**Markov Chain Monte Carlo**

*Markov chain Monte Carlo (MCMC) is a technique for estimating by* ***simulation*** *the expectation of a statistic in a* ***complex model****. Successive random selections form a Markov chain, the stationary distribution of which is the target distribution. It is particularly useful for the evaluation of posterior distributions in complex Bayesian models. In the Metropolis–Hastings algorithm, items are selected from an arbitrary “proposal” distribution and are retained or not according to an acceptance rule. The Gibbs sampler is a special case in which the proposal distributions are conditional distributions of single components of a vector parameter. Various special cases and applications are considered.*

**The Problem is Drawing from a Distribution**

Markov Chain Monte Carlo is a technique to solve the problem of ***sampling from a complicated distribution****.*Let me explain by the following imaginary scenario. Say I have a magic box which can estimate probabilities of baby names very well. I can give it a string like “Malcolm” and it will tell me the exact probability p_{\textup{Malcolm}} that you will choose this name for your next child. So there’s a distribution D over all names, it’s very specific to your preferences, and for the sake of argument say this distribution is **fixed** and you don’t get to tamper with it.

Now comes the problem: I want to ***efficiently****draw* a name from this distributionD. This is the problem that Markov Chain Monte Carlo aims to solve. Why is it a problem? Because I have no idea what process you use to pick a name, so I can’t simulate that process myself. Here’s another method you could try: generate a name xuniformly at random, ask the machine for p_x, and then flip a biased coin with probability p_xand use xif the coin lands heads. The problem with this is that there are exponentially many names! The variable here is the number of bits needed to write down a name n = |x|. So either the probabilities p_x will be exponentially small and I’ll be flipping for a very long time to get a single name, or else there will only be a few names with nonzero probability and it will take me exponentially many draws to find them. Inefficiency is the death of me.

So this is a serious problem! Let’s restate it formally just to be clear.

**Definition (The sampling problem):**Let D be a distribution over a finite setX. You are given black-box access to the probability distribution function p(x) which outputs the probability of drawing x \in X according to D. Design an efficient randomized algorithm Awhich outputs an element of X so that the probability of outputting x is approximately p(x). More generally, output a sample of elements from Xdrawn according to p(x).

Assume that A has access to only fair random coins, though this allows one to efficiently simulate flipping a [biased coin of any desired probability](https://jeremykun.com/2014/02/12/simulating-a-biased-coin-with-a-fair-coin/).

Notice that with such an algorithm we’d be able to do things like estimate the expected value of some random variable f : X \to \mathbb{R}. We could take a large sample S \subset Xvia the solution to the sampling problem, and then compute the average value of fon that sample. This is what a Monte Carlo method does when sampling is easy. In fact, the Markov Chain solution to the sampling problem will allow us to do the sampling *and* the estimation of \mathbb{E}(f)in one fell swoop if you want.

But the core problem is really a sampling problem, and “Markov Chain Monte Carlo” would be more accurately called the “Markov Chain Sampling Method.” So let’s see why a Markov Chain could possibly help us.

**Random Walks, the “Markov Chain” part of MCMC**

Markov Chain is essentially a fancy term for a random walk on a graph.

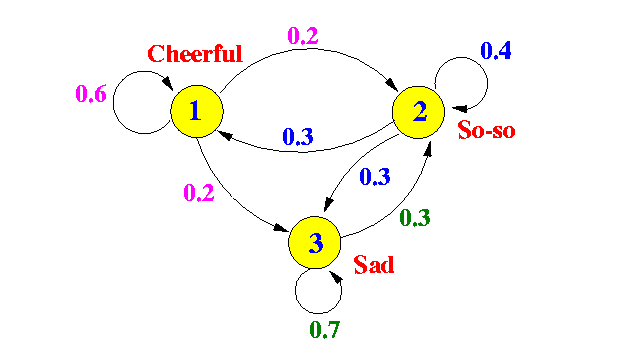
You give me a directed graph G = (V,E), and for each edge e = (u,v) \in Eyou give me a number p_{u,v} \in [0,1]. In order to make a random walk make sense, the p_{u,v} need to satisfy the following constraint:

For any vertex x \in V, the set all values p_{x,y} on outgoing edges (x,y)must sum to 1, i.e. form a probability distribution.

