Lecture 23

Peter Shaffery

4/13/2021

Introduction

Last week we began looking at Bayesian statistics. As we saw, the central quantity in Bayesian analysis was the *posterior distribution*: a probability density (or mass) function that represents our beliefs about the true value of model parameters, given some set of observations:

$$P[\theta|y] \propto P[y|\theta]P[\theta]$$

Using this distribution, we saw a few examples of Bayesian analogues for the frequentist quantities that we've already learned.

Frequentist	Bayesian
MLE	Posterior Mean
Std. Error	Posterior St. Dev.
Confidence Interval	Credible Interval

Where the posterior mean (for example) is defined as:

$$E_{\theta|y}[\theta] = \int \theta P[\theta|y] d\theta$$

The idea of *averaging* quantities over the posterior turns out to be a very important pattern in Bayesian analysis. As we will see, it plays a role not only in estimating *multiple* parameters (ie. θ_1 and θ_2), but also in *forecasting* or *predicting* future observations y'.

However, if you're like me, you probably find taking integrals by hand to be quite painful. Moreover, sometimes it may not even be possible to compute $E_{\theta|y}[\theta]$ by hand! Because such integrals are so important to Bayesian statistics, we need a way to compute them, without having to always do a bunch of calculus.

Option 1- Numerical Integration

Last lecture we saw one way that we could approximately compute posterior integrals:

```r
library(palmerpenguins)
library(magrittr)
library(tidyverse)

adelie = penguins %>% filter(species=='Adelie') %>% drop_na
others = penguins %>% filter(species!='Adelie') %>% drop_na

h=.01
```

```r
theta.grid = seq(30,50,h)

log.prior = dnorm( theta.grid, mean=45, sd= 5, log=TRUE)
adelie.sd = adelie$bill_length_mm %>% sd

log.like = sapply(theta.grid,
                  function(theta){
                    sum(
                      dnorm(adelie$bill_length_mm, mean=theta, sd=adelie.sd,log=TRUE)
                      )
                    }
                  )

normalize = function(x){ return(x/sum(x*h)) }
post = exp(log.like + log.prior) %>% normalize
```

In the above code block the function `normalize` is computing the integral $\int P[y|\theta]P[\theta]d\theta$, the normalizing constant to ensure that $P[\theta|y]$ integrates to 1.

It is doing this is by applying the *trapezoidal rule*, a simple technique for approximating integrals.. If we have a function $f(\theta)$, then the trap rule approximates the integral of f by

$$\int_a^b f(\theta)d\theta \approx \sum_{k=1}^n h \times f(\theta_k)$$

Where $\theta_k = a + hk$.

The intuition here is that we are "gridding up" the values of θ from a to b, in intervals of h. We then "fill in" the area under $f(\theta)$ with rectangles of width h and height given by $f(\theta)$. The sum of the areas of those rectangles is approximately equal to the true area under the curve:

If we wanted to use this method to compute the posterior expectation for our example above, it would look like this:

```r
E.post = sum(h*theta.grid*post)
E.post
```

```
## [1] 38.83595
```

This method has the advantage of being relatively straightforward to compute. As you can see above, it can be implemented in just a few lines of computer code.

However it also has some drawbacks. For one, we need to always make sure that we're working with "true" probabilities. If (as I did last lecture) you drop a factor of h somewhere along the way then all of your answers will be wrong, because the $f(\theta_k)$ will be wrong.

Furthermore, once we start dealing with multiple parameters, computing multi-dimensional integrals using this gridded approach becomes non-trivial.

For this reason, most of time Bayesian statisticians uses another approach.

Option 2- Posterior Sampling

Why Posterior Sampling?

Ignoring the "how" for a second, say that we were able to draw random samples from our posterior density:

$$\theta^{(i)} \sim P[\theta|y]$$

If we could do that, then computing things like the posterior expectation would be super easy:

$$E_{\theta|y}[\theta] \approx \frac{1}{n}\sum_{i=1}^{n}\theta^{(i)}$$

Posterior variance could be computed similarly, and even credible intervals could be computed from the quantiles of our random samples.

For this reason, *posterior sampling* turns out to be the workhorse of Bayesian inference, perhaps comparable to some of the numerical optimization routines (eg. Newton-Raphson) we talked about for MLE estimators

Frequentist	Bayesian
MLE	Posterior Mean
Std. Error	Posterior St. Dev.
Confidence Interval	Credible Interval
Numerical Optimization	Posterior Sampling

Once you have posterior samples, approximating the quantities that you're after is pretty easy. The hard part is actually *how* you end up sampling $\theta^{(i)} \sim P[\theta|y]$. While functions like `rnorm` make this simple for basic posterior distributions, for a more complex posterior there's not going to be a simple function call available.

