3.1 Markov Chain Monte Carlo (MCMC) Methods

Computation via conjugate analysis (Chapter 2) produces closed-form results (good) but is limited in scope to a fairly small set of models for which straightforward conjugate results are possible (bad); for example, there is no conjugate prior for $(\mu, \sigma^2, \nu)$ in the NB10 $t$ model.

This was a severe limitation for Bayesians for almost 250 years (from the 1750s to the 1980s).

Over the past 25 years or so the Bayesian community has “discovered” and developed an entirely new computing method, Markov chain Monte Carlo (MCMC) (“discovered” because the physicists first figured it out about 70 years ago: Metropolis and Ulam, 1949; Metropolis et al., 1953).

It became clear above that the central Bayesian practical challenge is the computation of high-dimensional integrals.

People working on the first atom bomb in World War II faced a similar challenge, and noticed that digital computers (which were then passing from theory (Turing 1943) to reality) offered an entirely new approach to solving the problem.
The idea (Metropolis and Ulam, 1949) was based on the observation that anything I want to know about a probability distribution can be learned to arbitrary accuracy by sampling from it.

Suppose, for example, that I’m interested in a posterior distribution $p(\theta|y)$ that cannot be worked with (easily) in closed form, and initially (to keep things simple) think of $\theta$ as a scalar (real number) rather than a vector.

Three things of direct interest to me about $p(\theta|y)$ would be

- its low-order moments, including the mean $\mu = E(\theta|y)$ and standard deviation $\sigma = \sqrt{V(\theta|y)}$,
- its shape (basically I’d like to be able to trace out (an estimate of) the entire density curve), and
- one or more of its quantiles (e.g., to construct a 95% central posterior interval for $\theta$ I need to know the 2.5% and 97.5% quantiles, and sometimes the posterior median (the 50th percentile) is of interest too).
Suppose I could take an **arbitrarily large random sample** from $p(\theta|y)$, say $
abla \theta_1, \ldots, \theta_m$.

Then each of the above three aspects of $p(\theta|y)$ can be **estimated** from the $\theta^*$ sample:

- $\hat{E}(\theta|y) = \bar{\theta}^* = \frac{1}{m} \sum_{j=1}^{m} \theta_j^*$, and $\sqrt{\hat{V}(\theta|y)} = \sqrt{\frac{1}{m-1} \sum_{j=1}^{m} (\theta_j^* - \bar{\theta}^*)^2}$;

- the density curve can be estimated by a **histogram** or **kernel density estimate**; and

- percentiles can be estimated by **counting** how many of the $\theta^*$ values fall below a series of specified points — e.g., to find an estimate of the 2.5% quantile I solve the equation

\[
\hat{F}_\theta(t) = \frac{1}{m} \sum_{j=1}^{m} I(\theta_j^* \leq t) = 0.025
\]

for $t$, where $I(A)$ is the **indicator function** (1 if $A$ is true, otherwise 0).

These are called **Monte Carlo** estimates of the true summaries of $p(\theta|y)$ (in
honor of the casinos) because they’re based on the controlled use of chance.

Theory shows that with large enough \( m \), each of the Monte Carlo (or simulation-based) estimates can be made arbitrarily close to the truth with arbitrarily high probability, under some reasonable assumptions about the nature of the random sampling.

One way to achieve this, of course, is to make the sampling IID (interestingly, this is sufficient but not necessary — see below).

If, for example, \( \bar{\theta}^* = \frac{1}{m} \sum_{j=1}^{m} \theta_j^* \) is based on an IID sample of size \( m \) from \( p(\theta|y) \), I can use the frequentist fact that in repeated sampling \( V(\bar{\theta}^*) = \frac{\sigma^2}{m} \), where (as above) \( \sigma^2 \) is the variance of \( p(\theta|y) \), to construct a Monte Carlo standard error (MCSE) for \( \bar{\theta}^* \):

\[
\widehat{SE}(\bar{\theta}^*) = \frac{\hat{\sigma}}{\sqrt{m}},
\]

where \( \hat{\sigma} \) is the sample SD of the \( \theta^* \) values.

This can be used, possibly after some preliminary experimentation, to decide on \( m \), the Monte Carlo sample size, which later will be called the
An IID Example

length of the monitoring run.