If this is satisfied then we can take a random walk on Gaccording to the probabilities as follows: start at some vertex x_0. Then pick an outgoing edge at random according to the probabilities on the outgoing edges, and follow it to x_1. Repeat if possible.

I say “if possible” because an arbitrary graph will not necessarily have any outgoing edges from a given vertex. We’ll need to impose some additional conditions on the graph in order to apply random walks to Markov Chain Monte Carlo, but in any case the idea of randomly walking is well-defined, and we call the whole object (V,E, \{ p_e \}_{e \in E})a *Markov chain.*

Here is an example where the vertices in the graph correspond to emotional states.

[](https://jeremykun.files.wordpress.com/2015/02/markov01.gif)

In statistics land, they take the “state” interpretation of a random walk very seriously. They call the edge probabilities “state-to-state transitions.”

The main theorem we need to do anything useful with Markov chains is the stationary distribution theorem (sometimes called the “Fundamental Theorem of Markov Chains,” and for good reason). What it says intuitively is that for a very long random walk, the probability that you end at some vertex vis independent of where you started! All of these probabilities taken together is called the *stationary distribution*of the random walk, and it is uniquely determined by the Markov chain.

However, for the reasons we stated above (“if possible”), the stationary distribution theorem is not true of every Markov chain. The main property we need is that the graph Gis *strongly connected.* Recall that a directed graph is called connected if, when you ignore direction, there is a path from every vertex to every other vertex. It is called *strongly connected* if you still get paths everywhere when considering direction. If we additionally require the stupid edge-case-catcher that no edge can have zero probability, then strong connectivity (of one component of a graph) is equivalent to the following property:

For every vertex v \in V(G), an infinite random walk started at v will return to vwith probability 1.

In fact it will return infinitely often. This property is called the *persistence*of the state vby statisticians. I dislike this term because it appears to describe a property of a vertex, when to me it describes a property of the connected component containing that vertex. In any case, since in Markov Chain Monte Carlo we’ll be picking the graph to walk on (spoiler!) we will ensure the graph is strongly connected by design.

Finally, in order to describe the stationary distribution in a more familiar manner (using linear algebra), we will write the transition probabilities as a matrix Awhere entry a_{j,i} = p_{(i,j)}if there is an edge (i,j) \in Eand zero otherwise. Here the rows and columns correspond to vertices of G, and each *column* i forms the probability distribution of going from state ito some other state in one step of the random walk. Note Ais the transpose of the weighted adjacency matrix of the directed weighted graph Gwhere the weights are the transition probabilities (the reason I do it this way is because matrix-vector multiplication will have the matrix on the left instead of the right; see below).

This matrix allows me to describe things nicely using the language of linear algebra. In particular if you give me a basis vector e_iinterpreted as “the random walk currently at vertex i,” then Ae_igives a vector whose j-th coordinate is the probability that the random walk would be at vertex jafter one more step in the random walk. Likewise, if you give me a probability distribution qover the vertices, then Aqgives a probability vector interpreted as follows:

If a random walk is in state iwith probability q_i, then the j-th entry of Aqis the probability that after one more step in the random walk you get to vertex j.

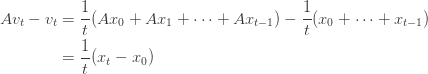
Interpreted this way, the stationary distribution is a probability distribution \pi such that A \pi = \pi, in other words \piis an eigenvector of Awith eigenvalue 1.

