Stat 5102 Notes: Markov Chain Monte Carlo and Bayesian Inference

Charles J. Geyer

April 12, 2015

1 Introduction

This handout does Bayesian inference via Markov chain Monte Carlo (MCMC). It gives a brief introduction to ordinary Monte Carlo (OMC) and MCMC. For more about MCMC, see Geyer (2011).

2 The Problem

This is an example of an application of Bayes rule that requires some form of computer analysis.

The problem is the same one that was done by maximum likelihood on the computer examples web pages (`http://www.stat.umn.edu/geyer/5102/examp/like.html`). The data model is gamma. We will use the Jeffreys prior.

2.1 Data

The data are loaded by the R command

```
> foo <- read.table(url("http://www.stat.umn.edu/geyer/5102/data/ex3-1.txt"),
+     header = TRUE)
> x <- foo$x
```

2.2 R Package

We load the R package `mcmc`, which is available from CRAN (`http://cran.r-project.org`).

```
> library(mcmc)
```

If this does not work, then get the `mcmc` package using the package menu on the R app or using the R function `install.packages`.

2.3 Random Number Generator Seed

In order to get the same results every time, we set the seed of the random number generator.

```
> set.seed(42)
```

To get different results, change the seed or simply omit this statement.

2.4 Prior

The Fisher information matrix for the two-parameter gamma distribution and sample size one is, from the course slides, deck 3, slide 96 (`http://www.stat.umn.edu/geyer/5102/slides/s3.pdf#page=96`),

$$I(\boldsymbol{\theta}) = \begin{pmatrix} \text{trigamma}(\alpha) & -1/\lambda \\ -1/\lambda & \alpha/\lambda^2 \end{pmatrix}$$

Its determinant is

$$|I(\boldsymbol{\theta})| = \begin{vmatrix} \text{trigamma}(\alpha) & -1/\lambda \\ -1/\lambda & \alpha/\lambda^2 \end{vmatrix} = \frac{\alpha \, \text{trigamma}(\alpha) - 1}{\lambda^2}$$

and the Jeffreys prior is

$$g(\alpha, \lambda) = \frac{\sqrt{\alpha \, \text{trigamma}(\alpha) - 1}}{\lambda}$$

3 Theory of Monte Carlo

This section rehashes material on the course slides, deck 4, slide 111 to the end of the deck (`http://www.stat.umn.edu/geyer/5102/slides/s4.pdf#page=111`).

3.1 Ordinary Monte Carlo

The "Monte Carlo method" refers to the theory and practice of learning about probability distributions by simulation rather than calculus. In ordinary Monte Carlo (OMC) we use independent and identically distributed (IID) simulations from the distribution of interest. Suppose X_1, X_2, ... are

IID simulations from some distribution, and suppose we want to know an expectation

$$\mu = E\{g(X_i)\}.$$

Then the law of large numbers (LLN) then says

$$\hat{\mu}_n = \frac{1}{n} \sum_{i=1}^{n} g(X_i)$$

converges in probability to μ, and the central limit theorem (CLT) says

$$\sqrt{n}(\hat{\mu}_n - \mu) \xrightarrow{\mathcal{D}} \mathcal{N}(0, \sigma^2)$$

where

$$\sigma^2 = \operatorname{var}\{g(X_i)\}$$

which can be estimated by the empirical variance

$$\hat{\sigma}^2 = \frac{1}{n} \sum_{i=1}^{n} \bigl(g(X_i) - \hat{\mu}_n\bigr)^2$$

and this allows us to say everything we want to say about μ, for example, an asymptotic 95% confidence interval for μ is

$$\hat{\mu}_n \pm 1.96 \frac{\hat{\sigma}_n}{\sqrt{n}}$$

The theory of OMC is just the theory of frequentist statistical inference. The only differences are that

- the "data" X_1, ..., X_n are computer simulations rather than measurements on objects in the real world,
- the "sample size" is the number of computer simulations rather than the size of some real world data, and
- the unknown parameter μ is in principle completely known, given by some integral, which, unfortunately, we are unable to do.

Often we say that n is the *Monte Carlo sample size* to distinguish it from anything else that may be called a sample size. Often we say that $\hat{\mu}_n$ is the *Monte Carlo estimate* or *Monte Carlo approximation* or *Monte Carlo calculation* of μ to distinguish it from anything else that may be called an estimate. Often we say that $\hat{\sigma}_n/\sqrt{n}$ is the *Monte Carlo standard error* of $\hat{\mu}_n$ to distinguish it from anything else that may be called a standard error.