An IID example. Consider the posterior distribution
\[ p(\theta|y) = \text{Beta}(76.5, 353.5) \] in the AMI mortality example in Part 2.

Theory says that the posterior mean of \( \theta \) in this example is
\[ \frac{76.5}{76.5 + 353.5} \approx 0.1779 ; \] let’s see how well the Monte Carlo method does in estimating this known truth.

Here’s an R function to construct Monte Carlo estimates of the posterior mean and MCSE values for these estimates.

```r
beta.sim <- function( m, alpha, beta, n.sim, seed ) {
  set.seed( seed )
  theta.out <- matrix( 0, n.sim, 2 )
  for ( i in 1:n.sim ) {
    theta.sample <- rbeta( m, alpha, beta )
    theta.out[ i, 1 ] <- mean( theta.sample )
    theta.out[ i, 2 ] <- sqrt( var( theta.sample ) / m )
  }
}
```
This function simulates, \texttt{n.sim} times, the process of taking an \textbf{IID sample} of size \textit{m} from the Beta(\textit{\alpha}, \textit{\beta}) distribution and \textbf{calculating} $\bar{\theta}^*$ and $\widehat{SE}(\bar{\theta}^*)$.

```r
> m <- 100
> alpha <- 76.5
> beta <- 353.5
> n.sim <- 500
> seed <- c(6425451, 9626954)
> theta.out <- beta.sim(m, alpha, beta, n.sim, seed)
```

This took about 0.2 second at 1.6 Unix GHz.

```r
> theta.out[1:5,]

[,1]   [,2]
[1,] 0.1756400 0.001854220
[2,] 0.1764806 0.001703780
[3,] 0.1781742 0.001979863
[4,] 0.1793588 0.002038532
[5,] 0.1781556 0.001596011
```
The $\bar{\theta}^*$ values fluctuate around the truth with a **give-or-take** of about 0.0018, which agrees well with the **theoretical SE** $\frac{\sigma}{\sqrt{m}} = \frac{0.0184}{\sqrt{100}} \approx 0.00184$ (the SD value 0.0184 comes from page 47 in Part 2).

```r
> theta.bar <- theta.out[, 1]
> qqnorm((theta.bar - mean(theta.bar)) / sd(theta.bar),
       xlab = "Quantiles of Standard Normal", main = "", pch = 20 )
> abline( 0, 1 )
```

![Quantile-Quantile Plot](image.png)
IID Example (continued)

Each of the $\bar{\theta}^*$ values is the mean of $m = 100$ IID draws, so (by the CLT) the distribution of the random variable $\bar{\theta}^*$ should be closely approximated by a Gaussian, and you can see from the qqplot above that this is true.

```r
> truth <- alpha / ( alpha + beta )
> theta.bar.SE <- theta.out[, 2]
> sum((theta.bar - 1.96 * theta.bar.SE < truth) *
>     (truth < theta.bar + 1.96 * theta.bar.SE)) / n.sim
> [1] 0.94
```

With this set of pseudo-random numbers, 94% of the nominal 95% Monte Carlo confidence intervals for the posterior mean included the truth.

Evidently frequentist ideas can be used to work out how big $m$ needs to be to have any desired Monte Carlo accuracy for $\bar{\theta}^*$ as an estimate of the posterior mean $E(\theta|y)$.

In practice, with $p(\theta|y)$ unknown, I would probably take an initial sample (in this case, of size $m = 100$) and look at the MCSE to decide how big $m$ really needs to be.
Let’s say I ran the program with \( n.\text{sim} = 1 \) and \( m = 100 \) and got the following results:

```r
> theta.bar <- beta.sim( m, alpha, beta, 1, seed )
> theta.bar

[,1]    [,2]
[1,] 0.1756400 0.001854220
```

(1) Suppose I wanted the MCSE of \( \bar{\theta}^* \) to be (say) \( \epsilon = 0.00005 \); then I could solve the equation

\[
\frac{\hat{\sigma}}{\sqrt{m}} = \epsilon \quad \leftrightarrow \quad m = \frac{\sigma^2}{\epsilon^2},
\]

which says (unhappily) that the required \( m \) goes up as the square of the posterior SD and as the inverse square of \( \epsilon \).

