21 Markov Chains

Markov Chains represent and model the flow of information in a graph, they give insight into how a graph is connected, and which nodes are important.

As we will see, they also provide important life lessons:

- **[L1]** *Only your current position matters going forward, don't worry about the past.*
- **[L2]** *You just need to worry about one step at a time; you will get there eventually (or you won't).*
- **[L3]** *In the limit, everyone has perfect karma.*

21.1 Review of Graphs

We start by reviewing the *abstract data type* of graphs, and their interpretation as matrices.

A graph $G = (V, E)$ is defined by a set of vertices $V = \{v_1, v_2, \ldots, v_n\}$ and a set of edges $E = \{e_1, e_2, \ldots, e_m\}$ where each edge e_j is an unordered (or ordered in a directed graph) pair of edges: $e_j = \{v_i, v_{i'}\}$.

Consider an example graph portrayed three ways.

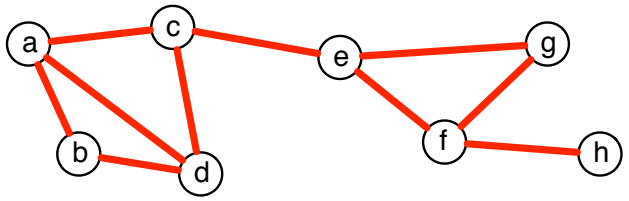
Mathematically: $G = (V, E)$ where

$$V = \{a, b, c, d, e, f, g\} \text{ and}$$
$$E = \Big\{\{a,b\},\{a,c\},\{a,d\},\{b,d\},\{c,d\},\{c,e\},\{e,f\},\{e,g\},\{f,g\},\{f,h\}\Big\}.$$

Matrix-Style: As a matrix with 1 if there is an edge, and 0 otherwise. (For a directed graph, it may not be symmetric). This is known as the *adjacency matrix*.

$$A = \begin{array}{c|cccccccc} & a & b & c & d & e & f & g & h \\ \hline a & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ b & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ c & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ d & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ e & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ f & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ g & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ h & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{array} = \begin{pmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Pictorially: A ball stick model of a graph.



21.2 Markov Chains

A *Markov chains* (V, P, q) is defined by a set of nodes V, a probability transition matrix P, and an initial state q. In some contexts q is not needed, and P is implicitly described by the associated matrix of a graph.

The initial state q represents a probability distribution over which nodes we are located. For instance, if we are at state $b \in V$ (with probability 1) then

$$q^T = [0\ 1\ 0\ 0\ 0\ 0\ 0\ 0].$$

If we have a 10% chance of being in state a, a 30% chance of being in state d and a 60% change of being in state f, then

$$q^T = [0.1\ 0\ 0\ 0.3\ 0\ 0.6\ 0\ 0].$$

In general we need to enforce that

- each $q[i] \geq 0$
- $\sum_i q[i] = 1$.

Now the *transition matrix* P can be described as the normalized adjacently matrix

$$P = \begin{pmatrix} 0 & 1/2 & 1/3 & 1/3 & 0 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 1/3 & 0 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 1/3 & 1/3 & 0 & 0 & 0 \\ 1/3 & 1/2 & 1/3 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/3 & 0 & 0 & 1/3 & 1/2 & 0 \\ 0 & 0 & 0 & 0 & 1/3 & 0 & 1/2 & 1 \\ 0 & 0 & 0 & 0 & 1/3 & 1/3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1/3 & 0 & 0 \end{pmatrix}$$

That is each row A_j of A is represented in P as the column $P_j = A_j / \|A\|_1$, after it has been normalized.

Now given a state $q^T = [0\ 1\ 0\ 0\ 0\ 0\ 0\ 0]$ can "transition" to the next state as

$$q_1 = Pq = \begin{bmatrix} \frac{1}{2} & 0 & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 \end{bmatrix}^T.$$

Then we can get to the next state as

$$q_2 = Pq_1 = PPq = P^2q = \begin{bmatrix} \frac{1}{6} & \frac{2}{6} & \frac{2}{6} & \frac{1}{6} & 0 & 0 & 0 & 0 \end{bmatrix}^T.$$

and

$$q_3 = Pq_2 = \begin{bmatrix} \frac{1}{3} & \frac{1}{9} & \frac{1}{9} & \frac{1}{3} & \frac{1}{9} & 0 & 0 & 0 \end{bmatrix}^T.$$

In general we can write $q_n = P^n q$, that is starting with q and "hitting" q on the left n times by P, the transition matrix.