Great! The only problem is that is is very difficult to simulate IID simulations of random variables or random vectors whose distribution is not a brand name distribution.

3.2 Markov Chains

A *Markov chain* is a sequence of dependent random variables X_1, X_2, ... having the property that the conditional distribution of the future given the past depends only on the present: the conditional distribution of X_{n+1} given X_1, ..., X_n depends only on X_n.

We say the Markov chain has *stationary transition probabilities* if the conditional distribution of X_{n+1} given X_n is the same for all n. This property is so important that it is often assumed without comment. Every Markov chain used in MCMC has stationary transition probabilities, but this goes without saying.

We say the Markov chain has an *invariant distribution*, or *stationary distribution*, or *equilibrium distribution* if whenever X_n has this equilibrium distribution and the conditional distribution of X_{n+1} given X_n is that of the transition probabilities of the Markov chain, then X_{n+1} also has this equilibrium distribution (same as X_n). Thus if X_1 has the equilibrium distribution, then so does X_n for all n.

3.3 The Metropolis Algorithm

It turns out that there is an algorithm for simulating a Markov chain having any equilibrium distribution for which one has an unnormalized PDF, called the *Metropolis-Hastings-Green algorithm*. An R contributed package `mcmc` has a function `metrop` that does this using the most basic version, called the *normal random walk Metropolis algorithm*. It works as follows, suppose the current state of the Markov chain is X_n and suppose the unnormalized density of the desired equilibrium distribution is h. Then the next step of the Markov chain is simulated as follows.

- Simulate Y_n having a $\mathcal{N}(0, \tau^2)$ distribution.
- Calculate
$$r = \frac{h(X_n + Y_n)}{h(X_n)}$$
- Simulate U_n having a $\text{Unif}(0, 1)$ distribution.
- If $U_n < r$, then set $X_{n+1} = X_n + Y_n$, otherwise set $X_{n+1} = X_n$.

The algorithm works just as well when the state of the Markov chain is a vector. We just replace normal font with bold face, and the variance τ^2 with a variance matrix $\mathbf{M}$. Suppose the current state of the Markov chain is $\mathbf{X}_n$

and suppose the unnormalized density of the desired equilibrium distribution is h. Then the next step of the Markov chain is simulated as follows.

- Simulate $\mathbf{Y}_n$ having a $\mathcal{N}(0, \mathbf{M})$ distribution.
- Calculate
$$r = \frac{h(\mathbf{X}_n + \mathbf{Y}_n)}{h(\mathbf{X}_n)}$$
- Simulate U_n having a $\text{Unif}(0, 1)$ distribution.
- If $U_n < r$, then set $\mathbf{X}_{n+1} = \mathbf{X}_n + \mathbf{Y}_n$, otherwise set $\mathbf{X}_{n+1} = \mathbf{X}_n$.

The only thing we can adjust, and must adjust, in the algorithm is the *proposal variance*, τ^2 in the scalar version and $\mathbf{M}$ in the vector version. If we make the variance too small, then the chain only takes little baby steps and takes a very long time to equilibrate. If we make the variance too big, then the algorithm proposes very large steps Y_n or $\mathbf{Y}_n$, but most of those steps are not used in the last step because r calculated in the third step is too small. So we want something in the middle. It is generally accepted that the acceptance rate, which is the fraction of iterations in which $U_n < r$ in step 4 of the Metropolis algorithm (so $\mathbf{X}_{n+1} \neq \mathbf{X}_n$), should be about 20% (Geyer, 2011, Section 1.13).

3.4 Markov Chain Monte Carlo

MCMC is much like OMC. Most Markov chains used in MCMC obey the LLN and the CLT. These are the Markov chain LLN and Markov chain CLT and are not quite the same as the IID LLN and CLT. However, they serve the purpose.

Suppose X_1, X_2, ... is a Markov chain whose initial distribution is its equilibrium distribution, and suppose we want to know an expectation

$$\mu = E\{g(X_i)\}.$$

Then the law of large numbers (LLN) then says

$$\hat{\mu}_n = \frac{1}{n} \sum_{i=1}^{n} g(X_i)$$

converges in probability to μ, and the central limit theorem (CLT) says

$$\sqrt{n}(\hat{\mu}_n - \mu) \xrightarrow{\mathcal{D}} \mathcal{N}(0, \sigma^2)$$

where

$$\sigma^2 = \operatorname{var}\{g(X_i)\} + 2 \sum_{k=1}^{\infty} \operatorname{cov}\{g(X_i), g(X_{i+k})\}$$

which can be estimated by the method of batch means (more on this later).

