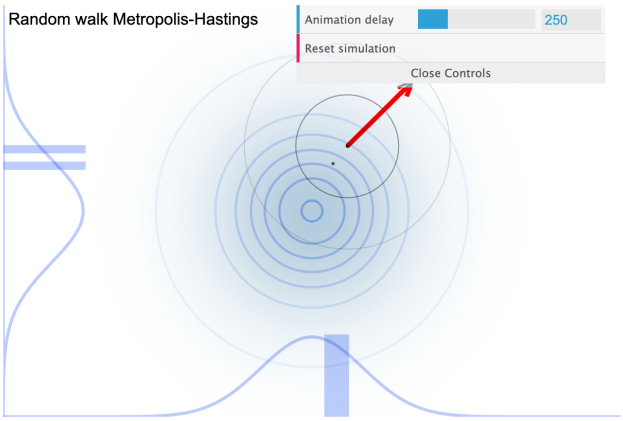


If you paste that code into R, you'll get samples from a perfectly good Markov chain.

These algorithms take samples from a target distribution by first (1) making a random proposal for new parameter values and then (2) accepting or rejecting the proposal. If both steps are done right, then the accepted parameter values will comprise samples from the target distribution.

This is easier to see than to understand. Below, I embed the MCMC simulations

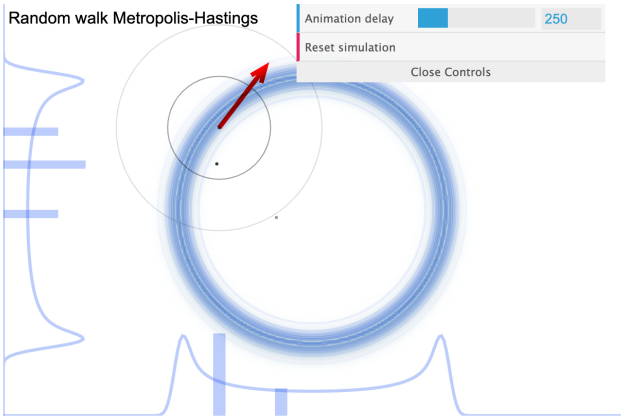
written by Chi Feng, found here. The target distribution is a benign two-dimensional Gaussian—a nice Gaussian hill. You are looking down on it, with its peak in the center. The Markov chain wanders around this hill, making random proposals to move away from its current position. These proposals are represented by the arrows. Green arrows are accepted proposals. The chain moves to the new location. Red arrows are rejections. No motion.



As samples build up, the target distribution takes shape, as seen in the histograms on the bottom and left margins. The algorithm works, even though it blindly stumbles around the target, doing a kind of random walk.

And that is exactly the problem. A major problem is that Metropolis-Hastings is, well, a bit too random. So it spends a lot of time re-exploring the same parts of the target, and unless it is tuned just right, it will also reject a lot of proposals (the red arrows), wasting computer time.

Here's another simulation, this time with a harder target: a donut. The donut target might look weird, but it represents a very common target, by analogy. In high dimensions—once there are many parameters—the target distribution occupies a narrow ring (in high-dimensional space). Most of the probability mass is not near the center. Now look what happens to honorable Metropolis-Hastings:



Notice how the Markov chain tends to get stuck in specific regions of the donut. Not only that, but it rejects a lot of proposals (the red arrows), so it wastes a lot of computer time doing nothing. Given enough time, it can explore the entire target. But it might take a very long time indeed. When we don't know what the target looks like in the first place, this kind of behavior is not only annoying, but also hazardous to inference.

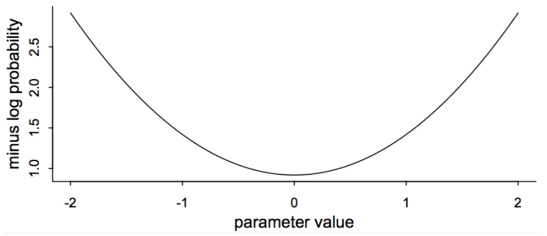
The fundamental problem with Metropolis-Hastings, and with Gibbs-Sampling as a special case, is that it is just too random. In simple targets, that isn't so bad. But in even moderately complex targets, it means inference often isn't reliable. It tends to get stuck in narrow regions of the target. There must be a better way.

Better Living Through Physics

Can your Markov chain do this?