In practice, the answer is almost always "use a package" (we'll see one example later). However I want to quickly walk through two methods to give you the flavor of how sampling might work. One "quick and dirty" approach that you can implement by hand in a pinch, and a second, more sophisticated approach which forms the basis for most posterior sampling algorithms you'll see in the wild.

Gridded Sampling

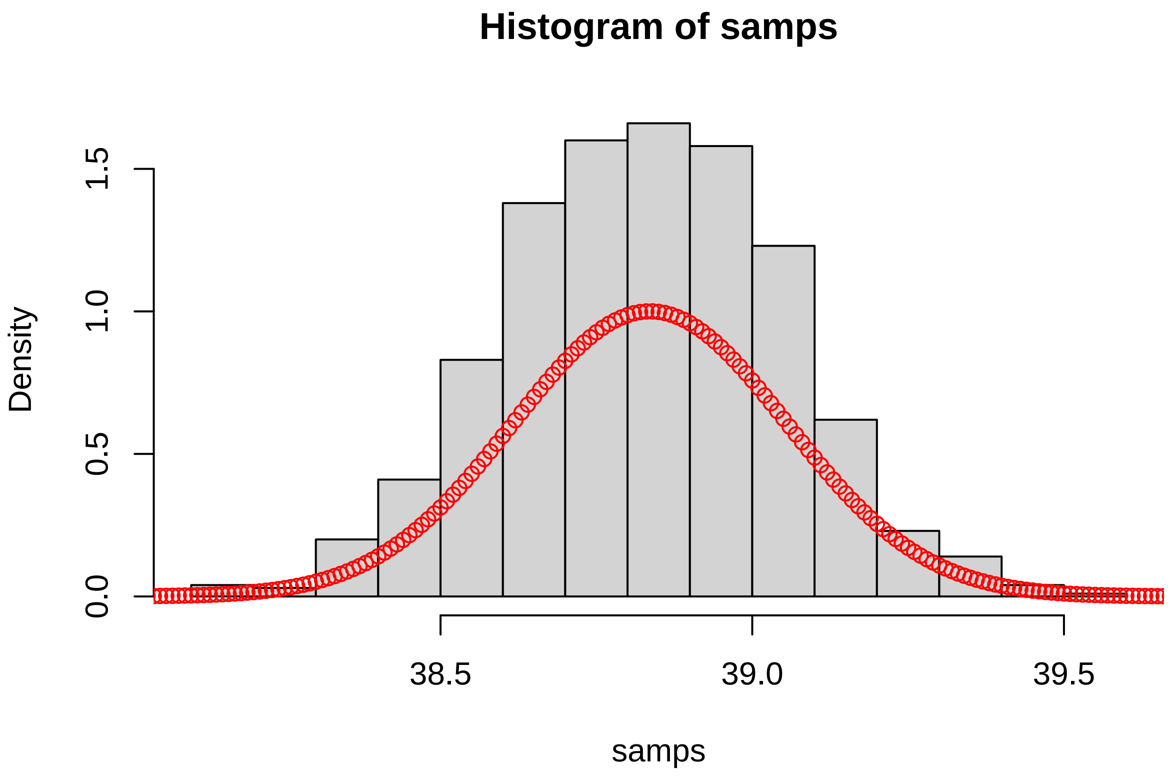
Gridded sampling is sort of a compromise between the numerical integration approach and true posterior sampling approach.

The idea here is approximate the probability *density* function $P[\theta|y]$ with a probability *mass* function that takes values only over a set of grid points θ_k.

In R, one way you might achieve this is:

```r
N = 1000

h=.01
theta.grid = seq(30,50,h)

log.prior = dnorm( theta.grid, mean=45, sd= 5, log=TRUE)
adelie.sd = adelie$bill_length_mm %>% sd

log.like = sapply(theta.grid,
                  function(theta){
                    sum(
                      dnorm(adelie$bill_length_mm, mean=theta, sd=adelie.sd,log=TRUE)
                      )
                    }
                  )

post = exp(log.like + log.prior)

samps = sample(theta.grid, N, replace=TRUE, prob=post)
```

```
hist(samps,freq=FALSE)
points(theta.grid,post/max(post),col='red')
```



```
mean(samps)
```

```
## [1] 38.83485
```

```
sd(samps)
```

```
## [1] 0.222555
```

```
quantile(samps,c(.025,.975))
```

```
##  2.5% 97.5%
## 38.39 39.26
```

Notice that we don't have to bother with h using this approach (outside of choosing our grid spacing). We simply evaluate the posterior density over our grid, and sample gridpoints accordingly.

Metropolis-Hastings

A much cooler algorithm, which forms the basis of pretty much all Bayesian sampling software you'll see in the wild, is the Metropolis-Hastings algorithm.

For this algorithm, we'll denote our posterior density function as $f(\theta)$, and we'll also introduce a *second* distribution $g(\theta^*;\theta^{(0)})$, which we'll call our **proposal** distribution.