This is called a "Markov" chain after Andrey Markov, because it is a *Markov process*. This means that it only depends on its current state, and nothing prior to that (unless it is implicitly encoded in the current state). This fulfills **L1**.

There are now *two* ways to think about this Markov chain process.

- It describes a *random walk* of a point starting at q (or in some position with distribution described by q). Then at each step it decides where to go next randomly based on the column of P describing the column its state corresponds to. It moves to *exactly one* new state. Then repeat.

- It describes the *probability distribution of a random walk*. At each state, we only track the distribution of where it *might* be: this is q_n after n steps. Alternatively, we can consider P^n, then for any initial state q_0, $P^n q_0$ describers the distribution of where q_0 might be after n steps. So entry $P^n_{j,i}$ (jth column, ith row) describes the probability that a point starting in j will be in state i after n steps.

Usually, only one of these two interpretations is considered. They correspond to quite different algorithms and purposes, each with their own advantages. We will talk about both, and in particular the first one shortly. ... but first some more definitions!

21.2.1 More Definitions!

A Markov chain is *ergodic* if there exists some t such that for all $n \geq t$, then each entry in P^n is positive. This means that from any starting position, after t steps there is *always* a chance we are in every state. That is, for any q, then $q_n = P^n q$ is positive in all entries.

It is important to make the distinction in the definition that it is not that we have some positive entry for *some* $n \geq t$, but for *all* $n \geq t$, as we will see.

When is a Markov chain *not* ergodic?

- It is *cyclic*. This means that it alternates between different sets of states every 2 or 3 or in general p steps. Here are some example cyclic transition matrices:

$$\begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1/2 & 1/2 & 1/2 & 1/2 & 0 \\ 1/4 & 0 & 0 & 0 & 0 & 1/4 \\ 1/4 & 0 & 0 & 0 & 0 & 1/4 \\ 1/4 & 0 & 0 & 0 & 0 & 1/4 \\ 1/4 & 0 & 0 & 0 & 0 & 1/4 \\ 0 & 1/2 & 1/2 & 1/2 & 1/2 & 0 \end{pmatrix}$$

- It has *absorbing and transient states*. (This only happens when the initial graph is *directed*, so you cannot go backwards on an edge.) In some Markov chains we can classify V into two class $A, T \subset V$ so that if a random walk leaves some node in T and lands in a state in A, then it *never* returns to any state in T. In this case, the nodes A are *absorbing*, and the nodes in T are *transient*. Here are some examples:

$$\begin{pmatrix} 1/2 & 0 \\ 1/2 & 1 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix} \qquad \begin{pmatrix} 1/2 & 1/2 & 0 & 0 & 0 & 0 \\ 1/2 & 49/100 & 0 & 0 & 0 & 0 \\ 0 & 1/100 & 1/4 & 1/4 & 1/4 & 1/4 \\ 0 & 0 & 1/4 & 1/4 & 1/4 & 1/4 \\ 0 & 0 & 1/4 & 1/4 & 1/4 & 1/4 \\ 0 & 0 & 1/4 & 1/4 & 1/4 & 1/4 \end{pmatrix}$$

- It is not *connected*. There are two sets of notes $A, B \subset V$ such that there is no possible way to transition from any node in A to any node in B. And some examples:

$$\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad \begin{pmatrix} 1/2 & 1/2 & 0 & 0 & 0 & 0 \\ 1/2 & 1/2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1/3 & 1/2 & 1/3 & 0 \\ 0 & 0 & 1/3 & 0 & 1/3 & 0 \\ 0 & 0 & 1/3 & 1/2 & 1/3 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

When it *is* ergodic. From now on, we will assume that the Markov chain *is* ergodic. At a couple of critical points we will show simple modifications to existing chains to help ensure this.

Now there is an **amazing** property that happens.

Let $P^* = P^n$ as $n \to \infty$ (it will converge). Now let $q_* = P^* q$ (this *does not* depend on the choice of q).

- That is, for all starting states q, the final state is q_* (if we run the chain long enough **L2**).
- As we do a random walk, we will eventually have an expected stated precisely described by q_*.