The program results above show that \( \frac{\hat{\sigma}}{\sqrt{100}} \doteq 0.001854220 \), from which \( \hat{\sigma} \doteq 0.01854220 \), meaning that to get \( \epsilon = 0.00005 \) I need a sample of size \( \frac{0.01854220^2}{0.00005^2} \doteq 137,525 \doteq 138K. \)
(2) Suppose instead that I wanted \( \bar{\theta}^* \) to differ from the true posterior mean \( \mu \) by no more than \( \epsilon_1 \) with Monte Carlo probability at least \( (1 - \epsilon_2) \):

\[
P(|\bar{\theta}^* - \mu| \leq \epsilon_1) \geq 1 - \epsilon_2,
\]

where \( P(\cdot) \) here is based on the (frequentist) Monte Carlo randomness inherent in \( \bar{\theta}^* \).

I know from the CLT and the calculations above that in repeated sampling \( \bar{\theta}^* \) is approximately Gaussian with mean \( \mu \) and variance \( \frac{\sigma^2}{m} \); this leads to the inequality

\[
m \geq \frac{\sigma^2 \left[ \Phi^{-1}(1 - \frac{\epsilon_2}{2}) \right]^2}{\epsilon_1^2},
\]

where \( \Phi^{-1}(q) \) is the place on the standard normal curve where 100\( q\% \) of the area is to the left of that place (the \( q \)th quantile of the standard Gaussian distribution).

(5) is like (3) except that the value of \( m \) from (3) has to be multiplied by \( \left[ \Phi^{-1}(1 - \frac{\epsilon_2}{2}) \right]^2 \), which typically makes the required sample sizes even bigger.
A Closer Look at IID Sampling

For example, with $\epsilon_1 = 0.00005$ and $\epsilon_2 = 0.05$ — i.e., to have at least 95% Monte Carlo confidence that reporting the posterior mean as 0.1756 will be correct to about **four significant figures** — (5) says that I would need a monitoring run of at least $137,525(1.959964)^2 \div 528,296 \div 528 \text{K}$. This sounds like a long monitoring run but only takes about **2 seconds** at 1.6 Unix GHz, yielding $\left[ \hat{\theta}^*, \hat{SE}(\hat{\theta}^*) \right] = (0.1779052, 0.00002)$, which compares **favorably** with the **true value** 0.1779070.

It’s evident from calculations like these that people often **report simulation-based answers** with numbers of significant figures **far in excess of what’s justified** by the actual accuracy of the Monte Carlo estimates.

I was able to easily perform the above simulation study because **R** has a large variety of built-in functions like `rbeta` for **pseudo-random-number generation**.

How would I go about **writing** such functions **myself**?

There are a number of **general-purpose** methods for generating random numbers (I won’t attempt a survey here); the one we need to look closely at, to
understand the algorithms that arise later in this part of the short course, is rejection sampling (von Neumann 1951), which is often one of the most computationally efficient ways to make IID draws from a distribution.

Example. Continuing the AMI mortality case study from Part 2, consider an alternative prior specification in which I’d like to put most (90%, say) of the prior mass in the interval (0.05, 0.50); calculations like those in Part 2 within the conjugate Beta family yield prior hyperparameter values of $(\alpha_0, \beta_0) = (2.0, 6.4)$ (this Beta distribution has prior mean and SD 0.24 and 0.14, respectively).

Suppose that the sample size $n$ was smaller at 74, and $s = 16$ AMI deaths were observed, so that the data mean was 0.216; the posterior is then

$$\text{Beta}(\alpha_0 + s, \beta_0 + n - s) = \text{Beta}(18.0, 64.4).$$

I’ll pretend for the sake of illustration of rejection sampling that I don’t know the formulas for the mean and SD of a Beta distribution, and suppose that I wanted to use IID Monte Carlo sampling from the

$$\text{Beta}(\alpha_0 + s, \beta_0 + n - s)$$

posterior to estimate the posterior mean.
Here’s von Neumann’s basic idea, which (as it turns out) works equally well for scalar or vector $\theta$: suppose the target density $p(\theta|y)$ is difficult to sample from, but you can find an integrable envelope function $G(\theta|y)$ such that 

(a) $G$ dominates $p$ in the sense that $G(\theta|y) \geq p(\theta|y) \geq 0$ for all $\theta$ and 

(b) the density $g$ obtained by normalizing $G$ — later to be called the proposal distribution — is easy and fast to sample from.

