Data 582 - Bayesian Inference

Lab 3: MCMC methods for Bayesian Inference

Contents

1 Introduction

So far we have seen examples that produced posterior distributions that have a well-known form. For more complicated models it may turn out that our posterior does *not* have a standard parametric form that we recognize. In these cases, obtaining posterior means, credible sets, or other numeric summaries of interest is not as straightforward as it was in previous considerations where we simply plugged in values into equations from Wikipedia/functions in R.

As discussed in lecture, the source of the problem essentially boils down to calculating integrals. There is a slurry of statistical procedures for approximating integrals. In this module we consider simulation-based (i.e. stochastic) methods for Bayesian fitting, but a number of deterministic methods are available which can sometimes produce acceptable numeric results. To include an example of a deterministic method relates to a question raised by a student in the MCMC lecture, *quadrature* rule methods attempt to approximate the posterior distribution by calculating $p(\theta \mid x)$ over a grid of plausible values for θ. By increasing the number of points we increase the accuracy of this procedure. In high dimension however, this procedure does not work well.

2 MCMC Algorithms

Here we explore how to use Markov chain Monte Carlo MCMC simulation techniques for approximating the posterior. While we we be demonstrating this on a problem for which we known the functional form of the posterior, the idea is that this method can be applied on posteriors that do not work out quite as nicely.

2.1 Pseudocode

Simulation forms a central part of much applied Bayesian analysis. Markov chain Monte Carlo (MCMC) methods comprise a class of algorithms for sampling from a probability distribution. In the context of Bayesian inference we wish to simulate from a Markov chain whose stationary distribution is our posterior distribution of interest.

Algorithm 1 Generic MCMC algorithm

```
1: At $t = 1$, initialize your chain at value $\theta^{(t)}$
2: Given current position, $\theta^{(t)}$, generate/propose a new value $\theta_{new} \sim q(\cdot \mid \theta^{(t)})$
3: Accept or reject $\theta_{new}$ according to some probability $r$.
4: Update the state of the chain; $\theta^{(t+1)} = \theta_{new}$ (if accepted) or $\theta^{(t+1)} = \theta^{(t)}$ (if rejected)
5: Set $t = t + 1$
6: Return to step 1
7: After a set number of iterations, return all accepted values.
```

Recall from lecture that the general set-up for MCMC algorithms matches the pseudocode presented in Algorithm 1. Assuming the regularity conditions are met (irreducible, aperiodic, positive recurrent) the Markov chain produced by this alogrithm will coverge to stationary distribution, $f(x)$ regardless of where the chain was initialized. The Markov chains that we will consider for MCMC are constructed in such a way that a they converge to some stationary distribution. For our use in the context of Bayesian statistics, this target, ie stationary distribution, will be a posterior.

3 Metropolis-Hastings Algorithm

The Metropolis algorithm was originally developed by Metropolis and co-authors in 1953 (the Metropolis Algorithm). It was then generalized by Hastings in 1970 (the Metropolis-Hastings Algorithm). As we will see, Metropolis is simply a special case of the more general Metropolis-Hastings algorithm. The difference comes down to how they generate new proposal values, θ_{new} for the chain. To put another way, the difference comes in the flexibility for specifying our proposal density q in Algorithm 1 Operation 2.

3.1 Proposal Distribution

For both the Metropolis and Metropolis-Hastings algoriths, the proposal densities depend on the current state of the chain. That is, we generate a proposed state θ_{new}, to move to in the chain, via proposal density $q(\theta_{new} \mid \theta^{(t)})$. When using the Metropolis-Hastings algorithm there is a great deal of flexibility in our choice of q. When q is a *symmetric distribution* such that $q(y \mid x) = q(x \mid y)$, we get the special case of the Metropolis algorithm. Popular choices include the normal, uniform and t-distribution. For instance we may define a normal proposal distribution with the mean parameter μ equal to the current state of the chain, and the standard deviation σ is set by the user. More succiently:

$$q(\theta_{new} \mid \theta^{(t)}) \sim \mathcal{N}(\theta^{(t)}, \sigma^2) \tag{1}$$

3.2 Proposal Width

The standard deviation of the proposal distribution is called the *proposal width*. This proposal width plays a significant role in terms of convergence. Larger values have will jump further and cover more space in the posterior distribution while smaller values cover less space and consequenty may take longer to reach convergence. Consequently, we must often tune this parameter such that appropriate "mixing" is obtained.