These results are stated in terms of a Markov chain started at equilibrium, which usually cannot be done, but they also hold regardless of the initial distribution, meaning $\hat{\mu}_n$ is calculated for a Markov chain started in any distribution, but $E\{g(X_i)\}$, $\operatorname{var}\{g(X_i)\}$, and $\operatorname{cov}\{g(X_i), g(X_{i+k})\}$ refer to a Markov chain with the same transition probabilities and stationary initial distribution.

Thus MCMC works just like OMC, except the variance in the CLT is more complicated and must be estimated differently.

3.5 Batch Means

The Markov chain CLT says that $\hat{\mu}_n$ is approximately normal and centered at the quantity μ that we are trying to calculate approximately. All we need to know is estimate the variance, which the CLT says obeys the square root law. It is of the form σ^2/n for some constant σ^2, which we don't know.

However, the estimate $\hat{\mu}_b$ for a subsection of the chain of length b, should also be approximately normal with mean μ and variance σ^2/b, so long as b is large enough. Suppose b evenly divides n and we form the means

$$\hat{\mu}_{b,k} = \frac{1}{b} \sum_{i=bk+1}^{bk+b} g(X_i)$$

for $k = 1, \ldots, m = n/b$. Each of these "batch means" is approximately normal with mean approximately μ and variance approximately σ^2/b. Thus their empirical variance

$$\frac{1}{m} \sum_{k=1}^{m} (\hat{\mu}_{b,k} - \hat{\mu}_n)^2$$

estimates σ^2/b. And b/n times this estimates σ^2/n, the asymptotic variance of $\hat{\mu}_n$.

Alternatively, we can just make a t confidence interval using the m batch means as data.

4 Trying it Out

4.1 Log Unnormalized Posterior

We need to define an R function that calculates the log unnormalized posterior.

```
> lupost <- function(theta) {
+     stopifnot(is.numeric(theta))
+     stopifnot(is.finite(theta))
+     stopifnot(length(theta) == 2)
+     alpha <- theta[1]
+     lambda <- theta[2]
+     if (alpha <= 0) return(- Inf)
+     if (lambda <= 0) return(- Inf)
+     logl <- sum(dgamma(x, shape = alpha, rate = lambda, log = TRUE))
+     lpri <- (1 / 2) * log(alpha * trigamma(alpha) - 1) - log(lambda)
+     return(logl + lpri)
+ }
```

The only tricky bit is that we define this function to be $-\infty$ off of the parameter space. This corresponds to the unnormalized posterior value of zero, and results in the Metropolis algorithm never taking a step into this region.

4.2 Try 1

```
> out <- metrop(lupost, initial = c(1, 1), nbatch = 1e4)
> print(out$accept)

[1] 0.0826

> print(out$time)

   user  system elapsed
  0.608   0.000   0.605
```

4.3 Try 2

Now we need to adjust the scale. We set the variance matrix $\mathbf{M}$ in the Metropolis algorithm to be $\tau^2\mathbf{I}$, where $\mathbf{I}$ is the identity matrix. Try $\tau = 0.5$.

```
> out <- metrop(out, scale = 0.5)
> print(out$accept)

[1] 0.2243

> print(out$time)

   user  system elapsed
  0.648   0.000   0.645
```

The idea is to get an acceptance rate to be roughly 20% (Section 3.3 above). Now we have done that.

5 Diagnostics

5.1 Time Series Plots

Figure 1 (page 9) shows the time series plot made by the R statement

```
> plot(ts(out$batch, names = c("alpha", "lambda")))
```

What we are looking for is no obvious trend. If the beginning of the run looks very different from the end, then the simulated Markov chain is nowhere near stationarity and we need to run longer. This looks o. k.

5.2 Autocorrelation Plots

Figure 2 (page 10) shows the autocorrelation plot made by the R statement

```
> acf(ts(out$batch, names = c("alpha", "lambda")), lag.max = 50)
```

The diagonal plots show the autocorrelation function

$$k \mapsto \frac{\operatorname{cov}\{g(X_i), g(X_{i+k})\}}{\operatorname{var}\{g(X_i)\}}$$

where $g(X_i)$ is the first or second coordinate of the state vector. The off-diagonal plots show the autocrosscorrelation function

$$k \mapsto \frac{\operatorname{cov}\{g_1(X_i), g_2(X_{i+k})\}}{\operatorname{sd}\{g_1(X_i)\} \operatorname{sd}\{g_2(X_i)\}}$$

where $g_1(X_i)$ is the first coordinate and $g_2(X_i)$ is the second coordinate of the state vector.