Then to get a random draw from $p$, make a draw $\theta^*$ from $g$ instead and accept or reject it according to an acceptance probability $\alpha_R(\theta^*|y)$; if you reject the draw, repeat this process until you accept.

von Neumann showed that the choice

$$\alpha_R(\theta^*|y) = \frac{p(\theta^*|y)}{G(\theta^*|y)}$$

(correctly) produces IID draws from $p$, and you can intuitively see that he’s right by the following argument.

Making a draw from the posterior distribution of interest is like choosing a
point at random (in two dimensions) under the density curve \( p(\theta|y) \) in such a way that all possible points are equally likely, and then writing down its \( \theta \) value.

If you instead draw from \( G \) so that all points under \( G \) are equally likely, to get correct draws from \( p \) you’ll need to throw away any point that falls between \( p \) and \( G \), and this can be accomplished by accepting each sampled point \( \theta^* \) with probability \( \frac{p(\theta^*|y)}{G(\theta^*|y)} \), as von Neumann said.

A summary of this method is on the next page.

The figure two pages below demonstrates this method on the Beta(18.0, 64.4) density arising in the Beta-Bernoulli example above.

Rejection sampling permits considerable flexibility in the choice of envelope function; here, borrowing an idea from Gilks and Wild (1992), I’ve noted that the relevant Beta density is log concave (a real-valued function is log concave if its second derivative on the log scale is everywhere non-positive), meaning that it’s easy to construct an envelope on that scale in a piecewise linear fashion, by choosing points on the log density and constructing
**Algorithm (rejection sampling).** To make $m$ draws at random from the density $p(\theta|y)$ for scalar or vector $\theta$, select an integrable **envelope function** $G$ — which when normalized to integrate to 1 is the **proposal distribution** $g$ — such that $G(\theta|y) \geq p(\theta|y) \geq 0$ for all $\theta$; define the acceptance probability

$$\alpha_R(\theta^*|y) = \frac{p(\theta^*|y)}{G(\theta^*|y)};$$

and

Initialize $t \leftarrow 0$

Repeat {
    Sample $\theta^* \sim g(\theta|y)$
    Sample $u \sim \text{Uniform}(0, 1)$
    If $u \leq \alpha_R(\theta^*|y)$ then
    $$\{ \theta_{t+1} \leftarrow \theta^*; \ t \leftarrow (t + 1) \}$$
}

until $t = m$.

**tangents** to the curve at those points.

The **simplest** possible such envelope involves **two line segments**, one on either side of the **mode**.
The optimal choice of the tangent points would maximize the marginal probability of acceptance of a draw in the rejection algorithm, which can be shown to be
Rejection Sampling (continued)

\[
\left[ \int G(\theta) \, d\theta \right]^{-1}; \tag{7}
\]

in other words, you should **minimize** the area under the (un-normalized) envelope function subject to the constraint that it **dominates** the target density \( p(\theta|y) \) (which makes eminently good sense).

Here this optimum turns out to be attained by locating the two tangent points at about **0.17** and **0.26**, as in the figure above; the resulting acceptance probability of about **0.75** could clearly be **improved** by adding more tangents.

**Piecewise linear** envelope functions on the log scale are a **good choice** because the resulting envelope density on the raw scale is a piecewise set of **scaled exponential distributions** (see the bottom panel in the figure above), from which random samples can be taken easily and **quickly**.

A **preliminary** sample of \( m_0 = 500 \) IID draws from the Beta(18.0, 64.4) distribution using the above rejection sampling method yields \( \bar{\theta}^* = 0.2197 \) and \( \hat{\sigma} = 0.04505 \), meaning that the posterior mean has already been estimated with an **MCSE** of only \( \frac{\hat{\sigma}}{\sqrt{m_0}} = 0.002 \) even with just **500** draws.
Suppose, however, that — as in equation (4) above — I want \( \bar{\theta}^* \) to differ from the true posterior mean \( \mu \) by no more than some (perhaps even smaller) tolerance \( \epsilon_1 \) with Monte Carlo probability at least \( (1 - \epsilon_2) \); then equation (5) tells me how long to monitor the simulation output.