3.3 Acceptance Probability

In order to determine if a new proposed value is accepted we must define an acceptance probability function (Algorithm 1 Operation3). The Metropolis algorithm uses:

$$\alpha(\theta_{new} \mid \theta^{t}) = \min\left(\frac{f(\theta_{new})}{f(\theta^{(t)})}, 1\right) = \min\left(r(\theta_{new}, \theta^{(t)}), 1\right) \tag{2}$$

We then generate a random number $u \sim \mathcal{U}(0,1)$. If

$$\begin{cases} u \leq \alpha(\theta_{new} \mid \theta^{t}) & \text{accept move} \\ \text{otherwise} & \text{reject} \end{cases}$$

$f(x)$ is called the **target distribution**.Notice that any constant w.r.t θ will be canceled out in this ratio! Hence we can generate samples from the target distribution even when the target distribution is not normalized. To put another way, in the context of Bayesian inference, our target distribution is the posterior $p(\theta \mid x) \propto p(x \mid \theta)p(\theta)$. Hence we have

$$r(\theta_{new}, \theta^{(t)}) = \frac{p(\theta_{new} \mid x)}{p(\theta^{(t)} \mid x)} = \frac{\frac{p(x \mid \theta_{new})p(\theta_{new})}{\cancel{p(x)}}}{\frac{p(x \mid \theta_t)p(\theta^{(t)})}{\cancel{p(x)}}} = \frac{p(x \mid \theta_{new})p(\theta_{new})}{p(x \mid \theta_t)p(\theta^{(t)})} \tag{3}$$

Hence this relies only the likelihoods and the priors, both of which we can calculate easily. Accepting according to α implies that we are sampling from regions having higher posterior probability more often than not in a manner which fully reflects the probability of the data.

Recall that the beta distribution is a conjugate prior to the binomial, i.e. the resulting posterior is also a beta distribution. More specifically, if we adopt a prior $\theta \sim \text{Beta}(a, b)$ and we observe y successes in n Bernoulli trials, then our posterior has the form of $\theta \mid y \sim \text{Beta}(a + y, b + n - y)$. Let's pretend we don't know this posterior distribution result and instead use the Metropolis Algorithm to estimate the posterior and perform Bayesian inference on θ, the Binomial proportion.

4 Beta-Binomial MCMC

In lecture we talked about MCMC methods for sampling from the posterior in the event it doesn't have a function form that we recognize. As a first example, let's return to the Beta-Binomial model. Recall that the beta distribution is a conjugate prior to the binomial likelihood. That is, if our data comes in the form of an observed Y random variable where $Y \sim \text{Binomial}(n, \theta)$ having unknown probability of success parameter θ, then we can assume a Beta prior distribution for θ and arrive at a Beta posterior. This is often represented as:

$$p(\theta) \sim \text{Beta}(\alpha, \beta) \tag{4}$$
$$p(y \mid \theta) \sim \text{Beta}(y + 1, n - y + 1) \tag{5}$$
$$p(\theta \mid y) \sim \text{Beta}(\alpha + y, \beta + n - y) \tag{6}$$

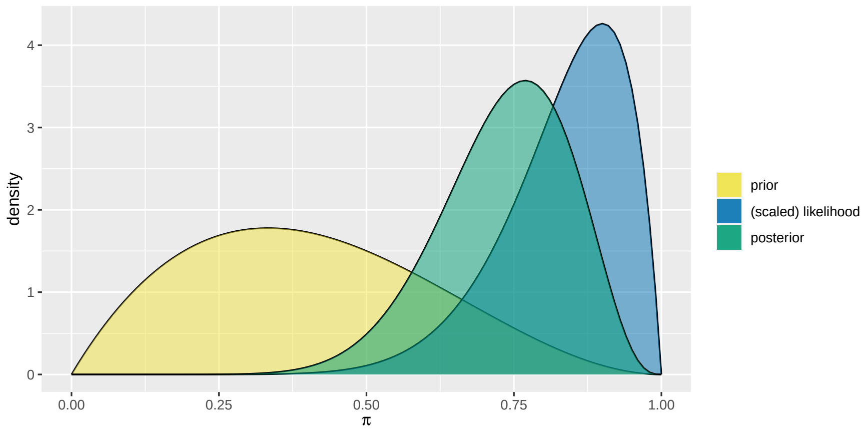
Now suppose we didn't know (6), that is we didn't know the functional form of the posterior. Then we could simulate approximate the posterior via Markov chain Monte Carlo (MCMC) simulation.