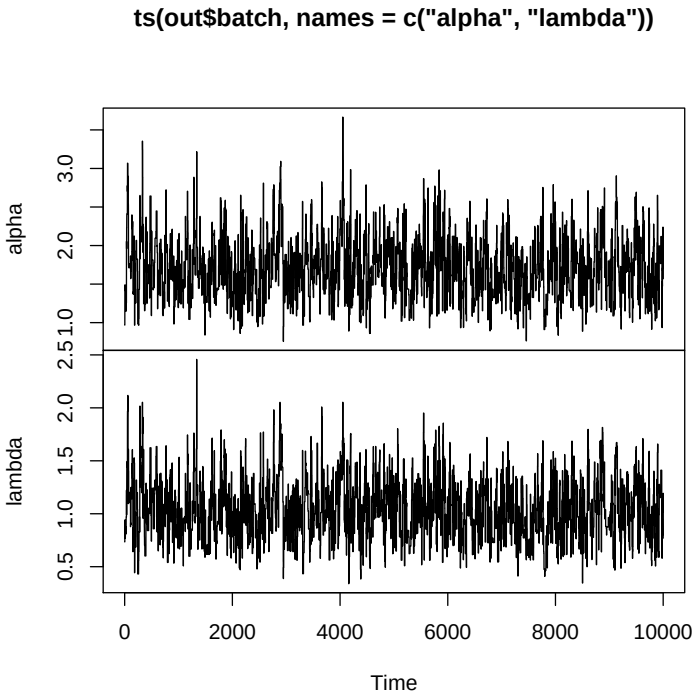

Figure 1: Time series plot of MCMC output.

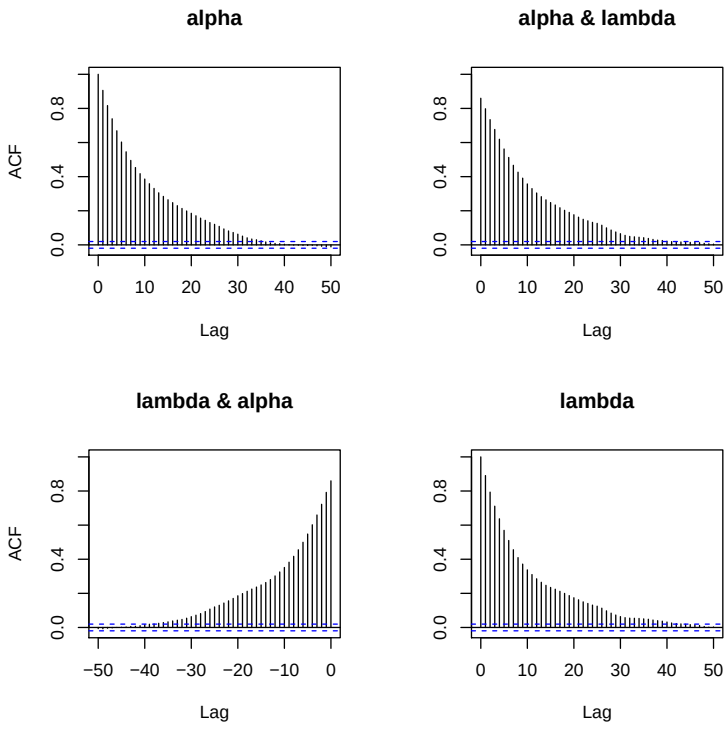

Figure 2: Autocorrelation plot of MCMC output.

What we are looking for is what lag the autocorrelations decrease to being not significantly different from zero (within the blue confidence region). It looks like about lag 40. We want batches for the method of batch means larger than that, perhaps much larger.

6 More Tries

```
> out <- metrop(out, blen = 200, nbatch = 500)
> print(out$accept)

[1] 0.22094

> print(out$time)

   user  system elapsed
  6.432   0.000   6.441
```

From this we can make frequentist confidence intervals for the posterior means of α and λ.

```
> alpha <- out$batch[ , 1]
> lambda <- out$batch[ , 2]
> t.test(alpha)

        One Sample t-test

data:  alpha
t = 291.0746, df = 499, p-value < 2.2e-16
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 1.663320 1.685927
sample estimates:
mean of x
 1.674624

> t.test(lambda)

        One Sample t-test

data:  lambda
t = 252.922, df = 499, p-value < 2.2e-16
```

```
alternative hypothesis: true mean is not equal to 0
95 percent confidence interval:
 0.9968048 1.0124127
sample estimates:
mean of x
 1.004609
```

The confidence intervals are fairly tight. We are done with our MCMC calculation if all we wanted to do is estimate the posterior means.

