Graphical Models For Complex Health Data (P8124)

Daniel Malinsky

Columbia University
dsm2128@cumc.columbia.edu

Approximate Inference
(Part II: Sampling)

Sampling

This week we'll turn our attention to a different approach to approximate inference: sampling methods (a.k.a. Monte Carlo methods a.k.a. particle-based inference). The idea is to generate samples $x^s \sim p(x|y)$ and then use these to compute any quantity of interest, e.g., a marginal/conditional $p(x_i|y)$.

Sampling from complex joint distributions is difficult. How do we do this efficiently?

Univariate case

Consider a random variable X with cdf F, and let F^{-1} be the inverse cdf. If $U \sim \text{Unif}(0,1)$ then $F^{-1}(U) \sim F$. Why?

$$\begin{aligned} P(F^{-1}(U) \leq x) &= P(U \leq F(x)) \\ &= F(x) \text{ because } P(U \leq u) = u. \end{aligned}$$

So, for any univariate distribution with known inverse cdf, we can generate random samples by using a pseudorandom number generator to sample U, i.e., get $u^1, u^2, ..., u^s$ and applying F^{-1} to each:
$x^1 = F^{-1}(u^1), x^2 = F^{-1}(u^2), ..., x^s = F^{-1}(u^s)$.

Univariate case

Consider a random variable X with cdf F, and let F^{-1} be the inverse cdf. If $U \sim \text{Unif}(0,1)$ then $F^{-1}(U) \sim F$. Why?

$$\begin{aligned} P(F^{-1}(U) \leq x) &= P(U \leq F(x)) \\ &= F(x) \text{ because } P(U \leq u) = u. \end{aligned}$$

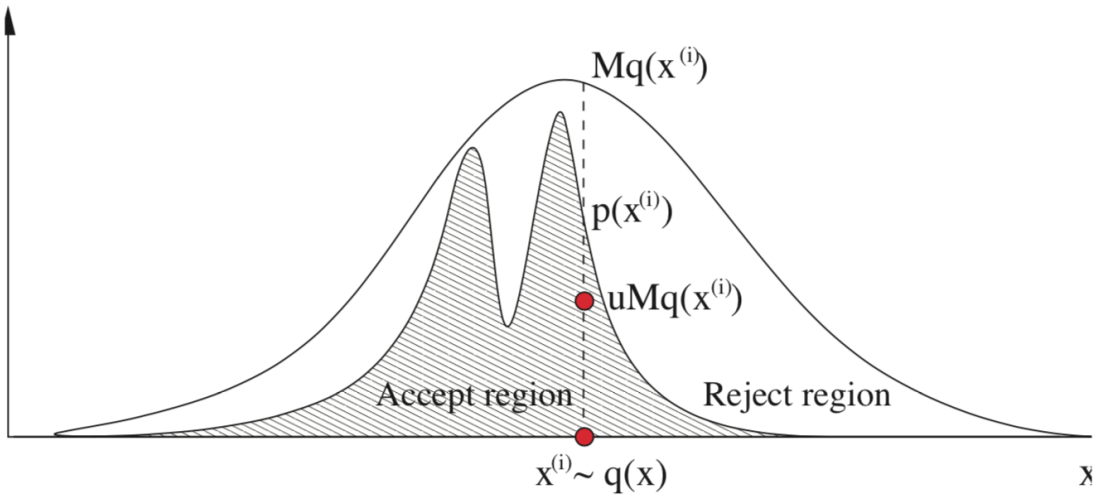
So, for any univariate distribution with known inverse cdf, we can generate random samples by using a pseudorandom number generator to sample U, i.e., get $u^1, u^2, ..., u^s$ and applying F^{-1} to each:
$x^1 = F^{-1}(u^1), x^2 = F^{-1}(u^2), ..., x^s = F^{-1}(u^s)$.

For multivariate distributions, there are similar approaches for some special cases (e.g., Box-Mueller method for multivariate Gaussian variables), but in general – it's hard!

Rejection sampling

Idea: create a proposal distribution $q(x)$ which satisfies $Mq(x) \geq \tilde{p}(x)$ for some constant M where $\tilde{p}(x)$ is an unnormalized version of $p(x)$ (e.g., $p(x) = \tilde{p}(x)/Z$ or $\tilde{p}(x) = p(x|y)p(y) = p(x, y)$). Then:

- sample $x \sim q(x)$ and sample $u \sim \text{Unif}(0, 1)$
- if $u \geq \frac{\tilde{p}(x)}{Mq(x)}$ reject the sample, otherwise accept
- collect accepted samples



Rejection sampling

Let $S = \{(x, u) : u \leq \tilde{p}(x)/Mq(x)\}$ and
$S^* = \{(x, u) : x \leq x^*, u \leq \tilde{p}(x)/Mq(x)\}$

The cdf of the accepted samples is

$$\begin{aligned} P(X = x \leq x^* | x \text{ accepted}) &= \frac{P(X = x \leq x^*, x \text{ accepted})}{P(x \text{ accepted})} \\ &= \frac{\int\int \mathbb{I}((x,u) \in S^*) q(x) du dx}{\int\int \mathbb{I}((x,u) \in S) q(x) du dx} \\ &= \frac{\int_{-\infty}^{x^*} \tilde{p}(x) dx}{\int_{-\infty}^{\infty} \tilde{p}(x) dx} = \int_{-\infty}^{x^*} p(x) dx \end{aligned}$$

which is the cdf of X.

Rejection sampling

$$
\begin{aligned}
P(X \text{ accepted}) &= P(U \leq \frac{\tilde{p}(x)}{Mq(x)}) \\
&= \int P(U \leq \frac{\tilde{p}(x)}{Mq(x)} | X = x) q(x) dx \\
&= \int \frac{\tilde{p}(x)}{Mq(x)} q(x) dx = \frac{1}{M} \int \tilde{p}(x) dx
\end{aligned}
$$

so to accept as many samples as possible (not be wasteful) we want to choose M as low as possible while satisfying $Mq(x) \geq \tilde{p}(x)$.

Rejection sampling

$$
\begin{aligned}
P(X \text{ accepted}) &= P(U \leq \frac{\tilde{p}(x)}{Mq(x)}) \\
&= \int P(U \leq \frac{\tilde{p}(x)}{Mq(x)} | X = x) q(x) dx \\
&= \int \frac{\tilde{p}(x)}{Mq(x)} q(x) dx = \frac{1}{M} \int \tilde{p}(x) dx
\end{aligned}
$$

so to accept as many samples as possible (not be wasteful) we want to choose M as low as possible while satisfying $Mq(x) \geq \tilde{p}(x)$.

It is difficult to choose M to be efficient in high dimensions. Consider approximating $X \sim N(0, \sigma_p^2 I)$ using a proposal $N(0, \sigma_q^2 I)$ (where I is a d-dimensional identity matrix). Want $\sigma_q^2 \geq \sigma_p^2$ to bound the true distribution, optimal choice is $M = (\sigma_q / \sigma_p)^d$. Acceptance rate is $1/M$. If σ_q exceeds σ_p by just 1% with $d = 1000$ we get $1/M = 1/(1.01)^{1000} \sim 1/20000$ which is very inefficient.

Importance sampling

Consider a sampling method for approximating quantities of the form $\mathbb{E}[f(x)] = \int f(x)p(x)dx$. For example, we might want to estimate the probability of a rare event so $f(x) = \mathbb{I}(x \in E)$. Sampling from all of $p(x)$ is not the most efficient strategy. Sample from some $q(x)$ and use samples to estimate

$$\mathbb{E}[f] = \int f(x)\frac{p(x)}{q(x)}q(x)dx$$

by

$$\widehat{\mathbb{E}}[f] = \frac{1}{S}\sum_{s=1}^{S} f(x^s)\frac{p(x^s)}{q(x^s)} = \frac{1}{S}\sum_{s=1}^{S} w_s f(x^s)$$

where $w_s = \frac{p(x^s)}{q(x^s)}$ are called the *importance weights*.

Optimal (min variance) choice is to sample from $q(x) \propto |f(x)|p(x)$.

NB: this uses all the samples, unlike in rejection sampling.

Importance sampling on a DAG

Given a Bayesian network model with DAG $\mathcal{G}$, we can imagine sampling from the distribution $p(x)$ as follows: first sample from the exogenous nodes (nodes with no parents), and then their children conditional on the parents, and so on (*ancestral sampling*). In other words, for each node i draw $x_i^s \sim p(x_i | \text{Pa}(X_i, \mathcal{G}) = x^s_{\text{Pa}(i)})$.

The each sample $x_1^s, ..., x_p^s$ is a sample from joint $p(x)$ under the DAG model.

Importance sampling on a DAG

If we want to combine that with some "evidence," i.e., get samples of $p(x|y)$ we could do ancestral sampling (here $\mathcal{G}$ has vertices $V = X \cup Y$) + reject values inconsistent with $Y = y$.

That would be very inefficient and wouldn't work with real-valued evidence.

Instead we use importance sampling.

Importance sampling on a DAG

Sample $X_i \in X$ variables as before (ancestral sampling) but for variables in Y use their observed values y. This is equivalent to using a proposal

$$q = \prod_{i \notin Y} p(x_i | \operatorname{Pa}(X_i, \mathcal{G}) = x_{\operatorname{Pa}(i)}) \prod_{i \in Y} \delta(y_i)$$

Which amounts to using importance weights

$$w_s = \frac{p}{q} = \prod_{i \in Y} p(y_i | \operatorname{Pa}(Y_i, \mathcal{G}) = y_{\operatorname{Pa}(i)})$$

Markov Chain Monte Carlo (MCMC)

MCMC is a very influential and common approach to approximate inference, popular because of its ability to work in high dimensions and theoretical properties. It was first discovered by physicists working on the atomic bomb during WWII, and first published by Metropolis (1953) in a chemistry journal.

The basic idea is to construct a Markov chain whose stationary distribution is the target density of interest $p(x)$.

Markov Chain

A sequence of random variables $X_1, X_2, X_3, ...$ is a Markov chain if

$$p(x_1, ..., x_T) = p(x_1) \prod_{t=1}^{T} p(x_t|x_{t-1})$$

If the transition function $p(x_t|x_{t-1})$ is assumed independent of t then the chain is called homogonous, stationary, or time-invariant.