This algorithm iterates through the following steps:

1. From a starting point $\theta^{(0)}$, draw a proposal point $\theta^* \sim g(\theta^*;\theta^{(0)})$
2. Compute the ratio $\alpha = \frac{f(\theta^*)g(\theta^{(0)};\theta^*)}{f(\theta^{(0)})g(\theta^*;\theta^{(0)})}$

3. Draw a Bernoulli variable U where $Pr[U=1] = \min(1,\alpha)$. If $U=1$, set $\theta^{(1)} = \theta^*$. Otherwise retain $\theta^{(1)} = \theta^{(0)}$
4. Loop from Step 1, using $\theta^{(1)}$ as the new starting point

As you run this algorithm longer and longer, the distribution of the points $\theta^{(k)}$ approaches $f(\theta)$.

A formal proof of this fact is beyond the scope of the course, but it can be intuitively understood by thinking about how the algorithm operates.

First, g is commonly chosen so that $g(\theta^*;\theta^{(0)}) = g(\theta^{(0)};\theta^*)$, so that α reduces to $\frac{f(\theta^*)}{f(\theta^{(0)})}$. For example, a typical choice is just:

$$g(\theta^*|\theta^{(k)}) = N(\theta^{(k)}, \omega^2 I)$$

This type of proposal distribution picks a random point from "nearby" the starting point $\theta^{(k)}$.

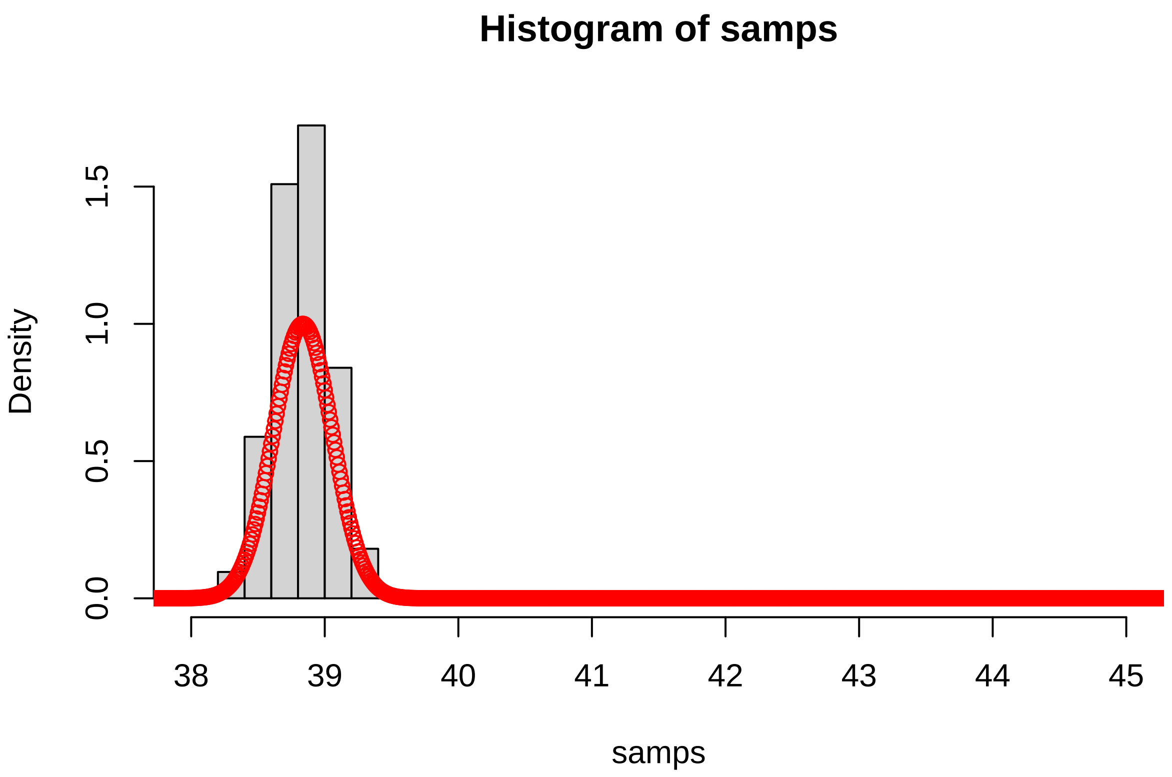
Now, given a proposal sampled from this distribution, there are two cases:

1. θ^* has *higher* posterior probability than our current point $\theta^{(k)}$. In this case we *always* set $\theta^{(k+1)} = \theta^*$.
2. θ^* has *lower* posterior probability than our current point $\theta^{(k)}$. In this case we *sometimes* set $\theta^{(k+1)} = \theta^*$.