The package vignette for the `mcmc` package illustrates also calculating posterior variances and posterior standard deviations.

7 More Diagnostics

7.1 Time Series Plots

Figure 3 (page 13) shows the time series plot made by the R statement

```
> plot(ts(out$batch, names = c("alpha", "lambda")))
```

Again, what we are looking for is no obvious trend. If the beginning of the run looks very different from the end, then the simulated Markov chain is nowhere near stationarity and we need to run longer. This looks o. k.

7.2 Autocorrelation Plots

Figure 4 (page 14) shows the autocorrelation plot made by the R statement

```
> acf(ts(out$batch, names = c("alpha", "lambda")))
```

Note that this is the autocorrelation plot of the batch means (not of the individual $g(\mathbf{X}_i)$ that are averaged to make the batch means). If we have chosen the batch length long enough, the batch means will be nearly independent. So what we are looking for now is that the autocorrelations of the batches are not significant at all lags other than lag zero (where it is defined to be one for the diagonal plots). Looks o. k.

8 Marginal Distributions

8.1 Yet Another Try

Let also do a run without batches, so we can make histograms of the marginal distributions.

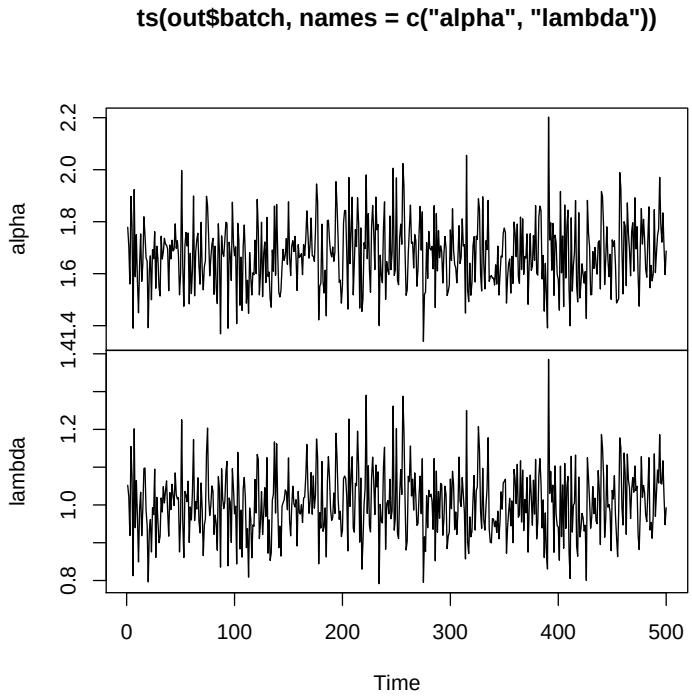

Figure 3: Time series plot of MCMC output. Batches of length 200

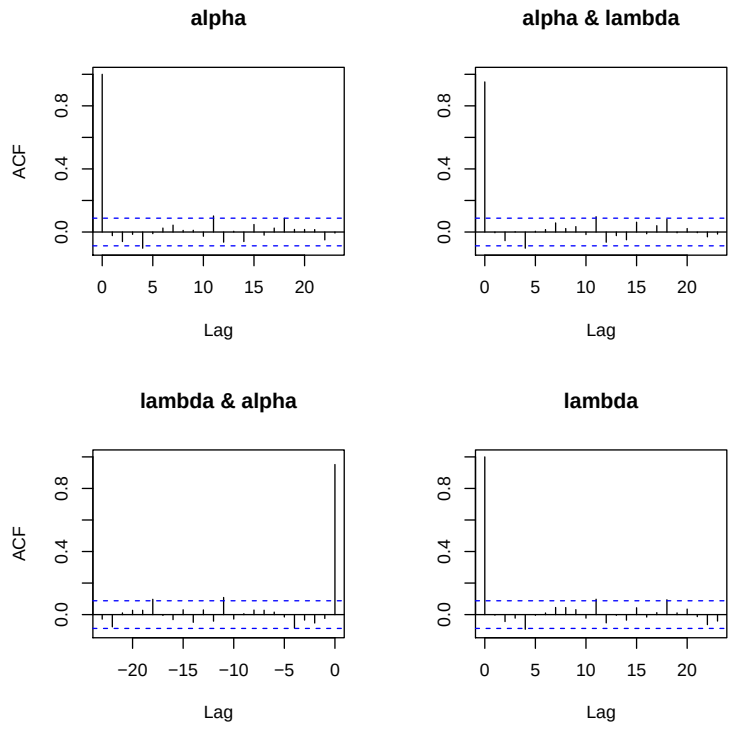

Figure 4: Autocorrelation plot of MCMC output. Batches of length 200

```
> out4 <- metrop(out, blen = 1, nbatch = 1e5)
> print(out4$accept)

[1] 0.22261

> print(out4$time)

   user  system elapsed
  6.448   0.004   6.458
```