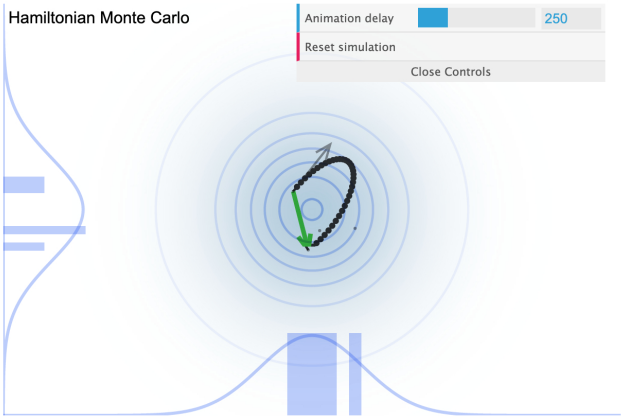
If there's a random way to do something, there's usually a less random way that is both better and requires more thought. Instead of making random proposals, suppose instead that you run a physics simulation. This is going to sound crazy, but it isn't. Your vector of parameters is now a particle in n-dimensional space. The surface in this space is a giant n-dimensional bowl. The shape of the bowl is determined by the shape of the

logarithm of the target distribution. If the target is a nice Gaussian, for example, then the log-Gaussian is a smooth parabolic bowl like this (in one-dimension):



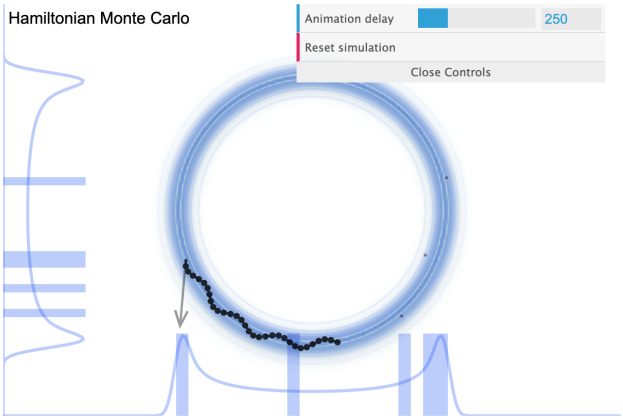
To make things a little crazier, suppose that this surface is frictionless. Now what we do is flick the particle in a random direction. It will frictionlessly flow across the bowl, eventually turning around. If we take samples of the particle's position along its path, before flicking it off in another random trajectory, then we can learn about the shape of the surface.

This is the principle behind **Hamiltonian Monte Carlo**. It will be easier to see it in action. Here is another simulation, this time using Hamiltonian Monte Carlo, again on the two-dimensional Gaussian target. The paths are flicks of the particle, and the green arrows again represent accepted proposals.



ow the proposals are both within the high-probability region of the target—so many fewer proposals are rejected—and the proposals can get far away from their starting point, so that the chain efficiently explores the whole shape of the target in less time. Effectively, it flows across the target and maps out its whole shape much faster.

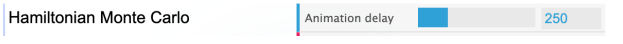
The cost of all this elegance is needing more information about the target. Hamiltonian Monte Carlo does a lot more calculation. But it also needs fewer samples to get a good image of the target. Where this really counts is with the donut. Let's revisit it:

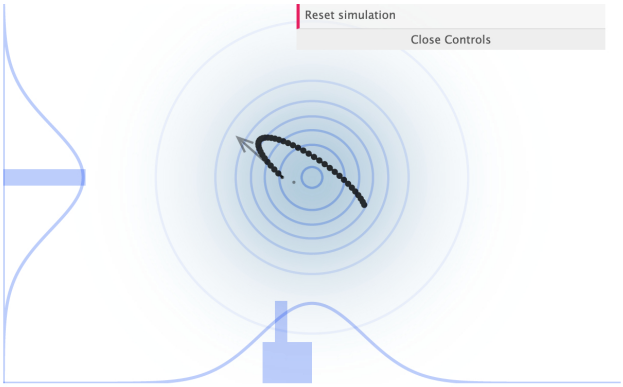


Now instead of getting stuck, the chain sweeps around the target. Even though all the chain knows at any moment in time is the local shape of the target—it can't see the whole distribution like you can here—it still manages to glide smoothly around it. This shouldn't be so amazing—a ball doesn't know the shape of the surface it rolls on, yet its path is governed by it.