The trajectory of points $\theta^{(k)}$ defined in this way forms a **random walk** across the values of θ, which spends most of its time around values of θ where $f(\theta)$ is large, but occasionally ventures out into the tails where $f(\theta)$ is small. The balance between these two behaviors is key to effective performance of this sampler.

```r
# a function to produce n Metropolis-Hastings samples from f
set.seed(777)
mh.sampler = function(log.f,n,theta0,omega=.1) {
  d = length(theta0)
  samps = matrix(0,nrow=n,ncol=d)

  for (i in 1:n) {
    theta.star = mvtnorm::rmvnorm(1,matrix(theta0,nrow=d),omega*diag(d)) %>% as.numeric

    log.alpha = log.f(theta.star) - log.f(theta0)

    U = runif(1)

    if (is.na(log.alpha)){
          samps[i,] = theta0

    }

    else if (log(U)<log.alpha) {
        samps[i,] = theta.star
        theta0 = theta.star
    }

    else{
      samps[i,] = theta0
    }
  }

  return(samps)
}

log.prior = function(theta) {dnorm(theta, mean=45, sd= 5, log=TRUE)}

adelie.sd = adelie$bill_length_mm %>% sd
```

```r
log.like = function(theta){ sum(dnorm(adelie$bill_length_mm, mean=theta, sd=adelie.sd,log=TRUE))}

log.post = function(theta){log.prior(theta)+log.like(theta)}

theta0=c(45)
samps = mh.sampler(log.post,10000,theta0,omega=.1)

hist(samps,freq=FALSE,breaks=30)
points(theta.grid,post/max(post),col='red')
```



```r
mean(samps)
```

```
## [1] 38.84559
```

```r
sd(samps)
```

```
## [1] 0.3431322
```

```r
quantile(samps,c(.025,.975))
```

```
##     2.5%    97.5%
## 38.41311 39.27985
```

Bayesian Analysis of Multiple Parameters

Now that we have some tools to perform more sophisticated analyses, let's see apply them to the case of multiple parameters.

To begin, let's say that we are interested in estimating two parameters: θ_1 and θ_2 from some observations y. Given these parameters, we can write the most general form of the **joint** posterior:

$$P[\theta_1, \theta_2|y] \propto P[y|\theta_1, \theta_2]P[\theta_1, \theta_2]$$

Pay close attention to the prior here: $P[\theta_1, \theta_2]$. Notice that this is a **joint distribution** over θ_1 and θ_2, which means that it may be the case that θ_1 is *not* independent from θ_2.

While there could be a could reason for your prior to include some relationship between these parameters, in most cases it's more reasonable to assume independence, which gives us the next most general form of the posterior:

$$P[\theta_1, \theta_2|y] \propto P[y|\theta_1, \theta_2]P[\theta_1]P[\theta_2]$$

Now, there are two important quantities that will be useful to define for our discussion: **conditional** and **marginal** posteriors.

The **conditional posterior** refers to quantities of the form:

$$P[\theta_1|\theta_2, y]$$

This represents our belief about the values of θ_1 *if the value of* θ_2 *were fixed and known.*

The **marginal posterior** is defined as the *integral* of the joint posterior, with respect to one of the parameters:

$$P[\theta_1|y] = \int P[\theta_1, \theta_2|y]d\theta_2$$

This represents our belief about the values of θ_1 *ignoring any knowledge about* θ_2.

It turns out that there is a close relationship between the marginal and conditional posteriors!

To see this, recall that we can rewrite any joint distribution $P[A, B] = P[A|B]P[B]$ (this was the justification behind Bayes theorem). We can apply this here to rewrite the marginalization integral:

$$\int P[\theta_1, \theta_2|y]d\theta_2 = \int P[\theta_1|\theta_2, y]P[\theta_2|y]d\theta_2$$

We can interpret the second integral as a *posterior expectation*:

$$P[\theta_1|y] = E_{\theta_2|y}[P[\theta_1|\theta_2, y]]$$

Thus our marginal belief about θ_1, what we believe about θ_1, *ignoring* θ_2, is a *mixture* of what we believe about θ_1 if θ_2 was fixed, *weighted* by how likely we believe it is for θ_2 to have that value.

Now, all of these quantities can be approximated using posterior sampling. Here I'll apply the Metropolis-Hastings sampler I wrote above to the problem of normal data with both parameters unknown.

Example- Normal Data, Unkwnown Mean and Variance

Say that we have n observations y_i $N(\theta, \sigma^2)$, and that we would like to estimate both θ and σ^2 together.

We start by assuming prior independence:

$$P[\theta, \sigma^2] = P[\theta]P[\sigma]$$

As we saw yesterday, there are a number of reasonable priors we might choose for these two parameters. For θ we'll keep things simple and apply a uniform prior.

Since θ can be any real number, technically such a prior is said to be **improper-** that is, the uniform distribution on $[-\infty, \infty]$ does not converge, and thus the distribution doesn't integrate to 1. Although this *prior* cannot be treated as a probability distribution, it turns out that the posterior is still integrable, and so we allow the prior to be used. Nevertheless, be aware that this will not always be the case! It's important, when using an improper prior, to double check that the posterior is still integrable.