8.2 Histograms

Figure 5 (page 16) shows the histogram made by the R statements

```
> alpha <- out4$batch[ , 1]
> hist(alpha, freq = FALSE)
```

Figure 6 (page 17) shows the histogram made by the R statements

```
> lambda <- out4$batch[ , 2]
> hist(lambda, freq = FALSE)
```

8.3 Smooth Density Plots

Figure 7 (page 18) shows the histogram made by the R statements

```
> out <- density(alpha)
> plot(out)
```

Figure 8 (page 19) shows the histogram made by the R statement

```
> out <- density(lambda)
> plot(out)
```

8.4 Smooth Density Plots Done Right

The plots produced in the previous section are undersmoothed because the "bandwidth selection" algorithm expects IID data rather than Markov chain output. We can find the correct bandwidth by giving the `density` function an approximately independent subsample of the data. From Figure 2 we see that lag 30 gives nearly uncorrelated data. So we subsample the output at spacing 30. Then we use the "bandwidth" calculated for the subsample to make a density plot using all the output.

Figure 9 (page 20) shows the histogram made by the R statements

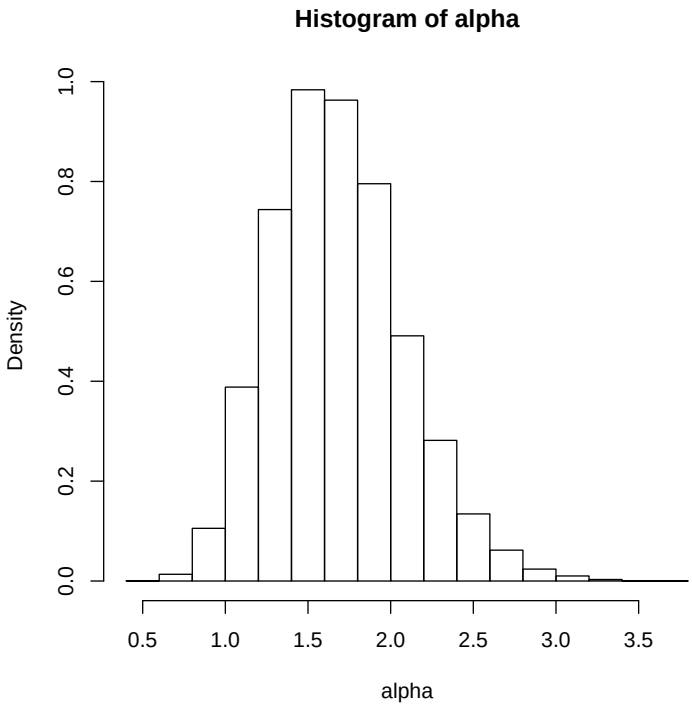

Figure 5: Histogram of marginal posterior for parameter α.