Stan is NUTS

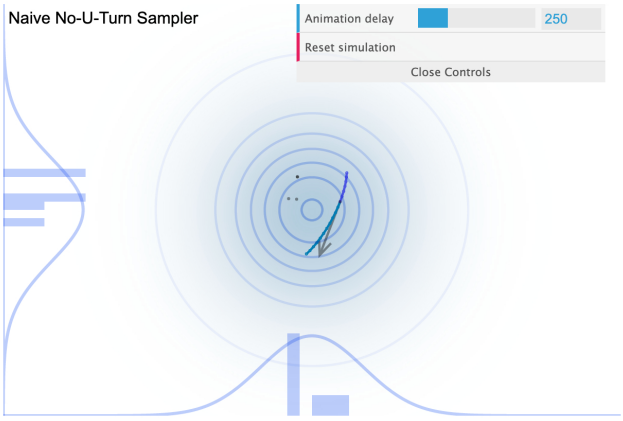
There are still some improvements to be had. Hamiltonian Monte Carlo needs to be told how many steps to take in its simulated paths. The step number determines how long the path continues before a new flick is made in a new random direction. If it takes too few steps, then it ends up with samples too similar to one another. If it takes too many, it can also end up with samples too similar to one another. Why? Because the path eventually makes a U-turn. Here's an example.





The path goes on long enough that it often eats its own tail—it makes an unfortunate U-turn. The algorithm still works, but it isn't very efficient, because it again explores in local spaces. We can of course tune the number of steps by hand, but that is not so easy when the target distribution is complex.

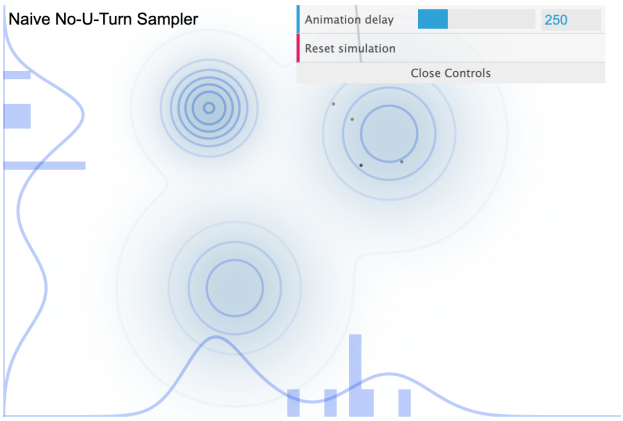
The **No U-Turn Sampler** (NUTS) is an approach for adaptively finding a good number of steps. The NUTS algorithm tries to figure out when the path starts to turn around. In order to do this efficiently, it needs to simulate the path in both directions. It looks pretty weird:



Notice how the path grows in both directions. This is the algorithm figuring out when the path turns around. When the path starts to turn around, NUTS stops the simulation and takes a sample. Then it flicks the particle in another random direction and starts another simulation. There are lots of little adaptive nudges in these algorithms that help them explore the target more efficiently. An implementation like Stan (mc-stan.org) uses an advanced version of NUTS that is even slicker.

Problems Remain

Hamiltonian algorithms still have limitations. Some targets are still hard to explore. Here's a example: a multimodal target. While the paths do a good job exploring each lump of probability mass, they have trouble transitioning among them.



Targets like this are not so unusual. They arise in many classification and latent variable models. With some clever coding, you can collapse some modes together. But you have to realize what is going on, first. Other issues arise with models that contain very steep changes in log-probability. Wizards are working on solutions to these problems. But even before solutions are found, Hamiltonian samplers are typically much better than Gibbs zombies.

Discrete Parameters

One aspect of Hamiltonian Monte Carlo that keeps some people from adopting it is

that it requires continuous parameter spaces—discrete parameters are not allowed. Discrete parameters are parameters that can only take, for example, integer values. These are useful for a wide variety of state-based models.

Hamiltonian samplers can sample from such models, however. It just takes a little more work in thinking about the target distribution. This topic really needs its own post. I've written one here. An advantage of using Hamiltonian Monte Carlo for such models is that they often end up sampling much better than going the easy way and using Gibbs. I think another advantage is that it forces the user to understand the probability model better. This is not a small benefit, in my experience.

Read More

The BUGS project's history is summarized in: Lunn, Spiegelhalter, and Best. (2009). "The BUGS project: Evolution, critique and future directions." doi:10.1002/sim.3680

Hamiltonian Monte Carlo was originally called "Hybrid Monte Carlo": Duane, Kennedy, Pendleton, and Roweth (1987) "Hybrid Monte Carlo". doi:10.1016/0370-2693(87)91197-X

The No U-Turn Sampler (NUTS) was first described by Hoffman and Gelman (2011) "The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo." arxiv.org/abs/1111.4246

Michael Betancourt's "Conceptual Introduction to Hamiltonian Monte Carlo" is only slightly technical and worth your time. arxiv.org/abs/1701.02434