A quick side note for avid readers of this blog: this analysis of a random walk is exactly what we did back in the early days of this blog when we studied [the PageRank algorithm](https://jeremykun.com/2011/06/12/googles-pagerank-introduction/) for ranking webpages. There we called the matrix A“a web matrix,” noted it was column stochastic (as it is here), and appealed to a special case of the Perron-Frobenius theorem to show that there is a unique maximal eigenvalue equal to one (with a dimension one eigenspace) whose eigenvector we used as a sort of “stationary distribution” and the final ranking of web pages. There we described an algorithm to actually find that eigenvector by iterated multiplication by A. The following theorem is essentially a variant of this algorithm but works under weaker conditions; for the web matrix we added additional “fake” edges that give the needed stronger conditions.

**Theorem:**Let Gbe a strongly connected graph with associated edge probabilities \{ p_e \}_e \in Eforming a Markov chain. For a probability vector x_0, define x_{t+1} = Ax_tfor all t \geq 1, and let v_tbe the long-term average v_t = \frac1t \sum_{s=1}^t x_s. Then:

1. There is a unique probability vector \piwith A \pi = \pi.
2. For all x_0, the limit \lim_{t \to \infty} v_t = \pi.

*Proof.* Since v_tis a probability vector we just want to show that |Av_t - v_t| \to 0as t \to \infty. Indeed, we can expand this quantity as

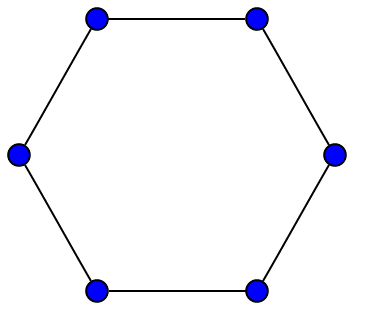


But x_t, x_0are unit vectors, so their difference is at most 2, meaning |Av_t - v_t| \leq \frac2t \to 0. Now it’s clear that this does not depend on v_0. For uniqueness we will cop out and appeal to the Perron-Frobenius theorem that says any matrix of this form has a unique such (normalized) eigenvector.

\square

One additional remark is that, in addition to computing the stationary distribution by actually computing this average or using an eigensolver, one can analytically solve for it as the inverse of a particular matrix. Define B = A-I_n, where I_nis the n \times nidentity matrix. Let Cbe Bwith a row of ones appended to the bottom and the topmost row removed. Then one can show (quite opaquely) that the last column of C^{-1}is \pi. We leave this as an exercise to the reader, because I’m pretty sure nobody uses this method in practice.

One final remark is about why we need to take an average over all our x_tin the theorem above. There is an extra technical condition one can add to strong connectivity, called *aperiodicity*, which allows one to beef up the theorem so that x_titself converges to the stationary distribution. Rigorously, aperiodicity is the property that, regardless of where you start your random walk, after some sufficiently large number of steps n the random walk has a positive probability of being at every vertex at every subsequent step. As an example of a graph where aperiodicity fails: an undirected cycle on an even number of vertices. In that case there will only be a positive probability of being at certain vertices every *other* step, and averaging those two long term sequences gives the actual stationary distribution.



One way to guarantee that your Markov chain is aperiodic is to ensure there is a positive probability of staying at any vertex. I.e., that your graph has a self-loop. This is what we’ll do in the next section.

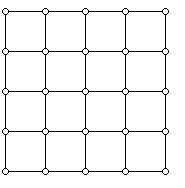
**Constructing a graph to walk on**

Recall that the problem we’re trying to solve is to draw from a distribution over a finite set Xwith probability function p(x). The MCMC method is to construct a Markov chain whose stationary distribution is exactly p, even when you just have black-box access to evaluating p. That is, you (implicitly) pick a graph Gand (implicitly) choose transition probabilities for the edges to make the stationary distribution p. Then you take a long enough random walk on Gand output the xcorresponding to whatever state you land on.