For instance, to pin down three significant figures (sigfigs) in the posterior mean in this example with high Monte Carlo accuracy I might take \( \epsilon_1 = 0.0005 \) and \( \epsilon_2 = 0.05 \), which yields a recommended IID sample size of

\[
\frac{(0.04505^2)(1.96)^2}{0.0005^2} \approx 31,200.
\]

So I take another sample of 30,700 (which is virtually instantaneous at 1.6 Unix GHz) and merge it with the 500 draws I already have; this yields \( \bar{\theta}^* = 0.21827 \) and \( \hat{\sigma} = 0.04528 \), meaning that the MCSE of this estimate of \( \mu \) is

\[
\frac{0.04528}{\sqrt{31200}} \approx 0.00026.
\]

I might announce that I think \( E(\theta|y) \) is about 0.2183, give or take about 0.0003, which accords well with the true value 0.2184.

Of course, other aspects of \( p(\theta|y) \) are equally easy to monitor; for example, if I want a Monte Carlo estimate of \( p(\theta \leq q|y) \) for some \( q \), as noted above I just work out the proportion of the sampled \( \theta^* \) values that are no larger than \( q \).
Beyond Rejection Sampling

Or, even better, I recall that \( P(A) = E[I(A)] \) for any event or proposition \( A \), so to the Monte Carlo dataset (see page 35 below) consisting of 31,200 rows and one column (the \( \theta^*_t \)) I add a column monitoring the values of the derived variable which is 1 whenever \( \theta^*_t \leq q \) and 0 otherwise; the mean of this derived variable is the Monte Carlo estimate of \( p(\theta \leq q|y) \), and I can attach an MCSE to it in the same way I did with \( \bar{\theta}^* \).

By this approach, for instance, the Monte Carlo estimate of \( p(\theta \leq 0.15|y) \) based on the 31,200 draws examined above comes out \( \hat{p} = 0.0556 \) with an MCSE of \( 0.0013 \).

Percentiles are typically harder to pin down with equal Monte Carlo accuracy (in terms of sigfigs) than means or SDs, because the 0/1 scale on which they’re based is less information-rich than the \( \theta^* \) scale itself; if I wanted an MCSE for \( \hat{p} \) of 0.0001 I would need an IID sample of more than 5 million draws (which would still only take a few seconds at contemporary workstation speeds).

**IID sampling is not necessary.** Nothing in the Metropolis-Ulam idea of
Monte Carlo estimates of posterior summaries requires that these estimates be based on **IID samples from the posterior**.

This is lucky, because in practice it’s often difficult, particularly when \( \theta \) is a **vector of high dimension** (say \( k \)), to figure out how to make such an IID sample, via rejection sampling or other methods (e.g., imagine trying to find an **envelope function** for \( p(\theta|y) \) when \( k \) is 10 or 100 or 1,000).

Thus it’s necessary to **relax** the assumption that \( \theta_j^* \overset{\text{IID}}{\sim} p(\theta|y) \), and to consider samples \( \theta_1^*, \ldots, \theta_m^* \) that form a **time series**: a series of draws from \( p(\theta|y) \) in which \( \theta_j^* \) may **depend on** \( \theta_{j'}^* \), for \( j' < j \).

In their pioneering paper Metropolis et al. (1953) allowed for **serial dependence** of the \( \theta_j^* \) by combining von Neumann’s idea of rejection sampling (which had itself only been published a few years earlier in 1951) with concepts from **Markov chains**, a subject in the theory of **stochastic processes**.

Combining **Monte Carlo sampling** with **Markov chains** gives rise to the name now used for this technique for solving the Bayesian high-dimensional integration problem: **Markov chain Monte Carlo** (MCMC).
Markov chains. A stochastic process is just a collection of random variables \( \{\theta^*_t, t \in T \} \) for some index set \( T \), usually meant to stand for time.

In practice \( T \) can be either discrete, e.g., \( \{0, 1, \ldots \} \), or continuous, e.g., \([0, \infty)\).

Markov chains are a special kind of stochastic process that can either unfold in discrete or continuous time — I’ll talk here about discrete-time Markov chains, which is all you need for MCMC.