Let's give this problem some numbers:

$$\begin{aligned} \theta &\sim \text{Beta}(2, 3). \\ Y \mid \theta &\sim \text{Bin}(10, \theta) \end{aligned} \tag{7}$$

We can interpret Y as the number of successes in 10 independent trials. Each trial has probability of success θ where our prior understanding about θ is captured by a Beta(2, 3) model. Suppose we observe $y = 9$ successes. While we have worked out the conjugate posterior is Beta($y + \alpha$ = 11, $\beta + n - y$ = 4), let's pretend we didn't and generate 10,000 samples from the posterior using MCMC.

```
a = 2; b=3              # prior hyperparamters
y = 9; n = 10           # binomial data (y = # of success, n = # of trial)
aa = a+y; bb = b+n-y    # posterior parameter values
library(bayesrules)
plot_beta_binomial(alpha = a, beta = b, y = y, n = n)
```



4.1 Metropolis-Hastings Algorithm

Following the steps outlined in Algorithm 1:

1. Initialize the chain at some initial value of $\theta^{(0)}$ (I will call store this the first element of a vector called `chain`; since we know we want 10,000 observations, I will set it to a length 10,000)

```
# Initialization
t = 1
nIter = 10000 # number of iterations
chain =  vector(length=nIter) # {theta^1, theta^2, ..., theta^nIter}
chain[t] = 0.5 # initialization
```

2. Generate a proposed value $\theta^{(1)}$. We will use $\theta_{new} \sim N(\theta^{(t)}, \sigma)$ as our proposal distribution $q(\cdot \mid \theta^{(t)})$. Recall that σ is called out *proposal width*; we will start by setting it to 0.2 and assess the chains to determine if we need to tune it by setting it larger/smaller.

```
# set the proposal width:
pwidth = 0.1

# propose a new value for theta
theta.new <- rnorm(1, chain[t], pwidth)
```

3. Calculate acceptance probability

$$\begin{aligned}\alpha(\theta_{new} \mid \theta^t) &= \min\left(\frac{p(x \mid \theta_{new})p(\theta_{new})}{p(x \mid \theta^{(t)})p(\theta^{(t)})}, 1\right) \\ &= \min\left(r(\theta_{new}, \theta^{(t)}), 1\right) \\ &= \min\left(\frac{\text{likelihood}(\theta_{new}) \cdot \text{prior}(\theta_{new})}{\text{likelihood}(\theta^{(t)}) \cdot \text{prior}(\theta^{(t)})}, 1\right)\end{aligned}$$

With `a` = 2, `b` = 3, `y` = 9 and `n` = 10

```
# If theta.new falls outside of [0,1], then reject it with prob 1
if (theta.new>1 | theta.new<0){
  r <- 0
} else {
  # calculate the acceptance probability
  numerator <- dbeta(theta.new, a, b)*dbinom(y, n, theta.new)
  denom <- dbeta(chain[t], a, b)*dbinom(y, n, chain[t])
  r <- numerator/denom
  aprob <- min(r,1)
}
# compare alpha with a generated random number from U(0,1)
u <- runif(1)
```

4. Update the state of the chain. `t` = 1

```
# accept it with probability r
  if (u < aprob){
    # If the u > r, accept theta.new.
    chain[t+1] <- theta.new
  } else {
    # Otherwise, rejected theta.new,
    # i.e. we stay at the current state theta.t
    chain[t+1] <- chain[t]
  }
```