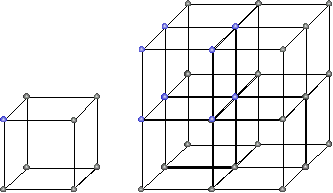
The easy part is coming up with a graph that has the right stationary distribution (in fact, “most” graphs will work). The hard part is to come up with a graph where you can prove that the convergence of a random walk to the stationary distribution is fast in comparison to the size of X. Such a proof is beyond the scope of this post, but the “right” choice of a graph is not hard to understand.

The one we’ll pick for this post is called the **Metropolis-Hastings** algorithm. The input is your black-box access to p(x), and the output is a set of rules that implicitly define a random walk on a graph whose vertex set is X.

It works as follows: you pick some way to put X on a lattice, so that each state corresponds to some vector in \{ 0,1, \dots, n\}^d. Then you add (two-way directed) edges to all neighboring lattice points. For n=5, d=2it would look like this:

[](https://jeremykun.files.wordpress.com/2015/02/2dlattice.jpg)

And for d=3, n \in \{2,3\}it would look like this:

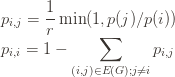
[](https://jeremykun.files.wordpress.com/2015/02/lattice.gif)

You have to be careful here to ensure the vertices you choose for Xare not disconnected, but in many applications Xis naturally already a lattice.

Now we have to describe the transition probabilities. Let rbe the maximum degree of a vertex in this lattice (r=2d). Suppose we’re at vertex i and we want to know where to go next. We do the following:

1. Pick neighbor jwith probability 1/r(there is some chance to stay at i).
2. If you picked neighbor jand p(j) \geq p(i)then deterministically go to j.
3. Otherwise, p(j) < p(i), and you go to jwith probability p(j) / p(i).

We can state the probability weight p_{i,j}on edge (i,j)more compactly as



It is easy to check that this is indeed a probability distribution for each vertex i. So we just have to show that p(x)is the stationary distribution for this random walk.

Here’s a fact to do that: if a probability distribution v with entries v(x)for each x \in X has the property that v(x)p_{x,y} = v(y)p_{y,x}for all x,y \in X, the vis the stationary distribution. To prove it, fix xand take the sum of both sides of that equation over all y. The result is exactly the equation v(x) = \sum_{y} v(y)p_{y,x}, which is the same as v = Av. Since the stationary distribution is the unique vector satisfying this equation, v has to be it.

Doing this with out chosen p(i)is easy, since p(i)p_{i,j}and p(i)p_{j,i}are both equal to \frac1r \min(p(i), p(j))by applying a tiny bit of algebra to the definition. So we’re done! One can just randomly walk according to these probabilities and get a sample.

**Last words**

The last thing I want to say about MCMC is to show that you can estimate the expected value of a function \mathbb{E}(f)simultaneously while random-walking through your Metropolis-Hastings graph (or any graph whose stationary distribution is p(x)). By definition the expected value of fis \sum_x f(x) p(x).

Now what we can do is compute the average value of f(x)just among those states we’ve visited during our random walk. With a little bit of extra work you can show that this quantity will converge to the true expected value of fat about the same time that the random walk converges to the stationary distribution. (Here the “about” means we’re off by a constant factor depending on f). In order to prove this you need some extra tools I’m too lazy to write about in this post, but the point is that it works.

The reason I did not start by describing MCMC in terms of estimating the expected value of a function is because the core problem is a sampling problem. Moreover, there are many applications of MCMC that need nothing more than a sample. For example, MCMC can be used to estimate the volume of an arbitrary (maybe high dimensional) convex set. See [these lecture notes](http://www.cs.berkeley.edu/%7Esinclair/cs294/n1.pdf) of Alistair Sinclair for more.

If demand is popular enough, I could implement the Metropolis-Hastings algorithm in code (it wouldn’t be industry-strength, but perhaps illuminating? I’m not so sure…).