The possible values that a stochastic process can take on are collectively called the state space \( S \) of the process — in the simplest case \( S \) is real-valued and can also either be discrete or continuous.

Intuitively speaking, a Markov chain (e.g., Feller, 1968; Roberts, 1996; Gamerman, 1997) is a stochastic process evolving in time in such a way that the past and future states of the process are independent given the present state—in other words, to figure out where the chain is likely to go next you don’t need to pay attention to where it’s been, you just need to consider where it is now.
More formally, a stochastic process \( \{\theta^*_t, t \in T\} \), \( T = \{0, 1, \ldots \} \), with state space \( S \) is a **Markov chain** if, for any set \( A \in S \),

\[
P(\theta^*_{t+1} \in A | \theta^*_0, \ldots, \theta^*_t) = P(\theta^*_{t+1} \in A | \theta^*_t).
\] (8)

The theory of Markov chains is **harder mathematically** if \( S \) is continuous (e.g., Tierney, 1996), which is what we need for MCMC with real-valued parameters, but **most of the main ideas emerge with discrete state spaces**, and I’ll assume discrete \( S \) in the intuitive discussion here.

**Example.** For a simple example of a **discrete-time Markov chain** with a **discrete state space**, imagine a **particle** that moves around on the integers \( \{\ldots, -2, -1, 0, 1, 2, \ldots \} \), starting at 0 (say).

Wherever it finds itself at time \( t \)—say at \( i \)—it **tosses a (3-sided) coin** and moves to \((i - 1)\) with probability \( p_1 \), stays at \( i \) with probability \( p_2 \), and moves to \((i + 1)\) with probability \( p_3 \), for some \( 0 < p_1, p_2, p_3 < 1 \) with \( p_1 + p_2 + p_3 = 1 \)—these are the **transition probabilities** for the process.

This is a **random walk** (on the integers), and it’s **clearly a Markov chain**.
Markov Chains (continued)

**Nice behavior.** The most nicely-behaved Markov chains satisfy three properties:

- They’re **irreducible**, which basically means that no matter where it starts the chain has to be able to reach any other state in a finite number of iterations with positive probability;

- They’re **aperiodic**, meaning that for all states $i$ the set of possible sojourn times, to get back to $i$ having just left it, can have no divisor bigger than 1 (this is a technical condition; periodic chains still have some nice properties, but the nicest chains are aperiodic).

- They’re **positive recurrent**, meaning that (a) for all states $i$, if the process starts at $i$ it will return to $i$ with probability 1, and (b) the expected length of waiting time til the first return to $i$ is finite.

Notice that this is a bit delicate: wherever the chain is now, we insist that it must certainly come back here, but we don’t expect to have to wait forever for this to happen.
The random walk defined above is clearly \textbf{irreducible} and \textbf{aperiodic}, but it may not be \textbf{positive recurrent} (depending on the $p_i$): it’s true that it has positive probability of returning to wherever it started, but (because $S$ is \textbf{unbounded}) this probability may not be 1, and on average you may have to wait forever for it to return.

We can fix this by \textbf{bounding} $S$: suppose instead that
\[ S = \{-k, -(k - 1), \ldots, -1, 0, 1, \ldots, k\}, \]
keeping the same transition probabilities except \textbf{rejecting} any moves \textbf{outside the boundaries} of $S$.

This bounded random walk now satisfies \textbf{all three of the nice properties}.

\textbf{The value of nice behavior.} Imagine running the bounded random walk for a long time, and look at the \textbf{distribution} of the \textbf{states} it visits—over time this distribution should \textbf{settle down} (converge) to a kind of limiting, \textbf{steady-state} behavior.