If X_t is discrete with k states, then $p(x_t|x_{t-1})$ can be written as a $k \times k$ matrix, the transition matrix A where $A_{ij} = P(x_t = j|x_{t-1} = i)$. Each row of the matrix sums to 1: $\sum_j A_{ij} = 1$.

Markov Chain diagrams

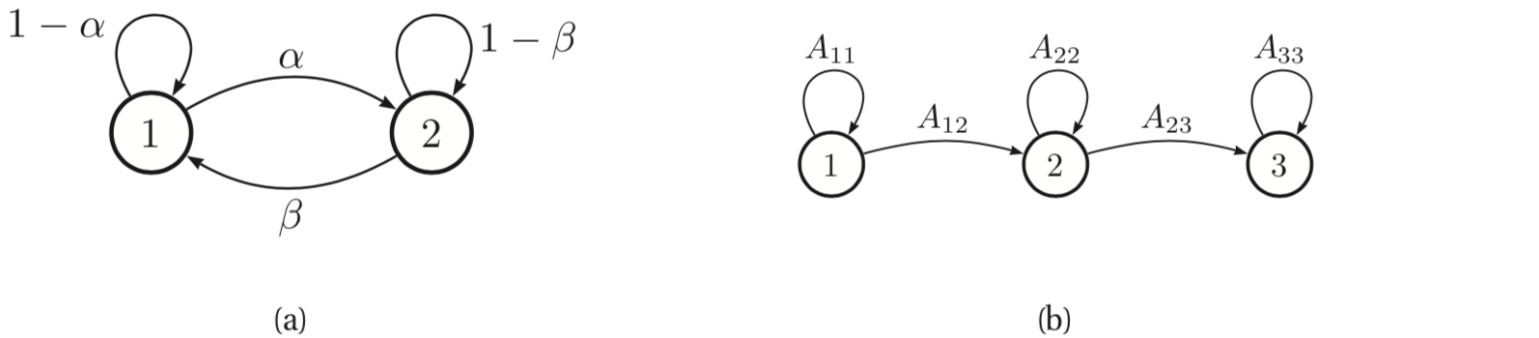


Figure 17.1 State transition diagrams for some simple Markov chains. Left: a 2-state chain. Right: a 3-state left-to-right chain.

Markov Chain

The n-step transition matrix $A(n)$ where $A_{ij}(n) = P(X_{t+n} = j | X_t = i)$ summarizing the probability of getting from i to j in n steps. $A(1) = A$.

One may derive $A(n) = A^n$.

Stationary distributions

Let $\pi_t(j) = P(X_t = j)$ be the probability of being in state j at time t. $\boldsymbol{\pi}_t$ is a row vector. Then:

$$\pi_1(j) = \sum_i \pi_0(i) A_{ij}$$

or in matrix notation

$$\boldsymbol{\pi}_1 = \boldsymbol{\pi}_0 A$$

If we iterate these equations over time and reach a stage where

$$\boldsymbol{\pi} = \boldsymbol{\pi} A$$

then we say we've reached the stationary distribution (or invariant or equilibrium distribution) of the Markov chain.

Solving for stationary distributions

Given an A we can solve for the stationary distribution.

[example on the board]

Solving for stationary distributions

Given an A we can solve for the stationary distribution.

[example on the board]

Solve the eigenvector equation $A^T v = v$, set $\pi = v^T$ where v is an eigenvector with eigenvalue 1. Then normalize to satisfy sum-to-one constraint. (Eigenvectors are only guaranteed to be real-valued if the matrix is positive. If some entries are zero, have to do a bit more work.)

When does a stationary distribution exist?

A Markov chain is called *irreducible* if its state transition diagram is a singly connected component, i.e., we can get from any state to any other state. A chain has a limiting distribution if for all j, $\pi_j = \lim_{n\to\infty} A^n_{ij}$ exists and is independent of i. (Long run is independent of the starting state.) A chain is *ergodic* if, roughly, it is aperiodic and the expected time for the chain to land on every state is finite.

Theorem. Every irreducible ergodic Markov chain has a limiting distribution, which is equal to $\boldsymbol{\pi}$, its unique stationary distribution.

How does this relate to sampling?

Use a Markov chain to go from state to state.

- sample x^0 from $p_0(x)$
- for $s = 1, ..., S$ sample x^s from $p(X = x|X = x^{s-1})$
- return $x^0, ..., x^S$

The stationary distribution is the target distribution you want to sample from.

Gibbs sampling

Idea: sample each variable in turn, conditioned on the values of all other variables in the distribution.

- start with some initial sample x^s
- $x_1^{s+1} \sim p(x_1|x_{-1}^s)$
- $x_2^{s+1} \sim p(x_2|x_1^{s+1}, x_{-(1,2)}^s)$
- $x_3^{s+1} \sim p(x_3|x_1^{s+1}, x_2^{s+1}, x_{-(1,2,3)}^s)$
- ...
- $x_p^{s+1} \sim p(x_p|x_{-p}^{s+1})$

With a graphical model, each update will only depend on the Markov blanket of variable X_i. It is necessary to discard some of the initial samples, before the process has entered its stationary distribution. Note: our samples are not iid!

Gibbs sampling example: pairwise MRF

$$p(x_i|x_{-i}) \propto \prod_{j \in \text{Ne}(i)} \phi_{ij}(x_i, x_j)$$

with $\phi_{ij} = \exp(W_{ij} x_i x_j)$ and $x_i \in \{-1, +1\}$ we have

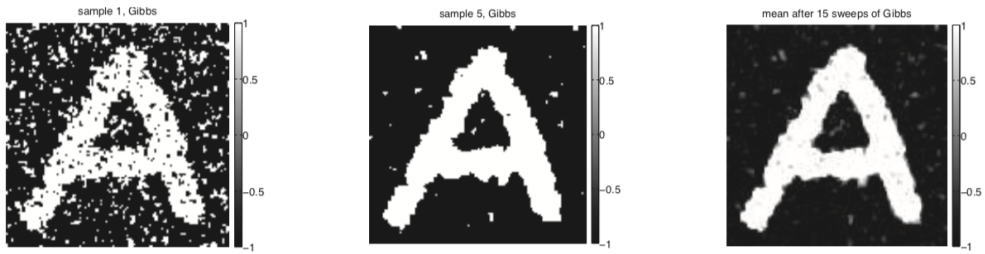
$$\begin{aligned} p(x_i = 1|x_{-i}) &= \frac{\prod_{j \in \text{Ne}(i)} \phi_{ij}(x_i = 1, x_j)}{\prod_{j \in \text{Ne}(i)} \phi_{ij}(x_i = 1, x_j) + \prod_{j \in \text{Ne}(i)} \phi_{ij}(x_i = -1, x_j)} \\ &= \frac{\exp(W_{ij} \sum_{j \in \text{Ne}(i)} x_j)}{\exp(W_{ij} \sum_{j \in \text{Ne}(i)} x_j) + \exp(-W_{ij} \sum_{j \in \text{Ne}(i)} x_j)} \\ &= \text{expit}(2W_{ij} \sum_{j \in \text{Ne}(i)} x_j) \end{aligned}$$

Gibbs sampling: denoising

We can combine with a model for the noisy observations Y_i given the true pixels X_i as we did in the denoising case $\phi_i(x_i) = p(y_i|x_i) = \exp(L_i(x_i))$ (e.g., $\mathrm{N}(x_i, \sigma^2)$) and get

$$p(x_i = 1|x_{-i}, y) = \frac{\exp(W_{ij} \sum_{j \in \mathrm{Ne}(i)} x_j + L_i(1))}{\exp(W_{ij} \sum_{j \in \mathrm{Ne}(i)} x_j + L_i(1)) + \exp(-W_{ij} \sum_{j \in \mathrm{Ne}(i)} x_j + L_i(-1))}$$

which basically matches up with what we got using mean field. However, we're using sampling here, not updating parameters of an approximating distribution q.



(a) (b) (c)

Gibbs sampling

Updating one variable at a time can be slow if the variables are highly correlated, so sometimes we update groups of variables at a time. This is called *blocking* Gibbs sampling or *blocked* Gibbs sampling.

Metropolis Hastings algorithm

Idea: specify a MC where at each step we propose to move from the current state x to a new state x' with probability $q(x'|x)$ (user-specified proposal distribution).

For example, a common choice of proposal is Gaussian centered on the current state, $q(x'|x) \propto \exp[-1/2(x'-x)^T\Sigma^{-1}(x'-x)]$. This is called *random walk Metropolis algorithm*. If we choose $q(x'|x) = q(x')$ we get an *independence sampler*.

Having proposed a move to x' we then decide whether to accept this proposal or not according to some formula, which ensures that the fraction of time spent in each state is proportional to $p(x)$. If the proposal is accepted, update the state to x' otherwise stay at x.

Metropolis Hastings algorithm

If the proposal is symmetric, so $q(x'|x) = q(x|x')$, the acceptance probability is

$$r = \min(1, \frac{p(x')}{p(x)})$$

If the proposal is asymmetric, so $q(x'|x) \neq q(x|x')$, the acceptance probability is

$$r = \min(1, \alpha)$$

$$\alpha = \frac{p(x')q(x|x')}{p(x)q(x'|x)} = \frac{p(x')/q(x'|x)}{p(x)/q(x|x')}$$

This is called the *Hastings correction*. Actually we only need the unnormalized distribution $\tilde{p}$ since normalizing constants cancel out.