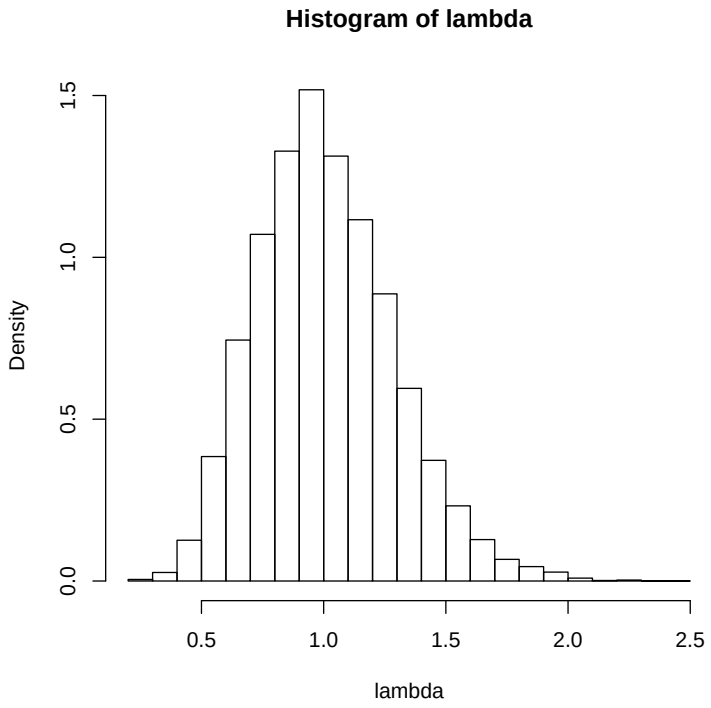

Figure 6: Histogram of marginal posterior for parameter λ.

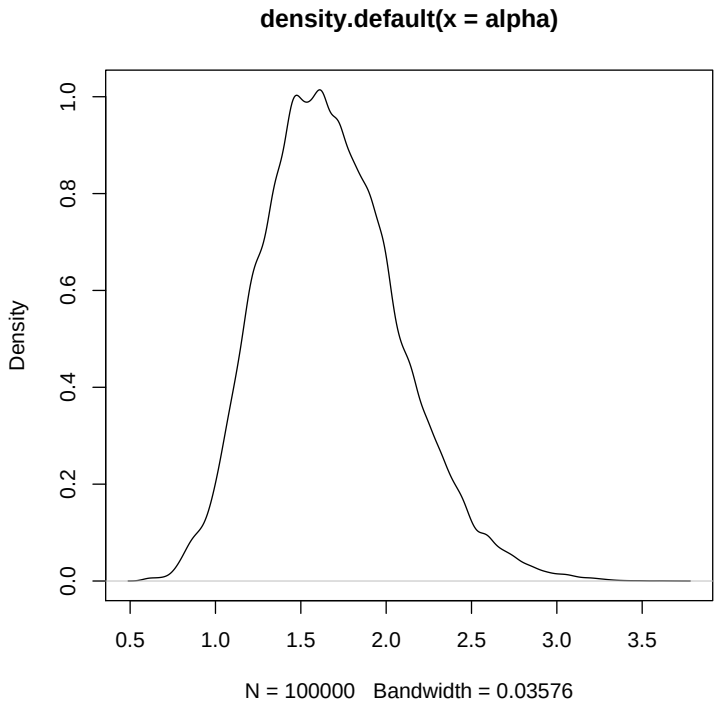

Figure 7: Smooth density plot of marginal posterior for parameter α.

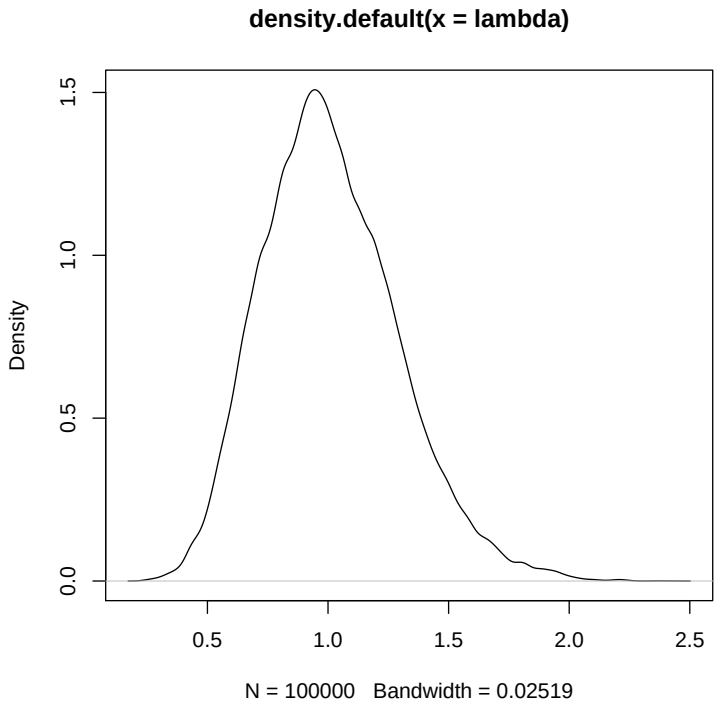

Figure 8: Smooth density plot of marginal posterior for parameter λ.