This can be demonstrated by \textbf{simulation}, for instance in R, and using the \textbf{bounded random walk} as an example:
Markov Chains (continued)

```r
rw.sim <- function( k, p, theta.start, n.sim, seed ) {
  set.seed( seed )
  theta <- rep( 0, n.sim + 1 )
  theta[ 1 ] <- theta.start
  for ( i in 1:n.sim ) {
    theta[ i + 1 ] <- move( k, p, theta[ i ] )
  }
  return( table( theta ) )
}
move <- function( k, p, theta ) {
  repeat {
    increment <- sample( x = c( -1, 0, 1 ), size = 1, prob = p )
    theta.next <- theta + increment
    if ( abs( theta.next ) <= k ) {
      return( theta.next )
      break
    }
  }
}
```

Bayesian Statistics 3a: Simulation-Based Computation 26
Markov Chains (continued)

greco 171> R
R version 2.5.1 (2007-06-27)
Copyright (C) 2007 The R Foundation for Statistical Computing
> p <- c( 1, 1, 1 ) / 3
> k <- 5
> theta.start <- 0
> seed <- c( 6425451, 9626954 )
> rw.sim( k, p, theta.start, 10, seed )
theta
0 1 2
5 5 1
> rw.sim( k, p, theta.start, 100, seed )
-2 -1 0 1 2 3 4 5
7 9 16 17 23 14 8 7
> rw.sim( k, p, theta.start, 1000, seed )
-5 -4 -3 -2 -1 0 1 2 3 4 5
65 115 123 157 148 123 106 82 46 21 15
> rw.sim( k, p, theta.start, 10000, seed )
-5 -4 -3 -2 -1 0 1 2 3 4 5
581 877 941 976 959 1034 1009 982 1002 959 681
You can see that the distribution of where the chain has visited is **converging** to something close to **uniform** on \{-5, -4, \ldots , 4, 5\}, except for the effects of the **boundaries**.

Letting \( q_1 \) denote the **limiting** probability of being in one of the 9 **non-boundary** states \((-4, -3, \ldots , 3, 4)\) and \( q_2 \) be the **long-run** probability of being in one of the 2 **boundary** states \((-5, 5)\), on grounds of **symmetry** you can guess that \( q_1 \) and \( q_2 \) should satisfy

\[
9q_1 + 2q_2 = 1 \quad \text{and} \quad q_1 = \frac{3}{2} q_2, \tag{9}
\]

from which \((q_1, q_2) = \left( \frac{3}{31}, \frac{2}{31} \right) \doteq (0.096774, 0.064516)\).

Based on the run of **1,000,001 iterations** above you would estimate these
probabilities \textit{empirically} as
\[
\left[ \frac{98535+...+95703}{(9)(1000001)}, \frac{65273+63767}{(2)(1000001)} \right] \doteq (0.096773, 0.064520).
\]

It should also be clear that the limiting distribution \textbf{does not depend} on the initial value of the chain:

> \texttt{rw.sim(k, p, 5, 100000, seed )}

-5 -4 -3 -2 -1 0 1 2 3 4 5
6515 9879 9876 9624 9374 9705 9959 9738 9678 9365 6288

Of course, you get a \textbf{different limiting distribution} with a \textbf{different choice} of \((p_1, p_2, p_3)\):

> \texttt{p <- c( 0.2, 0.3, 0.5 )}
> \texttt{rw.sim( k, p, 0, 10, seed )}

0 1 2 3
1 3 4 3
> \texttt{rw.sim( k, p, 0, 100, seed )}

0 1 2 3 4 5
1 3 6 13 30 48
Stationary distributions. A positive recurrent and aperiodic chain is called ergodic, and it turns out that such chains possess a unique stationary (or equilibrium, or invariant) distribution \( \pi \), characterized by the relation

\[
\pi(j) = \sum_i \pi(i) P_{ij}(t)
\]  

for all states \( j \) and times \( t \geq 0 \), where \( P_{ij}(t) = P(\theta^*_t = j | \theta^*_{t-1} = i) \) is the transition matrix of the chain.
The MCMC Payoff

Informally, the stationary distribution characterizes the behavior that the chain will settle into after it’s been run for a long time, regardless of its initial state.

The point of all of this. Given a parameter vector $\theta$ and a data vector $y$, the Metropolis et al. (1953) idea is to simulate random draws from the posterior distribution $p(\theta|y)$, by constructing a Markov chain with the following four properties:

- It should have the same state space as $\theta$,
- It should be easy to simulate from,
- It should work equally well with an un-normalized $p(\theta|y)$, so that it’s not necessary to evaluate the normalizing constant, and
- Its equilibrium distribution should be $p(\theta|y)$.