5. Set $t = t + 1$

```
t = t + 1
```

6. Repeat steps 2–5 `nIter -1` = 9999 times.

4.2 Posterior Estimation

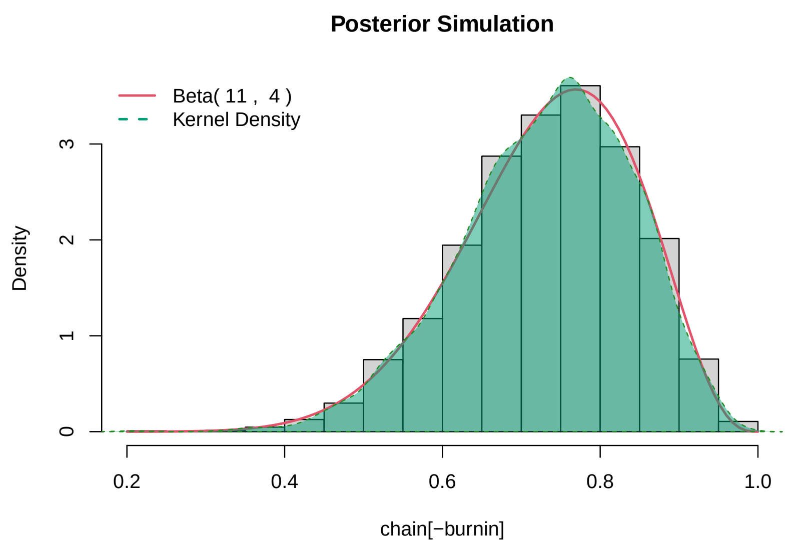
```r
source("Beta-BinomialMCMC.R")
```

I have saved the chain to an RData file and can load it using:

```r
load("BetaBinomialChain.RData") # loads in "chain"
```

Let's plot how well it approximates the true Beta(11, 4) posterior. I will remove the first 200 iterations of the chain (burn-in).

```r
a = 2; b=3            # prior hyperparamters
y = 9; n = 10         # binomial data (y = # of success, n = # of trial)
aa = a+y; bb = b+n-y  # posterior parameter values
burnin = 1:200
hist(chain[-burnin], freq = FALSE, main="Posterior Simulation")
curve(dbeta(x, aa, bb), add =TRUE, col = 2, lwd =2)
legend("topleft", lty=1:2, col =c(2,"#009e74") , bty = "n", lwd = 2,
       legend = c( paste("Beta(", aa, ", ", bb,")"), "Kernel Density"))

d <- density(chain[-burnin])
polygon(d, col=adjustcolor("#009e74", alpha.f=0.5), border='ForestGreen', lty = 2)
```



To compare our MCMC results with the theoretical ones, I'll define a function that calculates the mean and variance of a beta distribution:

```r
# define a function for finding the mean of a beta:
betaParams <- function(alpha, beta){
  mean = alpha/(alpha + beta)
  variance = (alpha*beta)/((alpha + beta)^2*(alpha+beta+1))
  out <- list(mean=mean, var = variance)
  return(out)}
```

Comparing the two values we see that they are very close:

```
# compare theoretical/ergodic mean/var:
# analytic mean/var
(pp <- betaParams(aa, bb))

## $mean
## [1] 0.7333333
##
## $var
## [1] 0.01222222

# ergodic mean/var
c(mean(chain[-burnin]), var(chain[-burnin]))

## [1] 0.73229774 0.01188459
```

We can compare the true 95% central credible intveral with the estimated:

```
(trueCI <- qbeta(c(0.025, 0.975), aa, bb))

## [1] 0.4920243 0.9161107

(estCI <- c(quantile(chain[-burnin], prob = 0.025),
quantile(chain[-burnin], prob = 0.975)))

##      2.5%     97.5%
## 0.5011535 0.9173080
```

Side note Some summary outputs are providve by the `summarize_beta_binomial` function from the **bayesrules** package:

```
summarize_beta_binomial(alpha = a, beta = b, y = y, n = n)

##       model alpha beta      mean      mode        var        sd
## 1     prior     2    3 0.4000000 0.3333333 0.04000000 0.2000000
## 2 posterior    11    4 0.7333333 0.7692308 0.01222222 0.1105542
```

1. Rerun the **Beta-BinomialMCMC.R** code with different values of `pdwidth`. See how this choice of tuning parameter affects the mixing of your chain. For instance, try σ = `pwidth` = 0.01 and σ = `pwidth` = 10 and examine your traceplots.