Metropolis Hastings algorithm

Algorithm 24.2: Metropolis Hastings algorithm

Initialize x^0 ;

for $s = 0, 1, 2, \ldots$ **do**

  Define $x = x^s$;

  Sample $x' \sim q(x'|x)$;

  Compute acceptance probability

$$\alpha = \frac{\tilde{p}(x')q(x|x')}{\tilde{p}(x)q(x'|x)}$$

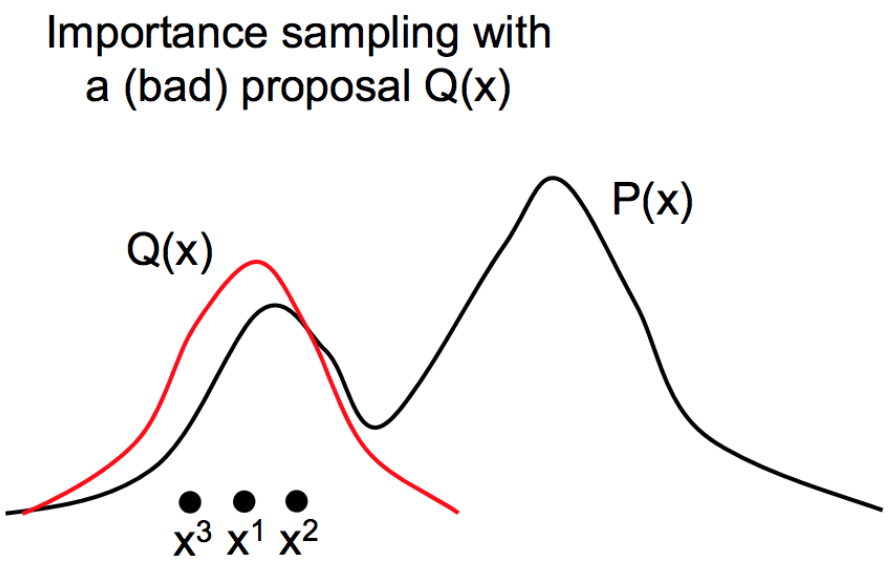
  Compute $r = \min(1, \alpha)$;

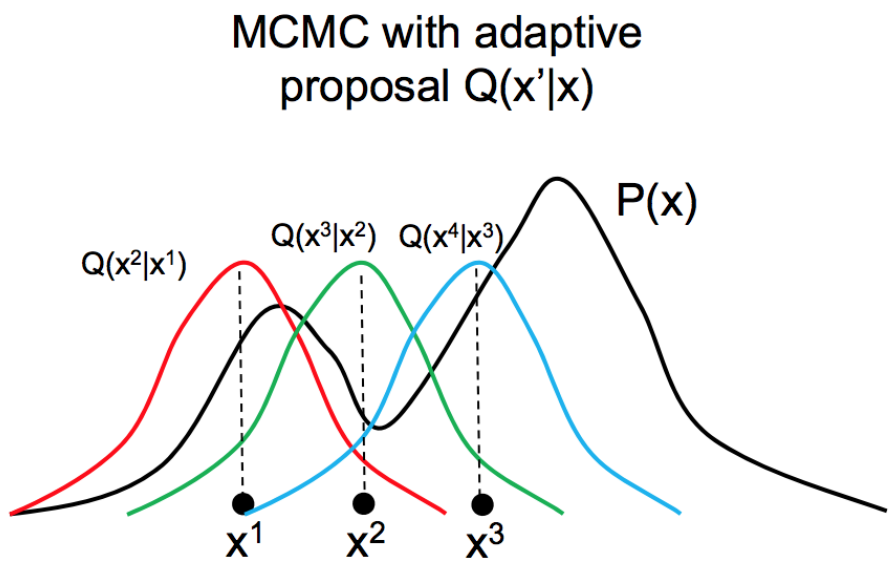
  Sample $u \sim U(0, 1)$;

  Set new sample to

$$x^{s+1} = \begin{cases} x' & \text{if } u < r \\ x^s & \text{if } u \geq r \end{cases}$$

Proposal is *adaptive*: it depends on the previously sampled state

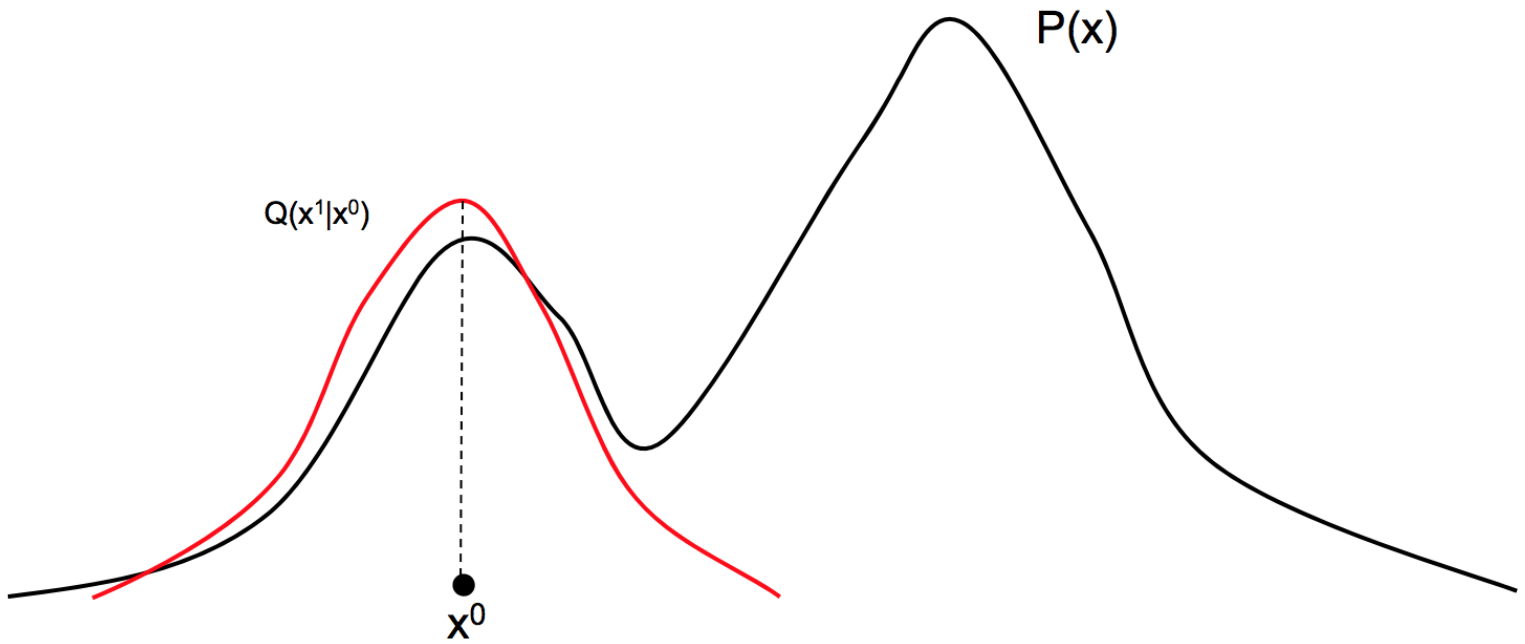




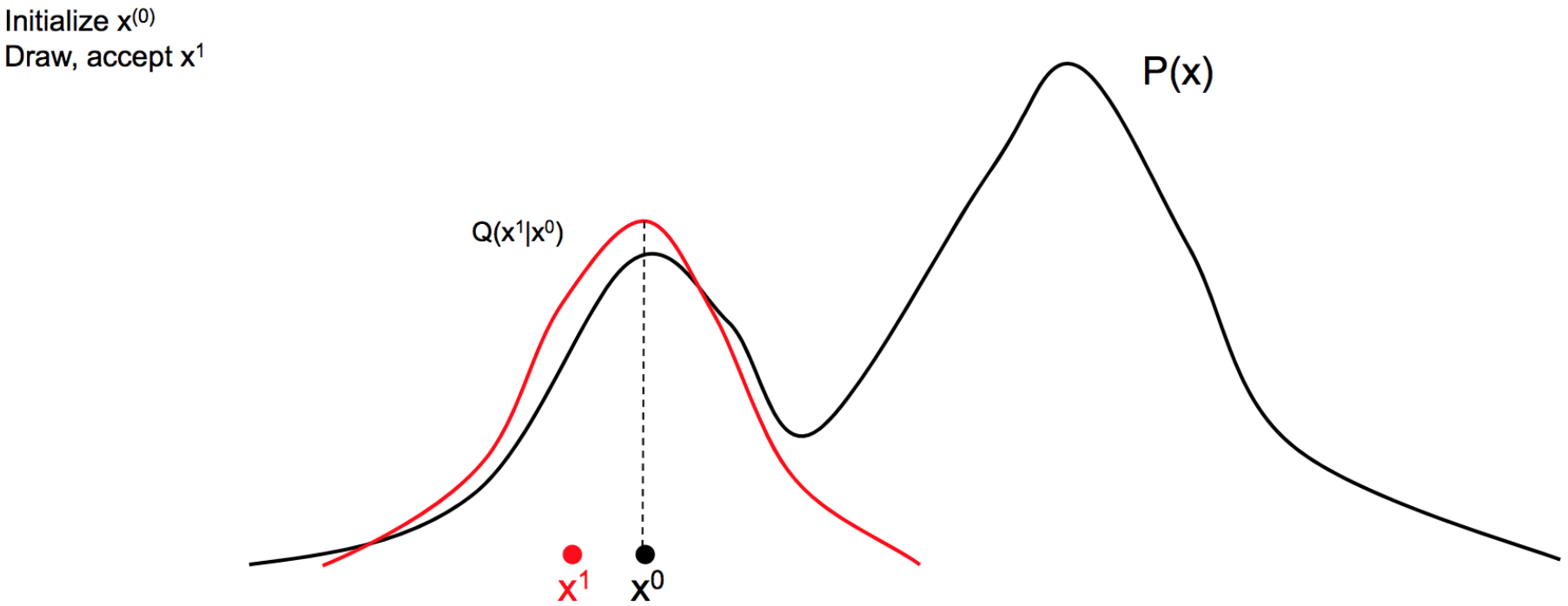
Proposal is *adaptive*: it depends on the previously sampled state

Initialize $x^{(0)}$

...