If you can do this, you can run the Markov chain for a long time, generating a huge sample from the posterior, and then use simple descriptive summaries (means, SDs, correlations, histograms or kernel density estimates) to extract any features of the posterior you want.
The Ergodic Theorem

The mathematical fact that underpins this strategy is the ergodic theorem: if the Markov chain \( \{\theta_t^*\} \) is ergodic and \( f \) is any real-valued function for which \( E_\pi[f(\theta)] \) is finite, then with probability 1 as \( m \to \infty \)

\[
\frac{1}{m} \sum_{t=1}^{m} f(\theta_t^*) \to E_\pi[f(\theta)] = \sum_i f(i) \pi(i),
\]

(11)

in which the right side is just the expectation of \( f(\theta) \) under the stationary distribution \( \pi \).

In plain English this means that — as long as the stationary distribution is \( p(\theta|y) \) — you can learn (to arbitrary accuracy) about things like posterior means, SDs, and so on just by waiting for stationarity to kick in and monitoring thereafter for a long enough period.

Of course, as Roberts (1996) notes, the theorem is silent on the two key practical questions it raises: how long you have to wait for stationarity, and how long to monitor after that.

A third practical issue is what to use for the initial value \( \theta_0^* \): intuitively the
closer $\theta_0^*$ is to the center of $p(\theta|y)$ the less time you should have to wait for stationarity.

The standard way to deal with waiting for stationarity is to (a) run the chain from a good starting value $\theta_0^*$ for $b$ iterations, until equilibrium has been reached, and (b) discard this initial burn-in period.

All of this motivates the topic of MCMC diagnostics, which are intended to answer the following questions:

- What should I use for the initial value $\theta_0^*$?
- How do I know when I’ve reached equilibrium? (This is equivalent to asking how big $b$ should be.)
- Once I’ve reached equilibrium, how big should $m$ be, i.e., how long should I monitor the chain to get posterior summaries with decent accuracy?

The basis of the Monte Carlo approach to obtaining numerical approximations to posterior summaries like means and SDs is the (weak) Law of Large Numbers: with IID sampling
The Monte Carlo and MCMC Datasets (continued)

the **Monte Carlo estimates** of the true summaries of $p(\theta|y)$ are **consistent**, meaning that they can be made arbitrarily close to the truth with arbitrarily high probability as the number of monitoring iterations $m \to \infty$.

Before we look at how Metropolis et al. attempted to achieve the same goal with a **non-IID Monte Carlo approach**, let’s look at the **practical consequences** of switching from IID to Markovian sampling.

Running the **IID rejection sampler** on the AMI mortality example above for a total of $m$ monitoring iterations would produce something that might be called the **Monte Carlo (MC) dataset**, with one **row** for each **iteration** and one **column** for each **monitored quantity**; in that example it might look like the table on the next page (MCSEs in parenthesis).

Running the **Metropolis sampler** on the same example would produce something that might be called the **MCMC dataset**.

It would have a **similar structure** as far as the **columns** are concerned, but the rows would be divided into **three phases**:

- Iteration 0 would be the value(s) used to **initialize** the Markov chain;
The MC and MCMC Data Sets

The MC Data Set:

<table>
<thead>
<tr>
<th>Iteration</th>
<th>$\theta$</th>
<th>$I(\theta \leq 0.15)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\theta_1^* = 0.244$</td>
<td>$I_1^* = 0$</td>
</tr>
<tr>
<td>2</td>
<td>$\theta_2^* = 0.137$</td>
<td>$I_2^* = 1$</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td>$\vdots$</td>
</tr>
<tr>
<td>$m = 31,200$</td>
<td>$\theta_m^* = 0.320$</td>
<td>$I_m^* = 0$</td>
</tr>
</tbody>
</table>

| Mean      | 0.2183 (0.003) | 0.0556 (0.0013) |
| SD        | 0.04528        | —                 |

- Density (like the bottom Trace plot on page 17) —

- Iterations 1 through $b$ would be the **burn-in** period, during which the chain reaches its **equilibrium** or **stationary** distribution (as mentioned above, iterations 0 through $b$ are generally **discarded**); and

- Iterations $(b + 1)$ through $(b + m)$ would be the **monitoring** run, on which **summaries** of the posterior (means, SDs, density traces, ...) will be based.