5 Normal-normal MCMC example

In lecture we also talked about the Normal-Normal conjugate family. I will review the results here with a bit more detail (feel free to skip and go straight to the example in

Section 5.5). The following assumes the Normal population parameter σ and known, and inference is done solely on μ. We denote a normal variable Y with mean μ and variance σ^2 (and thus standard deviation σ) by $Y \sim \mathcal{N}(\mu, \sigma^2)$ having pdf given by:

$$p(y) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y-\mu)^2}{2\sigma^2}\right)$$

Moving forward, let's denote the population mean by θ rather than μ to emphasize this is the parameter of interest and use "$\mid \sigma^2$" to denote that σ^2 is "given", or known, in the equations below. We need to identify the likelihood and define a prior distribution so we can calculate

$$p(\theta \mid \sigma^2, y_1, \ldots, y_n) \propto p(y_1, \ldots, y_n \mid \theta, \sigma^2) \times p(\theta \mid \sigma^2)$$

5.1 The Likelihood

The likelihood for normal samples can be written,

$$\begin{aligned} \mathcal{L}(\theta) = p(y_1, \ldots, y_n \mid \theta, \sigma^2) &= \prod_{i=1}^{n} p(y_i \mid \theta, \sigma^2) \\ &= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i-\theta)^2}{2\sigma^2}\right) \\ &\propto \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i-\theta)^2\right) \end{aligned} \tag{8}$$

5.2 The Prior

For $p(\theta \mid \sigma^2)$ to be a conjugate prior, the posterior needs to have the form $\exp(c_1(\theta - c_2)^2)$. Let the prior distribution be given as

$$\theta \mid \sigma^2 \sim \mathcal{N}(m, s^2)$$

Recall the notation that m and s^2 are the referred to as *hyperpameters* and they are set/chosen by us to match up with some prior belief.

5.3 The Posterior

The following was editted from Jesse Mu's notes available on his GitHub repo.

$$
\begin{aligned}
p(\theta \mid \sigma^2, y_1, \ldots, y_n) &\propto p(y_1, \ldots, y_n \mid \theta, \sigma^2) \times p(\theta \mid \sigma^2) \\
&\propto \exp\left(-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i-\theta)^2\right) \times \exp\left(-\frac{1}{2s^2}(\theta-m)^2\right) \\
&= \exp\left[-\frac{1}{2}\left(\frac{1}{s^2}(\theta^2-2\theta m+m^2)+\frac{1}{\sigma^2}(\sum y_i^2-2\theta y_i+n\theta^2)\right)\right] \\
&= \exp\left[-\frac{1}{2}\left(\left(\frac{1}{s^2}+\frac{n}{\sigma^2}\right)\theta^2+2\left(\frac{m}{s^2}+\frac{\sum y_i}{\sigma^2}\right)\theta+\left(\frac{m^2}{s^2}+\frac{1}{\sigma^2}\sum y_i^2\right)\right)\right] \\
&= \exp\left[-\frac{1}{2}\left(\frac{m^2}{s^2}+\frac{1}{\sigma^2}\sum y_i^2\right)\right] \times \exp\left[-\frac{1}{2}\left(\left(\frac{1}{s^2}+\frac{n}{\sigma^2}\right)\theta^2+2\left(\frac{m}{s^2}+\frac{\sum y_i}{\sigma^2}\right)\theta\right)\right] \\
&\propto \exp\left[-\frac{1}{2}\left(\left(\frac{1}{s^2}+\frac{n}{\sigma^2}\right)\theta^2+2\left(\frac{m}{s^2}+\frac{\sum y_i}{\sigma^2}\right)\theta\right)\right]
\end{aligned}
$$