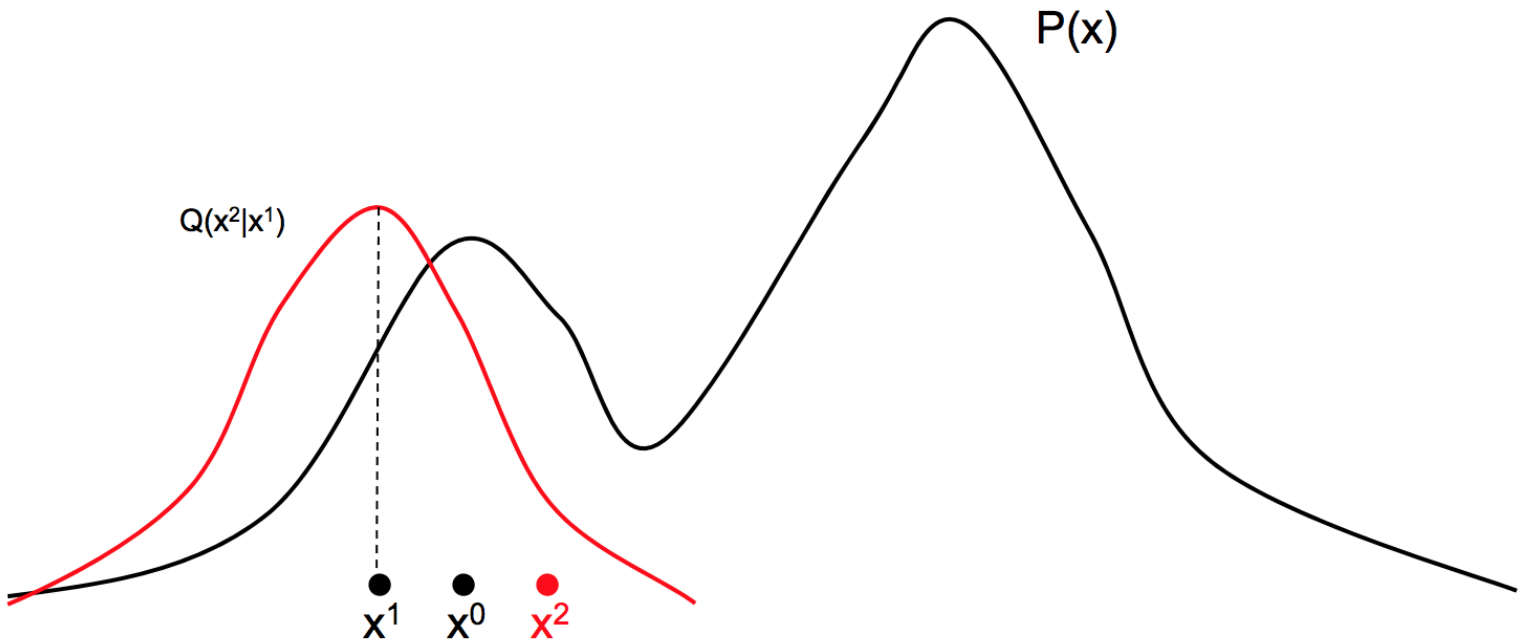


Proposal is *adaptive*: it depends on the previously sampled state



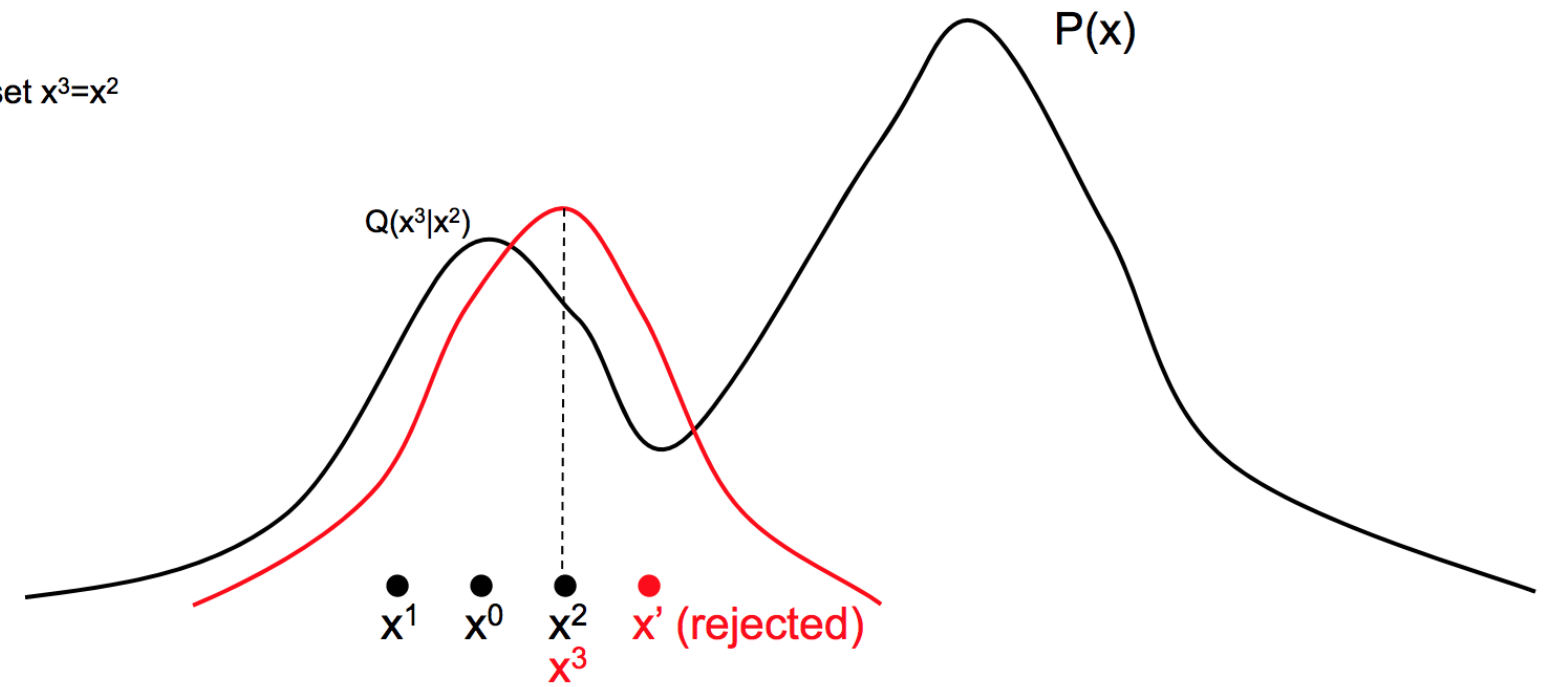
Proposal is *adaptive*: it depends on the previously sampled state

Initialize $x^{(0)}$
Draw, accept x^1
Draw, accept x^2



Proposal is *adaptive*: it depends on the previously sampled state

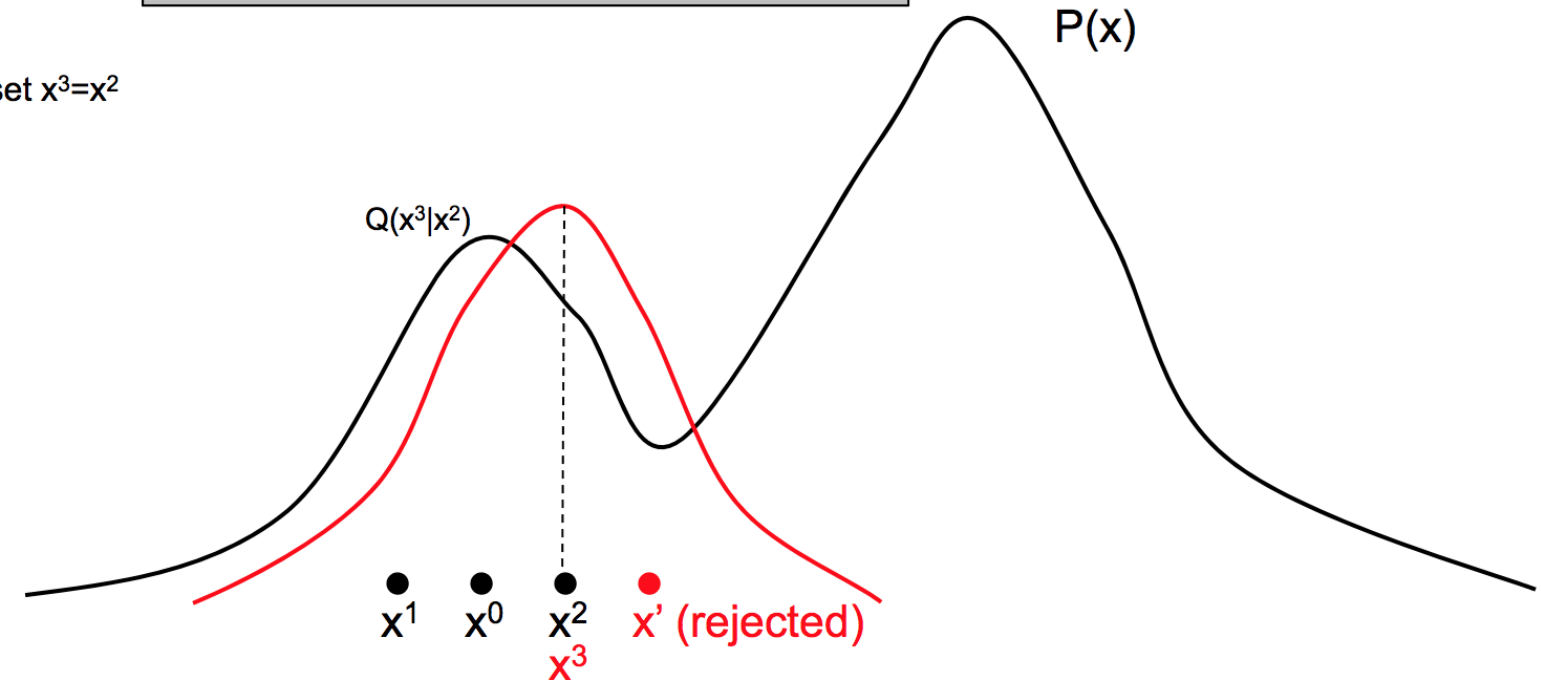
Initialize $x^{(0)}$
Draw, accept x^1
Draw, accept x^2
Draw but reject; set $x^3=x^2$