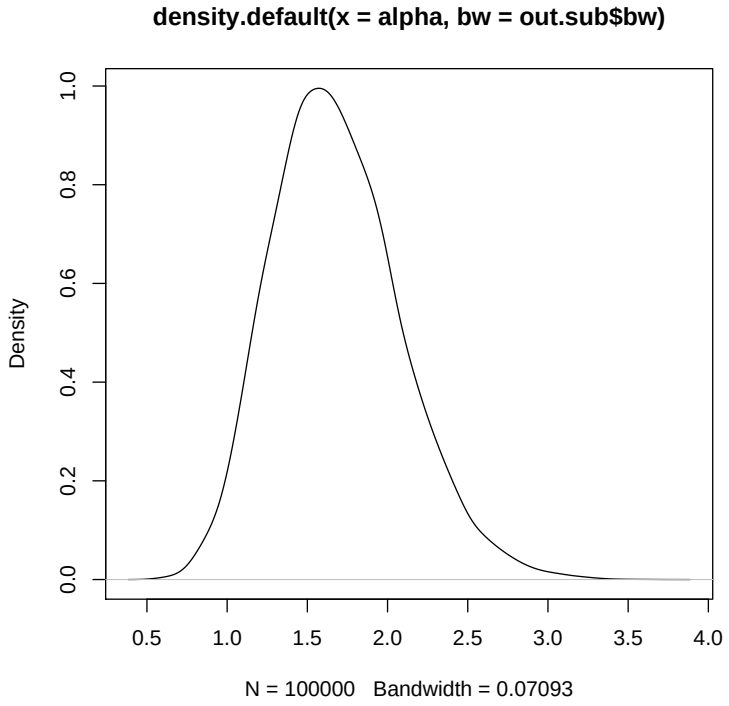

Figure 9: Smooth density plot of marginal posterior for parameter α. Bandwidth selected based on subsample with spacing 30.

```
> i <- seq(1, length(alpha), by = 30)
> out.sub <- density(alpha[i])
> out <- density(alpha, bw = out.sub$bw)
> plot(out)
```

Figure 10 (page 21) shows the histogram made by the R statements

```
> out.sub <- density(lambda[i])
> out <- density(lambda, bw = out.sub$bw)
> plot(out)
```

Figures 9 and 10 are much nicer. No unsightly and unbelievable bumps resulting from undersmoothing. Of course, these still have Monte Carlo sampling error. They are only approximations to the true posterior rather

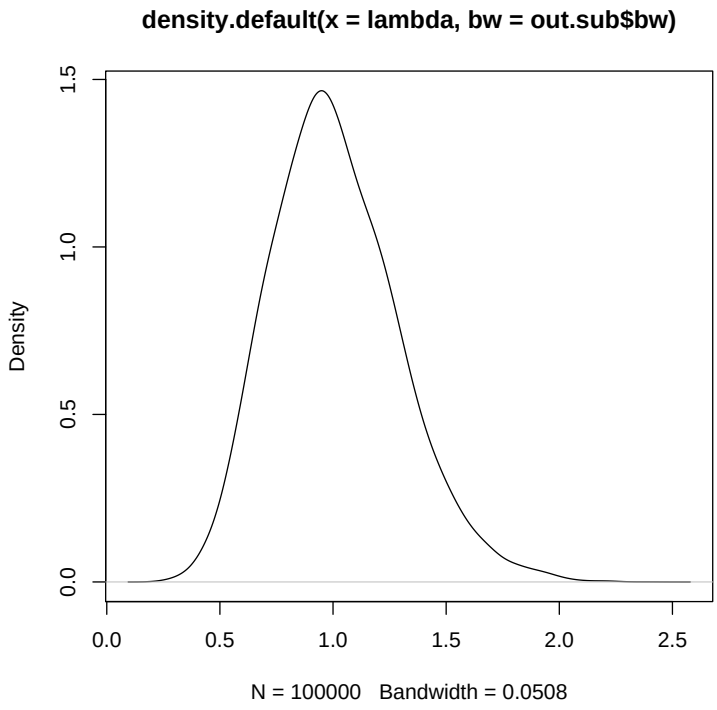

Figure 10: Smooth density plot of marginal posterior for parameter λ. Bandwidth selected based on subsample with spacing 30.

than exact, as can be seen by rerunning with a different random number generator seed.

9 Total Time

```
> proc.time()

   user  system elapsed
 15.648   0.068  15.843
```

These are times in seconds.

References

Geyer, C. J. (2011). Introduction to Markov chain Monte Carlo. In *Handbook of Markov Chain Monte Carlo*, edited by Brooks, S., Gelman, A., Jones, G., and Meng, X.-L., pp. 3–48. Chapman & Hall/CRC, Boca Raton, FL.