The last line removes the constants of proportionality $c = \exp\left[-\frac{1}{2}\left(\frac{m^2}{s^2}+\frac{1}{\sigma^2}\sum y_i^2\right)\right]$. To simplify this, let $a = \dfrac{1}{s^2}+\dfrac{n}{\sigma^2}$ and $b = \dfrac{m}{s^2}+\dfrac{\sum y_i}{\sigma^2}$. By completing the square and removing any constants of proportionality we get:

$$
\begin{aligned}
p(\theta \mid \sigma^2, y_1, \ldots, y_n) &\propto \exp\left[-\frac{1}{2}(a\theta^2-2b\theta)\right] \\
&\propto \exp\left[-\frac{1}{2}(a\theta^2-2b\theta+b^2/a)+\frac{1}{2}b^2/a\right] && \text{Completing the square} \\
&\propto \exp\left[-\frac{1}{2}(a\theta^2-2b\theta+b^2/a)\right] && \text{rm constants of prop.} \\
&\propto \exp\left[-\frac{1}{2}a(\theta^2-2b\theta/a+b^2/a^2)\right] \\
&\propto \exp\left[-\frac{(\theta-b/a)^2}{2(1/a)}\right]
\end{aligned}
$$

We recognize the last line as having the functional form of a normal distribution having mean equal to b/a and variance equal to $1/a$. Denoting these posterior parameters by μ_n and σ_n^2, we write $\theta \mid \sigma^2, y_1, \ldots, y_n \sim \mathcal{N}(\mu_n, \sigma_n^2)$ where,

$$
\begin{aligned}
\mu_n &= \frac{b}{a} = \frac{\frac{1}{s^2}m+\frac{n}{\sigma^2}\bar{y}}{\frac{1}{s^2}+\frac{n}{\sigma^2}} & \sigma_n^2 &= \frac{1}{a} = \frac{1}{\frac{1}{s^2}+\frac{n}{\sigma^2}} \\
&= m\frac{\sigma^2}{ns^2+\sigma^2}+\bar{y}\frac{ns^2}{ns^2+\sigma^2} & &= \frac{s^2\sigma^2}{ns^2+\sigma^2} \qquad (9)
\end{aligned}
$$

We can summarize the above as follows:

$$
\begin{aligned}
Y_i &\sim \mathcal{N}(\mu, \sigma^2) \ (\sigma^2 \text{ known}) \\
\theta \mid \sigma^2 &\sim \mathcal{N}(m, s^2) \qquad (10) \\
\theta \mid \sigma^2, y_1, \ldots, y_n &\sim \mathcal{N}(\mu_n, \sigma_n^2)
\end{aligned}
$$

5.4 A note on the likelihood

Expanding the polynomial and completing the square, we can express the likelihood as follows:

$$\begin{aligned}
\mathcal{L}(\theta) &= \propto \exp\left[-\frac{1}{2\sigma^2}\sum_{i=1}^{n}(y_i-\theta)^2\right] \\
&\propto \exp\left[-\frac{1}{2\sigma^2}\left(\sum y_i^2 - 2\theta\sum y_i + n\theta^2\right)\right] && \text{expand the polynomial} \\
&\propto \exp\left[-\frac{n}{2\sigma^2}\left(\frac{\sum y_i^2}{n} - 2\theta\frac{\sum y_i}{n} + \theta^2\right)\right] && \text{factor out the n} \\
&\propto \exp\left[-\frac{n}{2\sigma^2}\left(\frac{\sum y_i^2}{n} - 2\theta\overline{y} + \theta^2\right)\right] && \text{sub in } \overline{y} \\
&\propto \exp\left[-\frac{n}{2\sigma^2}\left(\frac{\sum y_i^2}{n} - 2\theta\overline{y} + \theta^2 + \overline{y}^2 - \overline{y}^2\right)\right] && \text{add 0} \\
&\propto \exp\left[-\frac{n}{2\sigma^2}\left(\frac{\sum y_i^2}{n} + (\theta-\overline{y})^2 - \overline{y}^2\right)\right] && \text{complete the square} \\
&\propto \exp\left[-\frac{n}{2\sigma^2}(\theta-\overline{y})^2\right] && \text{remove constants of proportionality}
\end{aligned}$$