For σ^2 we will use the prior $P[\sigma^2] \propto \frac{1}{\sigma^2}$, which is also an improper prior. This choice of prior might seem a little weird, but it turns out that this is equivalent to the prior $P[\ln \sigma^2] = \text{const}$, that is, a uniform prior on the scale of *log-variance* (see Appendix). This is a common choice of prior, because it allows us to apply a *uniform* (ie. non-informative) prior to a quantity which otherwise couldn't be negative (σ^2).

With these choices of prior, we have the join prior:

$$P[\theta, \sigma^2] \propto \frac{1}{\sigma^2}$$

The likelihood of the normal distribution we are already familiar with:

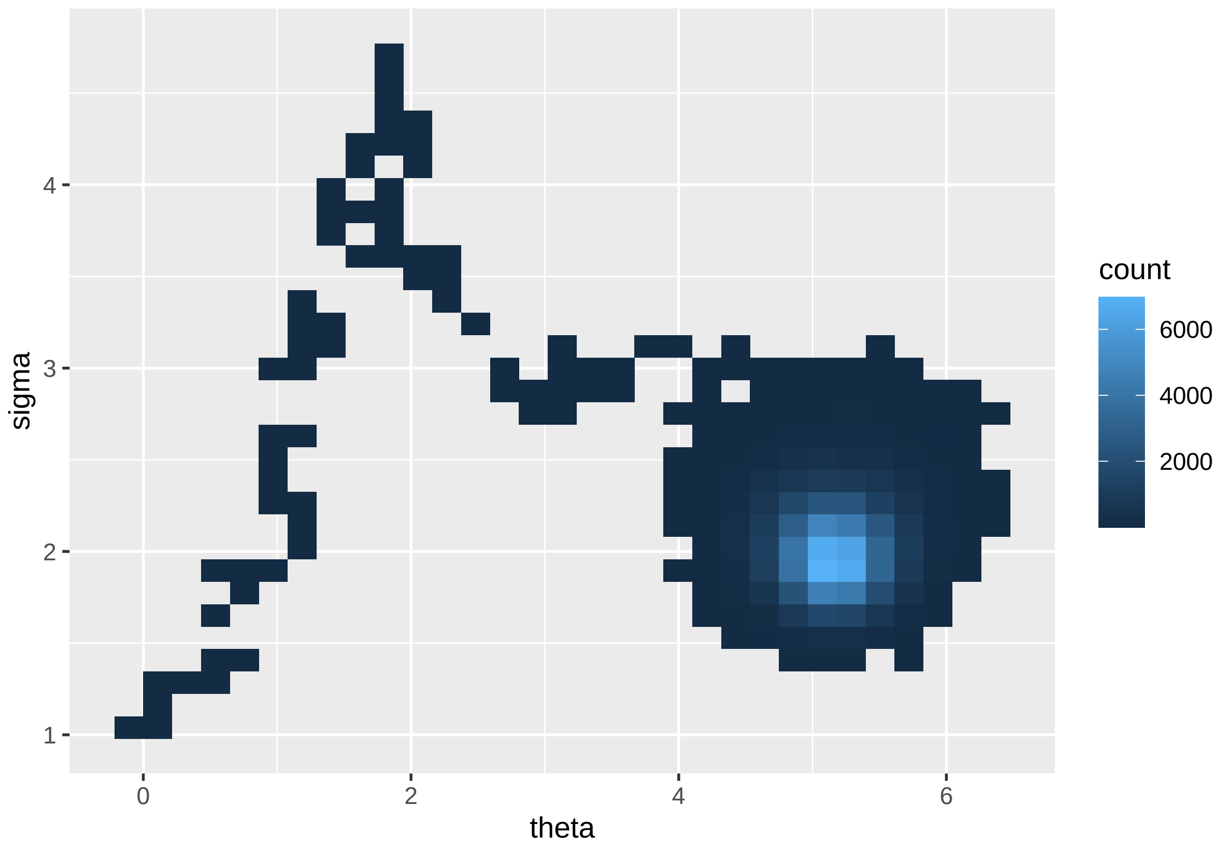
$$P[y|\theta, \sigma^2] \propto \prod_i^n \frac{1}{\sigma} \exp\left[\frac{1}{\sigma^2}(y_i - \theta)^2\right] = \frac{1}{\sigma^n} \prod_i^n \exp\left[\frac{1}{2\sigma^2}(y_i - \theta)^2\right]$$

Thus our posterior distribution is:

$$P[\theta, \sigma^2|y] \propto \frac{1}{\sigma^{n+2}} \exp\left[\frac{1}{2\sigma^2} \sum_i^n (y_i - \theta)^2\right]$$