Also, $q_* = PP^*q$ thus $q_* = Pq_*$. Thus fulfilling **L3** since at this point, the probability of being in a state i and leaving to j, is the same as being in another state j and arriving at i (this is called the *delicate balance*). Globally, we can generalize this to say, the probability of being in any state i and leaving (to any other state) is the same as being any other state and arriving in i. Thus, if a distribution starts in $q_0 = q_*$ it is already in the final distribution. The "further" it starts (e.g. q_0 is different from q_*), the longer it takes to converge.

Moreover, q_* is the *first* eigenvector of P, after normalizing so the sum of its elements are 1. In Matlab `[V,L] = eig(P);` and let `v1 = V(:,1);` and `qstar = v1/sum(v1)` to get the stable distribution `qstar`. This *second* eigenvalue λ_2 determines the rate of convergence. The smaller λ_2, the faster the rate of convergence. In our example graph, $q_* = (0.15, 0.1, 0.15, 0.15, 0.15, 0.15, 0.1, 0.05)^T = (\frac{3}{20}, \frac{1}{10}, \frac{3}{20}, \frac{3}{20}, \frac{3}{20}, \frac{3}{20}, \frac{1}{10}, \frac{1}{20})^T$ and the second eigenvalue of P is 0.875 which indicates a kinda (but not too) slow convergence.

21.3 Metropolis Algorithm

The Metropolis Algorithm, sometimes referred to as Markov Chains Monte Carlo (MCMC) was developed by Metropolis, Rosenbluth, Rosenbluth, Teller, and Teller in 1953 to help develop the atomic bomb. There is some controversy over who really deserves credit for the invention. But, the lesson is, it pays to have a name that is both cool sounding, and earliest in alphabetical order!

This was latter generalized by Hastings (1970) and it eventually led to enormous applications in computing Bayesian statistics, as Gibbs sampling (Geman, Geman 1984 and Gelfand, Smith 1990).

Here each state $v \in V$ has a weight associated with it:

$$w(v) \qquad \text{where} \qquad \sum_{v \in V} w(v) = W.$$

More generally, V may be continuous and then $W = \int_{v \in V} w(v)\, dv$. Then we want to land in a state v with probability $w(v)/W$. But...

- V might be *very* large, and W unknown.
- V can be continuous, so there can be *no* way to calculate W. I call this a *probe-only* distribution, since you can measure $\mu(v) = cw(v)$ at any one state at a time where c is some unknown constant (related to W).

So our goal is to design a special Markov chain so $q_*[v] = w(v)/W$ (without knowing W).

The Algorithm. Start with some $v_0 \in V$ so $q = [0\ 0\ 0\ \ldots\ 1\ \ldots\ 0\ 0]^T$.

Now iterate as follows:

Choose neighbor u (proportional to $K(v, u)$) where K is some notion of neighborhood/similarity (for instance a kernel, like a Gaussian kernel). And move to u with probability $\min\{1, w(u)/w(v)\}$. See Algorithm 21.3.1.

Algorithm 21.3.1 Metropolis on V and w

Initialize $v_0 = [0\ 0\ 0\ \ldots\ 1\ \ldots\ 0\ 0]^T$.
repeat
  Generate $u \sim K(v, \cdot)$
  if $(w(u) \geq w(v_i))$ **then**
    Set $v_{i+1} = u$
  else
    With probability $w(u)/w(v)$ set $v_{i+1} = u$
  else
    Set $v_{i+1} = v_i$
until "converged"
return $V = \{v_1, v_2, \ldots, \}$

This implicitly defines a Markov chain on the state space V. The transition matrix is implicitly defined by the algorithm. And moreover, if the chain is ergodic, then there exists some t such that $i \geq t$, then $\mathbf{Pr}[v_i = v] = w(v)/W$.

NOTE: this is not just in the limit, but for some finite t (even for continuous V), through the AMAZING property called "coupling from the past". But t is hard to find.

Often the goal is to create many samples (from $\sim w$).

- Officially: run for $t+$ steps, take *one* sample,
  run for another $t+$ steps, take *one* sample,
  repeat.
- In practice: Run for 1000 steps (the "burn in" period)
  take next 5000 steps as random sample.

The second method has "auto-correlation", as samples v_i and v_{i+1} are likely to be "near" each other (either since K is local, or because it did not accept new state).
Officially, we should take on every s steps, where s depends on the degree of auto-correlation. But in practice, we take all n samples, but treat them (for purpose of bounds) as n/s samples.

Big challenge: neither t or s is known.

This is inherently sequential (at least the practical version), this makes it very hard to parallelize. Another major challenge.