Proposal is *adaptive*: it depends on the previously sampled state

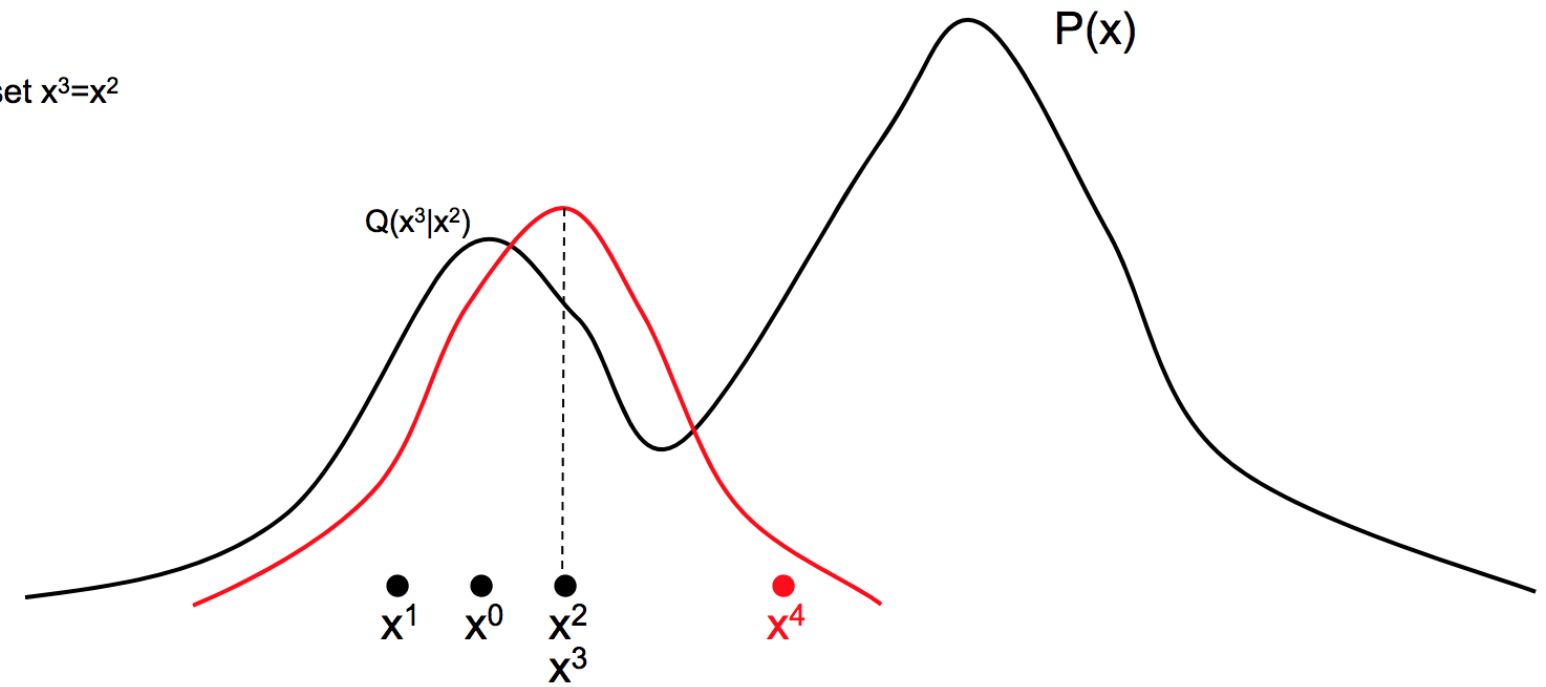
Initialize $x^{(0)}$
Draw, accept x^1
Draw, accept x^2
Draw but reject; set $x^3=x^2$

We reject because $P(x')/Q(x'|x^2) < 1$ and $P(x^2)/Q(x^2|x') > 1$, hence $A(x'|x^2)$ is close to zero!



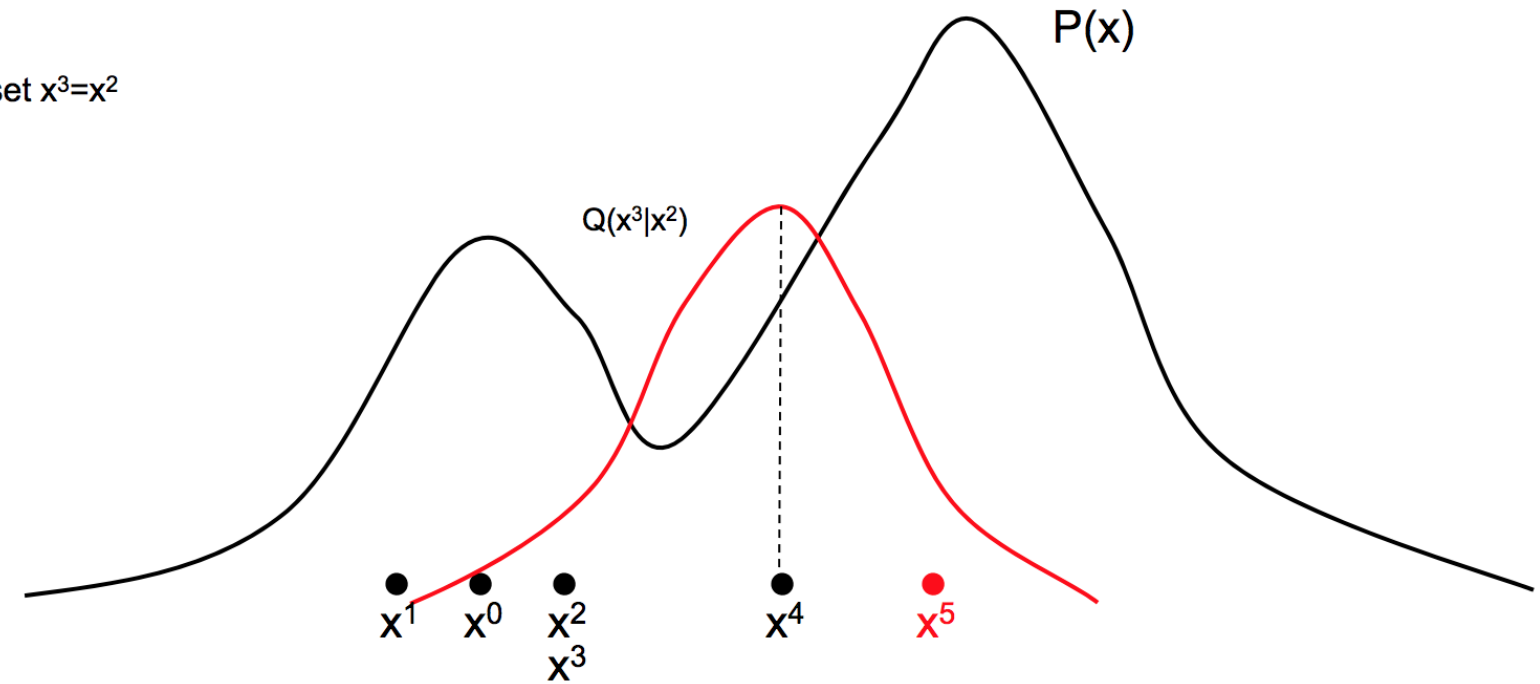
Proposal is *adaptive*: it depends on the previously sampled state

Initialize $x^{(0)}$
Draw, accept x^1
Draw, accept x^2
Draw but reject; set $x^3=x^2$
Draw, accept x^4



Proposal is *adaptive*: it depends on the previously sampled state

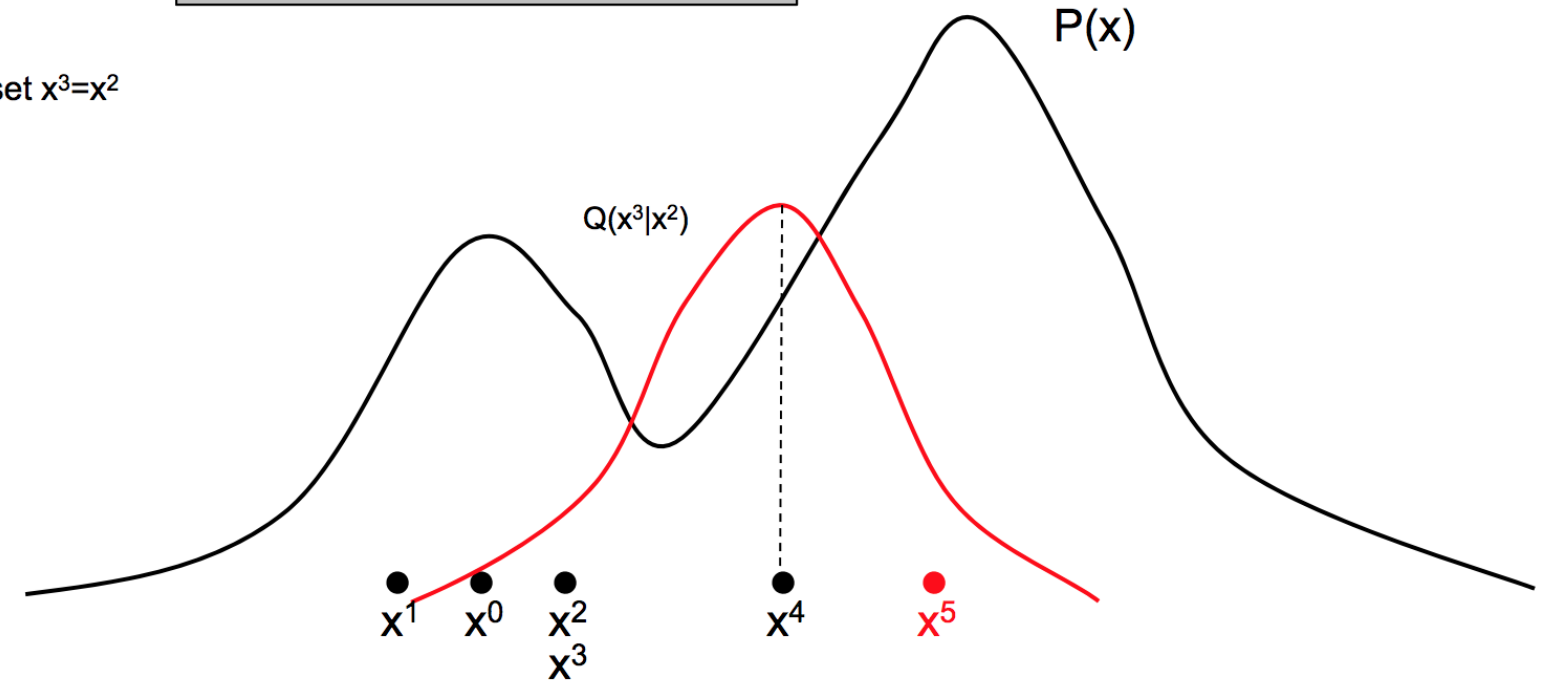
Initialize $x^{(0)}$
Draw, accept x^1
Draw, accept x^2
Draw but reject; set $x^3=x^2$
Draw, accept x^4
Draw, accept x^5