Let's now apply our sampler to this problem

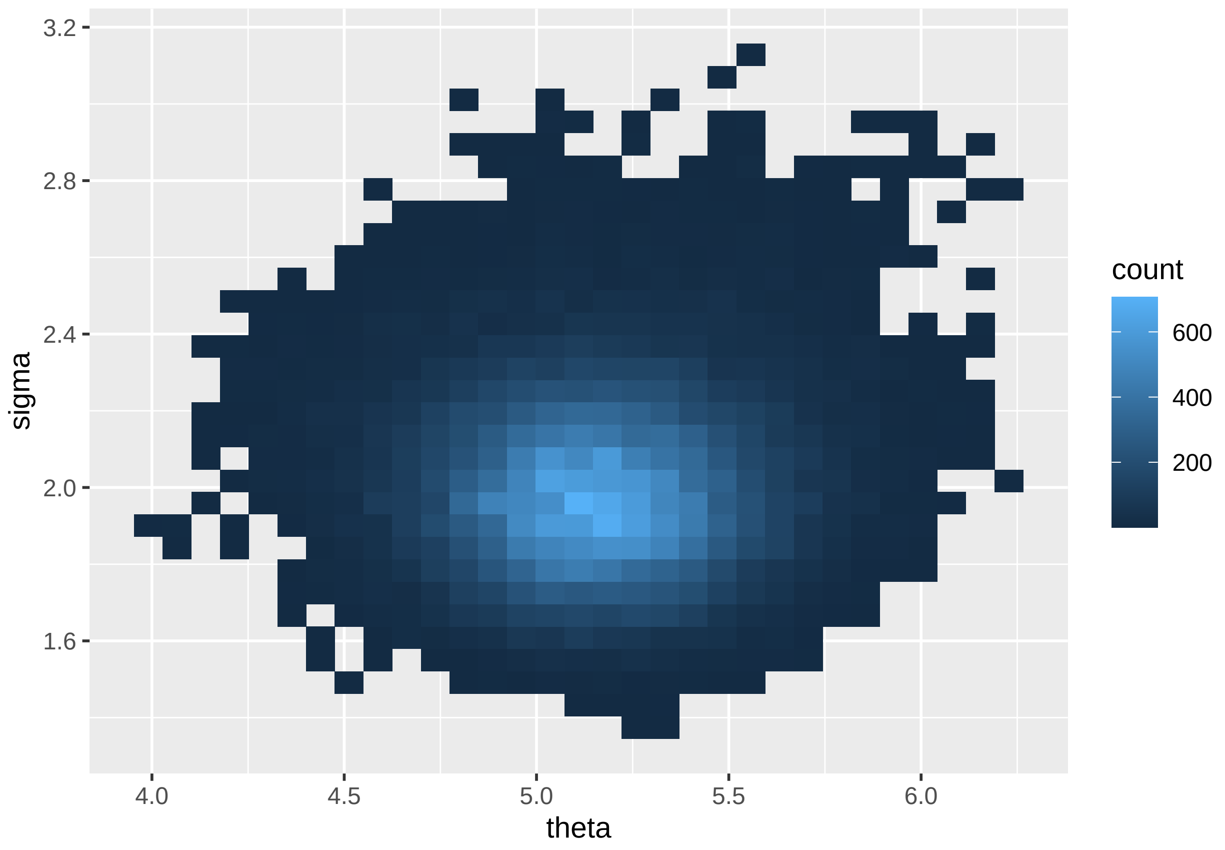
```r
set.seed(777)

theta = 5
sigma = 2
n = 50

y = rnorm(50, mean=theta, sd=sigma)

log.prior = function(parms){

  return(-2*log(parms[2]))
}

log.like = function(parms){
  l = dnorm(y,mean=parms[1],sd=parms[2],log=TRUE)

  return(sum(l))
}

log.post = function(parms){
  return(log.prior(parms)+log.like(parms))
}


parms0=c(0,1)
nsamps = 100000
samps = mh.sampler(log.post,nsamps,parms0,omega=.01)

plt.df = samps %>% as.data.frame
names(plt.df) = c('theta','sigma')

ggplot(plt.df, aes(x=theta,y=sigma)) + geom_bin2d()
```



Immediately we notice some very interesting behavior with the Metropolis-Hastings sampler. Observe that it takes a fairly large number of iterations for the sampler to "find" where the posterior density is. This process is sometimes called **burn-in**, and is quite typical of these algorithms.