Proposal is *adaptive*: it depends on the previously sampled state

Initialize $x^{(0)}$
Draw, accept x^1
Draw, accept x^2
Draw but reject; set $x^3=x^2$
Draw, accept x^4
Draw, accept x^5

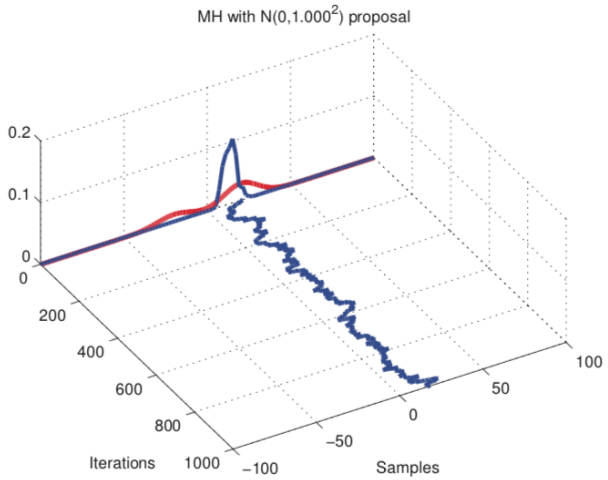
The adaptive proposal Q(x'|x) allows us to sample both modes of P(x)!



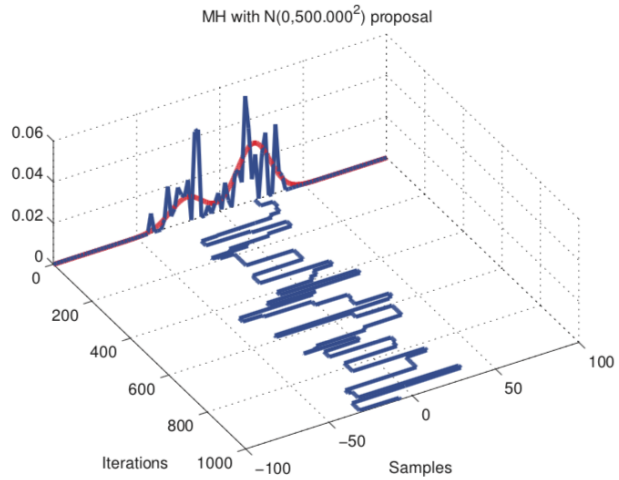
Can change proposal over time

In fact, one may make the proposal more adaptive by changing the parameters over time, for example by starting with a Gaussian q with high variance and then decreasing to lower variance after some sampling. This is called "adaptive MCMC." One may also use some feature of the data to tune the proposal over time, called "data-driven MCMC."

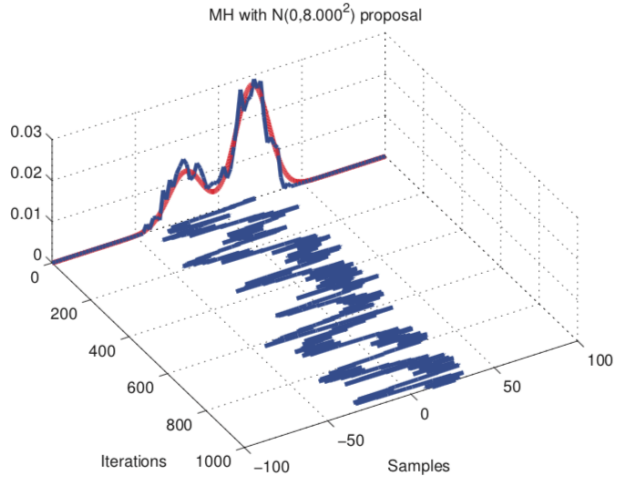
How to choose the parameters of the proposal?



(a)



(b)



(c)

How to choose the parameters of the proposal?

In practice, may try a number of short pilot runs to test different proposal parameters, and choose one with a reasonable acceptance rate, say 25-40 percent.

Stationary distribution of MH

Lemma. If a Markov chain with transition matrix A is regular and satisfies $\pi_i A_{ij} = \pi_j A_{ji}$ (detailed balance equations), then $\boldsymbol{\pi}$ is a stationary distribution of the chain.

Stationary distribution of MH

Lemma. If a Markov chain with transition matrix A is regular and satisfies $\pi_i A_{ij} = \pi_j A_{ji}$ (detailed balance equations), then $\boldsymbol{\pi}$ is a stationary distribution of the chain.

Proof. $\sum_i \pi_i A_{ij} = \sum_i \pi_j A_{ji} = \pi_j \sum_i A_{ji} = \pi_j$ which implies $\boldsymbol{\pi} = \boldsymbol{\pi} A$.

Stationary distribution of MH

Lemma. If a Markov chain with transition matrix A is regular and satisfies $\pi_i A_{ij} = \pi_j A_{ji}$ (detailed balance equations), then $\boldsymbol{\pi}$ is a stationary distribution of the chain.

Proof. $\sum_i \pi_i A_{ij} = \sum_i \pi_j A_{ji} = \pi_j \sum_i A_{ji} = \pi_j$ which implies $\boldsymbol{\pi} = \boldsymbol{\pi} A$.

The Metropolis-Hastings makes transitions from x to x' with probability

$$p(x'|x) = \begin{cases} q(x'|x) r(x'|x) & \text{if } x' \neq x. \\ q(x|x) + q(x'|x)(1 - r(x'|x)) & \text{otherwise.} \end{cases} \tag{1}$$

Theorem. If the transition matrix defined by (1) is ergodic and irreducible, then $p(x)$ is its unique limiting distribution.

Proof

Consider the case when $r(x'|x) = \min(1, \alpha) < 1$. It must be that $\alpha(x'|x) = \frac{p(x')q(x|x')}{p(x)q(x'|x)} < 1$. Then $r(x'|x) = \alpha(x'|x)$ and $r(x|x') = 1$.
This implies the transition probability
$p(x'|x) = q(x'|x)r(x'|x) = \frac{p(x')}{p(x)}q(x|x')$.
Thus,
$p(x)p(x'|x) = p(x')q(x|x')$.
The backwards transition probability is:
$p(x|x') = q(x|x')r(x|x') = q(x|x')$ since $r(x|x') = 1$.
Plugging into above we get:
$p(x)p(x'|x) = p(x')p(x|x')$ which is the detailed balance equation. So $p(x)$ is the stationary distribution. It is unique by our assumption that the process is ergodic and irreducible.

Gibbs sampling is a special case of MH

Gibbs sampling is equivalent to using the proposal
$q(x'|x) = p(x'_i|x_{-i})\mathbb{I}(x'_{-i} = x_{-i})$

The acceptance probability is 1

$$\begin{aligned}
\alpha &= \frac{p(x')q(x|x')}{p(x)q(x'|x)} \\
&= \frac{p(x'_i|x'_{-i})p(x'_{-i})p(x_i|x'_{-i})\mathbb{I}(x'_{-i} = x_{-i})}{p(x_i|x_{-i})p(x_{-i})p(x'_i|x_{-i})\mathbb{I}(x'_{-i} = x_{-i})} \\
&= \frac{p(x'_i|x_{-i})p(x_{-i})p(x_i|x_{-i})}{p(x_i|x_{-i})p(x_{-i})p(x'_i|x_{-i})} = 1
\end{aligned}$$

Summary

The MCMC strategy is to construct a Markov chain which has the target distribution $p(x)$ as its stationary distribution. We showed the Metropolis-Hastings algorithm accomplishes this, and likewise for Gibbs sampling which is a special case.

That's a result which says we'll be sampling from $p(x)$ *eventually*. How do we know when we've sampled enough to reach the stationary distribution? How do we implement MCMC in practice?

Burn in

As previously mentioned, we start MCMC with an initial random sample x^0, which is typically chosen to be some state that is thought to have high probability under $p(x)$ (e.g., near a mode). After some number of samples, the Markov chain should "forget" its starting value. The initial period, whose samples will be dropped, is called the *burn in* phase. Unfortunately, there's no way to know in advance exactly how many samples are required for the process to forget its starting point and settle into something near the stationary distribution.

The amount of time it takes for a Markov chain to converge to the stationary distribution, and forget its initial state, is called the *mixing time*. Chains with a long mixing time are called "slowly mixing."

Dependent samples

Another issue is that the samples produced by MCMC are not i.i.d. by construction. This is a problem if you want to construct parameter estimates from your generated samples, and calculate the variance of your estimators to report confidence intervals, etc. Your estimates of the *variance* will be off (too low), because you've taken averages of dependent samples.

Dependent samples

Say our goal is to estimate $\mu = \mathbb{E}[f(x)]$. We generate some samples by MCMC and use

$$\hat{\mu} = \frac{1}{S} \sum_{s=1}^{S} f(x^s)$$

can prove under some conditions that $\sqrt{S}(\hat{\mu} - \mu) \to N(0, \sigma^2)$ where $\sigma^2 = var(f(x^s)) + 2\sum_{k=1}^{\infty} cov(f(x^s), f(x^{s+k}))$

In R, if `z` is your vector of $f(x^s)$ values:

```
library(mcmc)
initseq(z)$var.con
```

estimates σ^2. See the `mcmc` documentation for details. Also there are some nice plots and discussion here: *link*

Strategies

Strategy 1:

- Run MCMC M times, each run for S steps and starting from a different intial state.
- Collect the last state of each run.
- Pro: better chance of covering the state space, not much dependence
- Con: have to "burn in" every chain, drop a lot of samples

Strategy 2:

- Run MCMC once, and after some burn in period keep samples separated by some fixed number of steps (e.g., keep every 100th sample from a long chain). This is called *thinning*.
- Pro: "burn in" only once.
- Con: have to run a very long chain, samples may still be somewhat dependent

Strategies

Strategy 3: hybrid of the two. Run several chains, keep some samples from both.

MCMC diagnostics

The following slides are taken from Suchi Saria, who put together some helpful MCMC diagnostic plots in R.

Acceptance Rates

It is important to monitor the *acceptance rate* (the fraction of candidate draws that are accepted) of your Metropolis-Hastings algorithm.

If your acceptance rate is too high, the chain is probably not mixing well (not moving around the parameter space quickly enough).

If your acceptance rate is too low, your algorithm is too inefficient (rejecting too many candidate draws).

What is too high and too low depends on your specific algorithm, but generally

Formally, there are theoretical arguments indicating that the optimal acceptance rate is 44% for one dimension, and has a limit of 23.4% as the dimension goes to infinity.

Convergence

From our theory of Markov chains, we expect our chains to eventually converge to the stationary distribution, which is also our target distribution.

However, there is no guarantee that our chain has converged after M draws.

How do we know whether our chain has actually converged?

We can never be sure, but there are several tests we can do, both visual and statistical, to see if the chain appears to be converged.

Convergence Diagnostics in R

All the diagnostics we will use are in the `coda` package in R.

```
> library(coda)
```

Before we use the diagnostics, we should turn our chains into `mcmc` objects.

```
> mh.draws <- mcmc(mh.draws)
```

We can tell the `mcmc()` function to burn-in or drop draws with the `start` and `end` arguments.

`mcmc()` also has a `thin` argument, which only tells it the thinning interval that was used (it does not actually thin for us).

See the **R package** CODA (Convergence Diagnostics and Output Analysis).

```
> gibbs <- function(n.sims, beta.start, alpha, gamma, delta,
+                   y, t, burnin = 0, thin = 1){
+   beta.draws = c();
+   lambda.draws = matrix(NA,nrow = n.sims, ncol = length(y));
+   beta.cur = beta.start;
+   lambda.update = function(alpha, beta, y, t){
+     rgamma(length(y), y + alpha, t + beta);
+   }
+   beta.update = function(alpha, gamma, delta, lambda, y){
+     rgamma(1, length(y) * alpha + gamma, delta + sum(lambda));
+   }
+   for(i in 1:n.sims){
+     lambda.cur = lambda.update(alpha,beta.cur,y,t);
+     beta.cur = beta.update(alpha,gamma,delta,lambda.cur,y)
+     if(i > burnin & (i-burnin)%%thin == 0){
+       lambda.draws[(i-burnin)/thin,] = lambda.cur;
+       beta.draws[(i-burnin)/thin] = beta.cur;
+     }
+   }
+   return(list(lambda.draws = mcmc(lambda.draws), beta.draws = mcmc(beta.draws)));
+ }
```

Mixing

One way to see if our chain has converged is to see how well our chain is **mixing**, or moving around the parameter space.

If our chain is taking a long time to move around the parameter space, then it will take longer to converge.

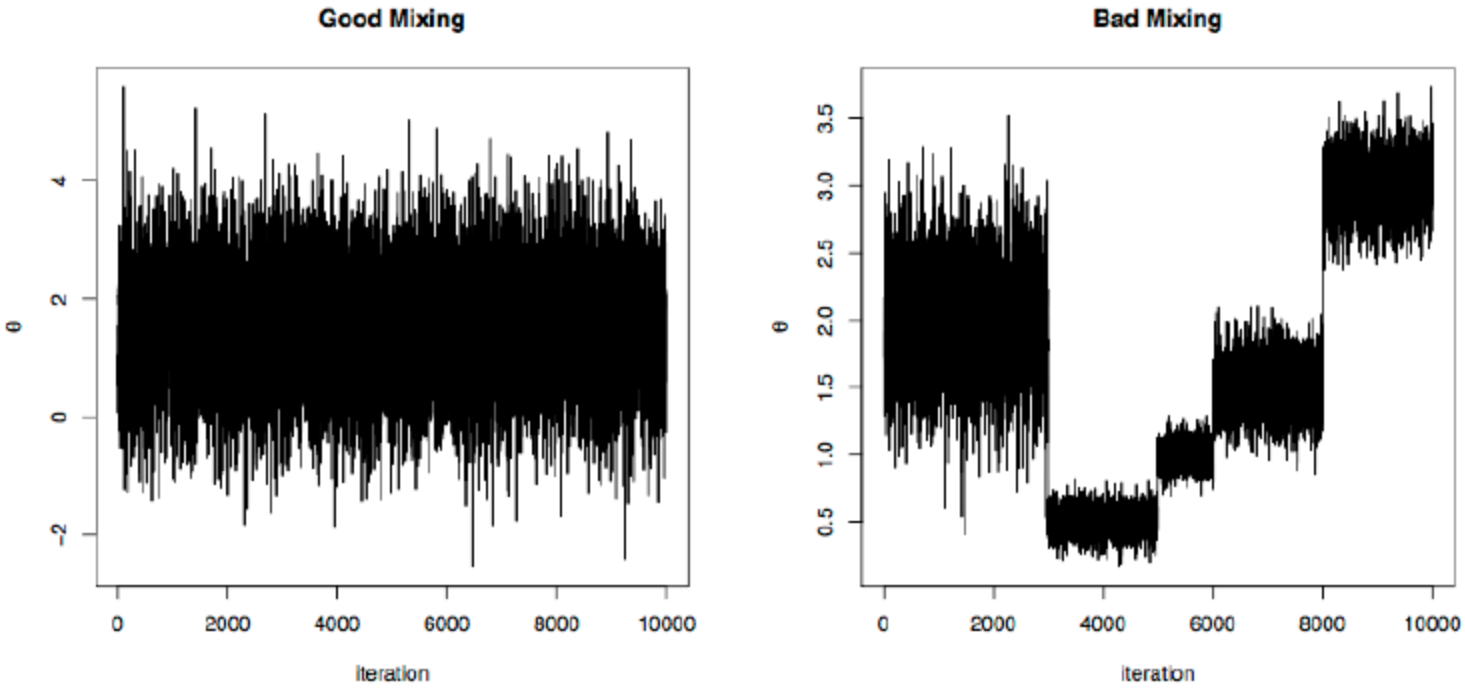
We can see how well our chain is mixing through visual inspection.

We need to do the inspections for every parameter.

Traceplots

A **traceplot** is a plot of the iteration number against the value of the draw of the parameter at each iteration.

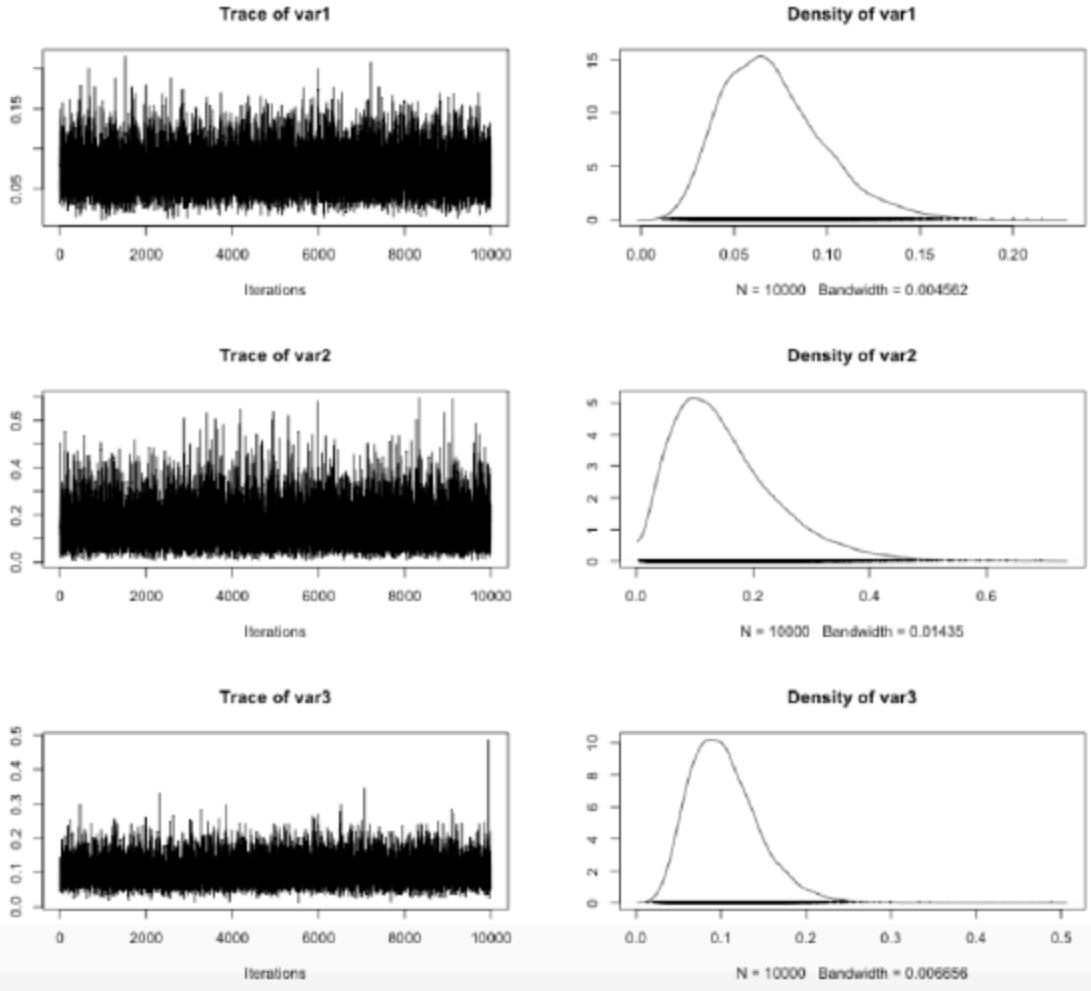
We can see whether our chain gets stuck in certain areas of the parameter space, which indicates bad mixing.