Almost always we will therefore discard some initial chunk of our posterior samples, to avoid any bias in our final estimates caused by this burn-in period. A common choice is 50% of samples:

```
burned.samps = plt.df[(nsamps/2):nsamps,]
ggplot(burned.samps, aes(x=theta,y=sigma)) + geom_bin2d()
```



```
apply(burned.samps,2,mean)
```

```
##    theta    sigma
## 5.158043 2.002909
```

```
cov(burned.samps)
```

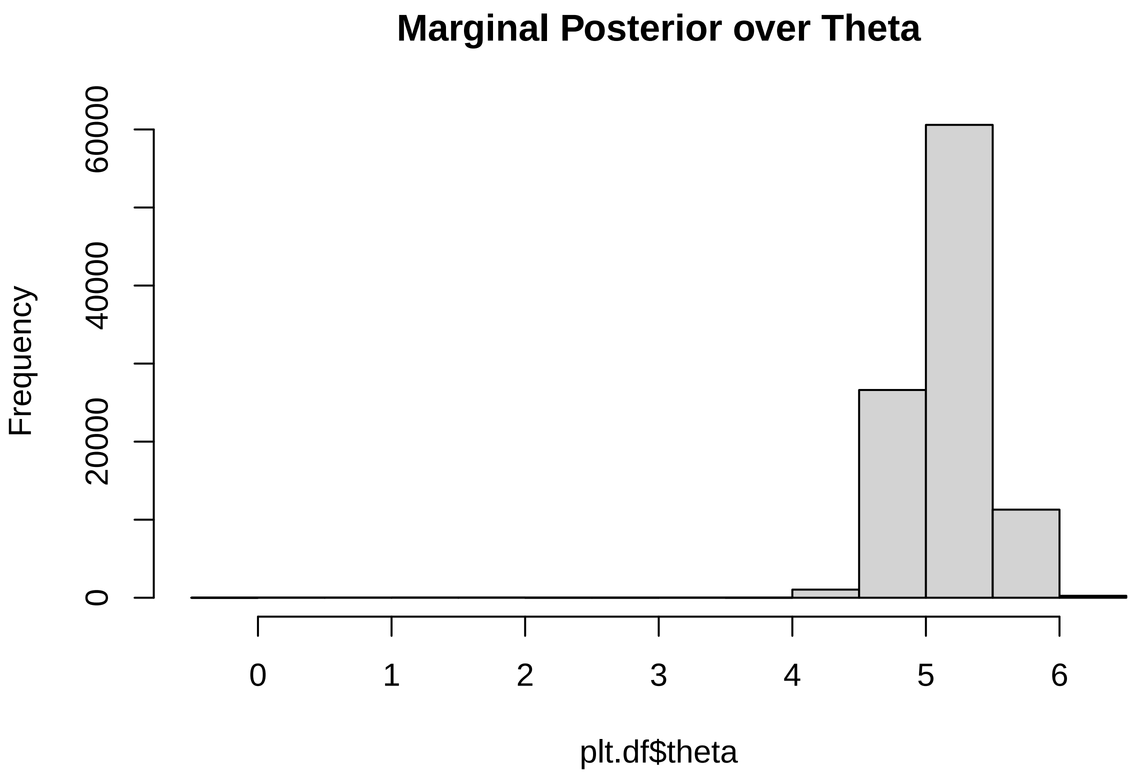
```
##              theta        sigma
## theta 0.0756445149 0.0008038936
## sigma 0.0008038936 0.0413565179
```

In the above we have our estimates of θ and σ, as well as our uncertainty in those estimates (the diagonal elements of the variance matrix). In addition, we have some estimates of the *posterior covariance*. This is an important quantity in many examples!

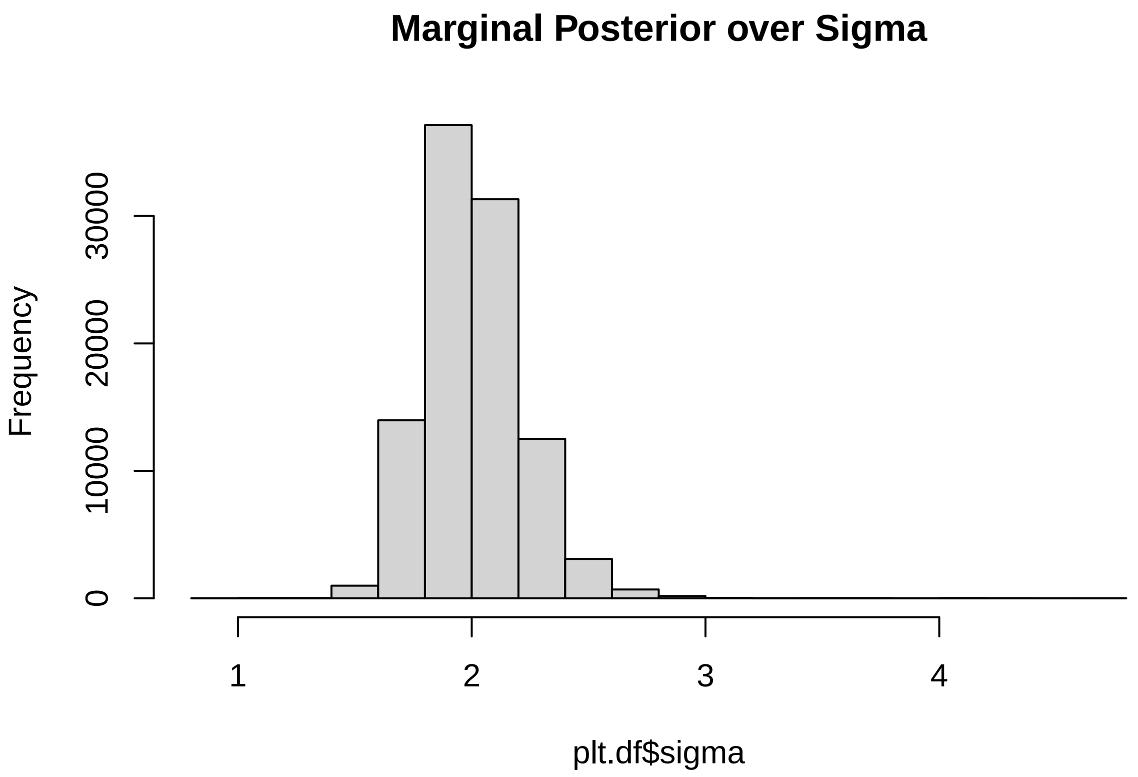
Posterior covariance reflects how our beliefs about the parameter valeus *covary*. In this example, that covariance is very small, so our belief about one parameter doesn't effect our belief about the other.

Once you have a set of samples, it's very easy to perform empirical marginalization. To do so you simply "drop" the dimension of the samples that you're marginalizing over:

```
hist(plt.df$theta,main='Marginal Posterior over Theta')
```



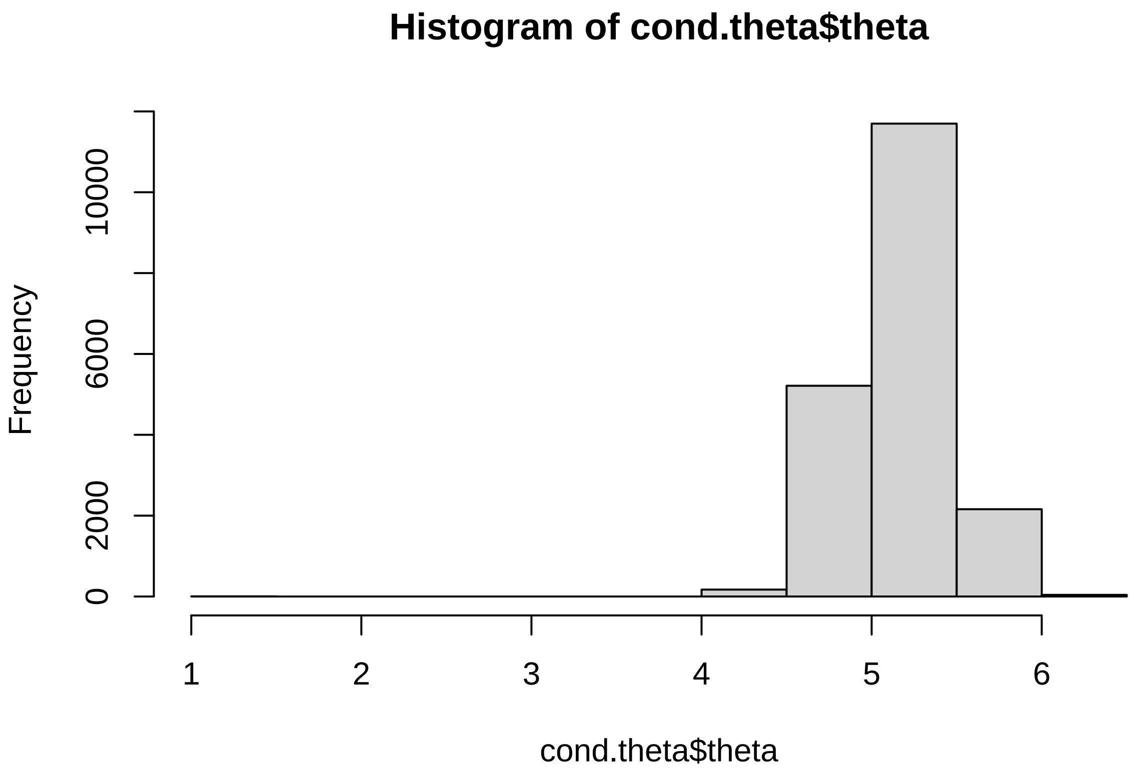
```
hist(plt.df$sigma,main='Marginal Posterior over Sigma')
```



Computing conditional posteriors is slightly more challenging, and requires a second layer of approximation:

```
sigma.mn = plt.df$sigma %>% mean
h=.05
```

```
cond.theta = plt.df %>% filter((sigma>sigma.mn-h)&(sigma<sigma.mn+h))

hist(cond.theta$theta)
```



As we can see (and as we suspected from the covariance matrix) θ and σ are nearly independent in the posterior.