Traceplots and Density Plots

We can do traceplots and density plots by plotting an `mcmc` object or by calling the `traceplot()` and `densplot()` functions.

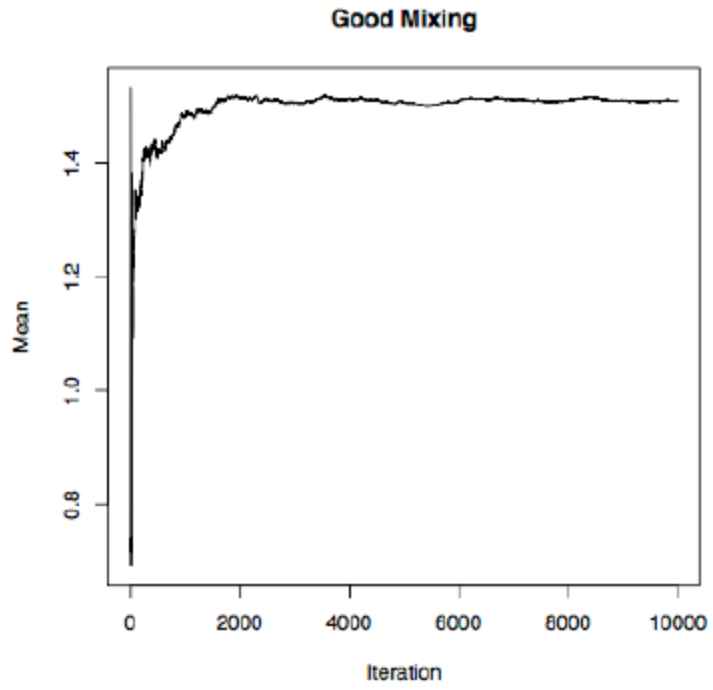
```
> plot(posterior$lambda.draws)
```

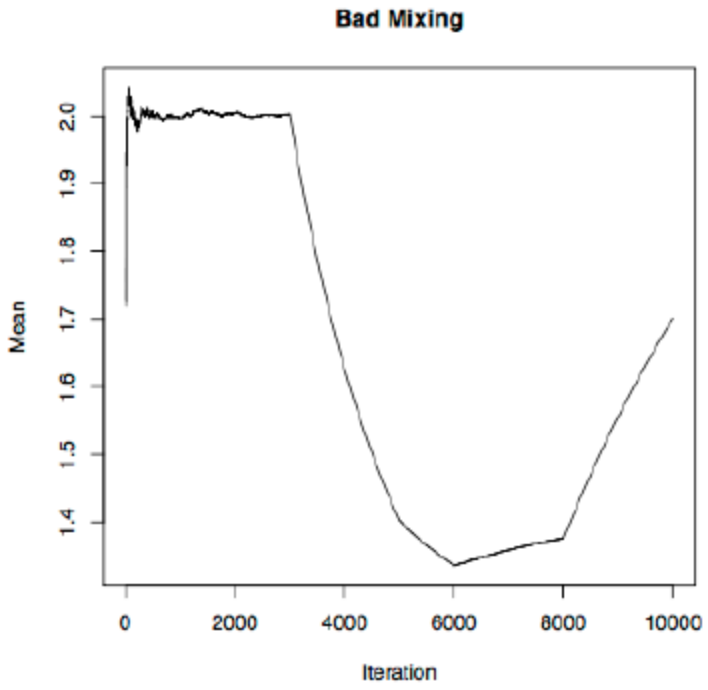


Running Mean Plots

We can also use **running mean plots** to check how well our chains are mixing.

A running mean plot is a plot of the iterations against the mean of the draws up to each iteration.





Autocorrelation

Another way to assess convergence is to assess the autocorrelations between the draws of our Markov chain.

The lag k autocorrelation ρ_k is the correlation between every draw and its kth lag:

$$\rho_k = \frac{\sum_{i=1}^{n-k}(x_i - \bar{x})(x_{i+k} - \bar{x})}{\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

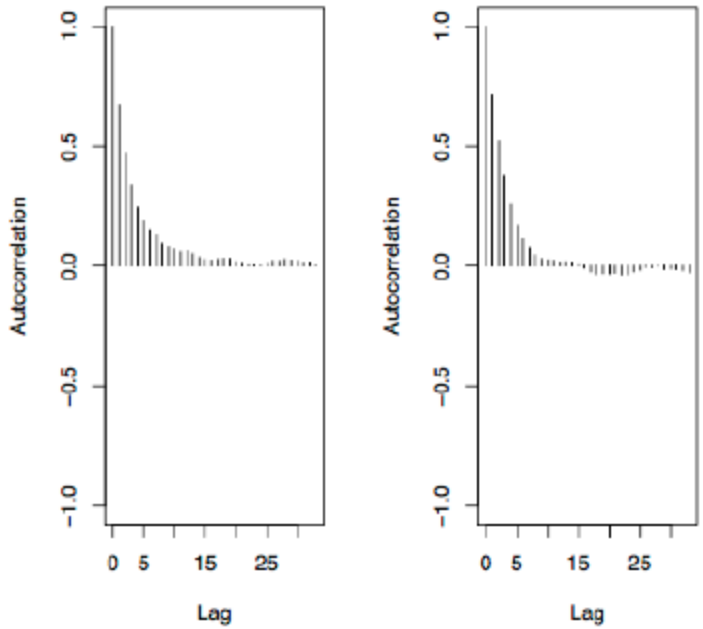
We would expect the kth lag autocorrelation to be smaller as k increases (our 2nd and 50th draws should be less correlated than our 2nd and 4th draws).

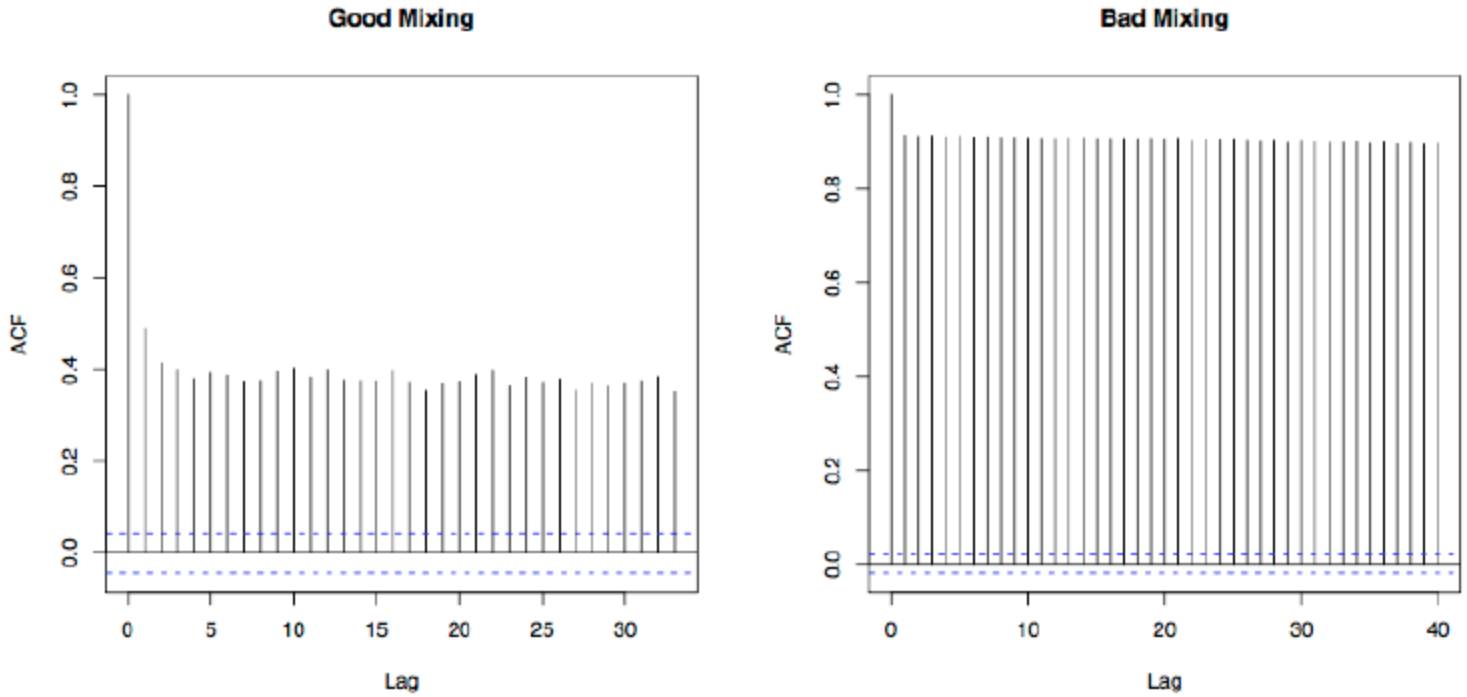
If autocorrelation is still relatively high for higher values of k, this indicates high degree of correlation between our draws and slow mixing.

Autocorrelation Plots

We can get autocorrelation plots using the `autocorr.plot()` function.

```
> autocorr.plot(mh.draws)
```




Good Mixing
ACF
1.0
0.8
0.6
0.4
0.2
0.0
0
5
10
15
20
25
30
Lag
Bad Mixing
ACF
1.0
0.8
0.6
0.4
0.2
0.0
0
10
20
30
40
Lag

Rejection Rate for Metropolis-Hastings Algorithm

We can also get a rejection rate for the Metropolis-Hastings algorithm using the `rejectionRate()` function.

```
> rejectionRate(mh.draws)

     var1      var2
0.5277055 0.4094819
```

To get the acceptance rate, we just want 1− rejection rate.

```
> acceptance.rate <- 1 - rejectionRate(mh.draws)
> acceptance.rate

     var1      var2
0.4722945 0.5905181
```