The last line tells us that **the likelihood of the normal random sample $y_1, \dots y_n$ is proportional to the likelihood of the sample mean of $\overline{y}$ and variance σ^2/n**, i.e.

$$y_1, \dots, y_n \mid \theta \sim \mathcal{N}(\overline{y}, \sigma^2/n) \tag{11}$$

Notice that the likelihood only depends on $\overline{y}$ rather than the individual observations $y_1, y_2, \dots, y_n$. That is, the individual observations $y_1, y_2, \dots, y_n$ do not provide any additional information about θ than knowing only $\overline{y}$ and $\overline{y} = \sum_{i=1}^{n} y_i$ is said to be a *sufficient statistic* for the likelihood.

5.5 Bayesian Updating Example

Before we jump into an MCMC let's go through a quick example which has been adjusted from Example 11.2 from Bolstad & Curran (2016)

Example 5.1. *Suppose we take a random sample of four observations from a normal distribution having mean μ and known variance $\sigma^2 = 1$. The observations are 3.2, 2.2, 3.6, and 4.1. Assuming a $N(0, 0.5^2)$ prior, obtain the posterior.*

From (9) and (10) we get the posterior

```r
# calculate the posterior parameters using eq (9)
mupost = m*sig^2/(n*s^2 + sig^2) + ybar*n*s^2/(n*s^2 + sig^2)
varpost = s^2*sig^2/(n*s^2 + sig^2)
sigpost = sqrt(varpost)
print(round(c(mupost, varpost, sigpost), 3))

## [1] 1.638 0.125 0.354
```

$$\text{Prior } \mu \sim \mathcal{N}(0, 0.25) \tag{12}$$
$$\text{Posterior } \mu \sim \mathcal{N}(1.638, 0.354^2) \tag{13}$$
$$\tag{14}$$

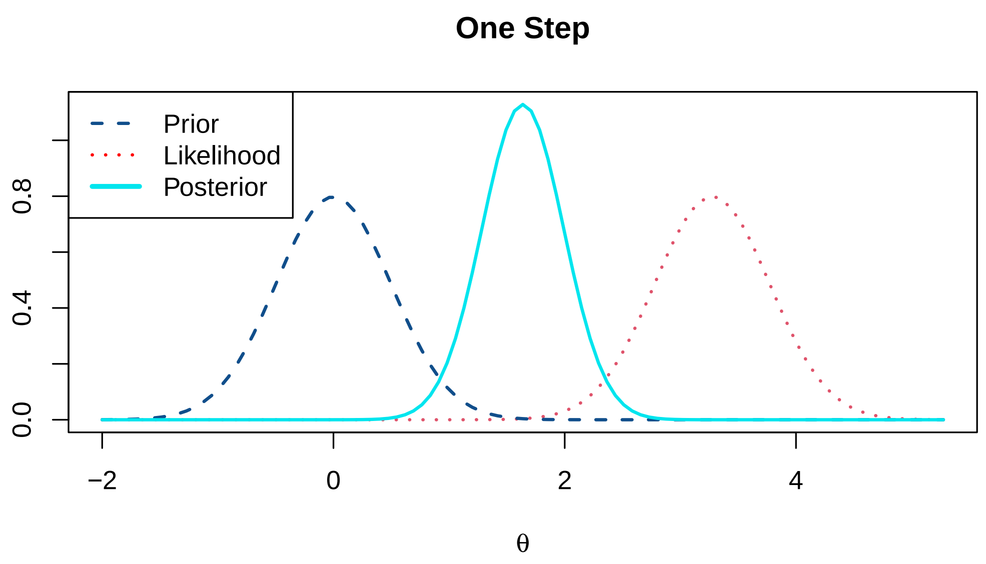
Before we plot the likelihood, prior, and posterior let's assign some variables:

```r
sigma2 = 1                      # known population variance
sig = sqrt(sigma2)              # known population SD
# data
ys = c(3.2, 2.2, 3.6, 4.1)      # sample data
ybar <- mean(ys)                # sufficient statistics
n <- length(ys)                 # sample size
# hyperparameters
m  = 0                          # prior mean
s = .5                          # prior sd
var.prior = s^2                 # prior variance
```

Coding the tri-plot using base R

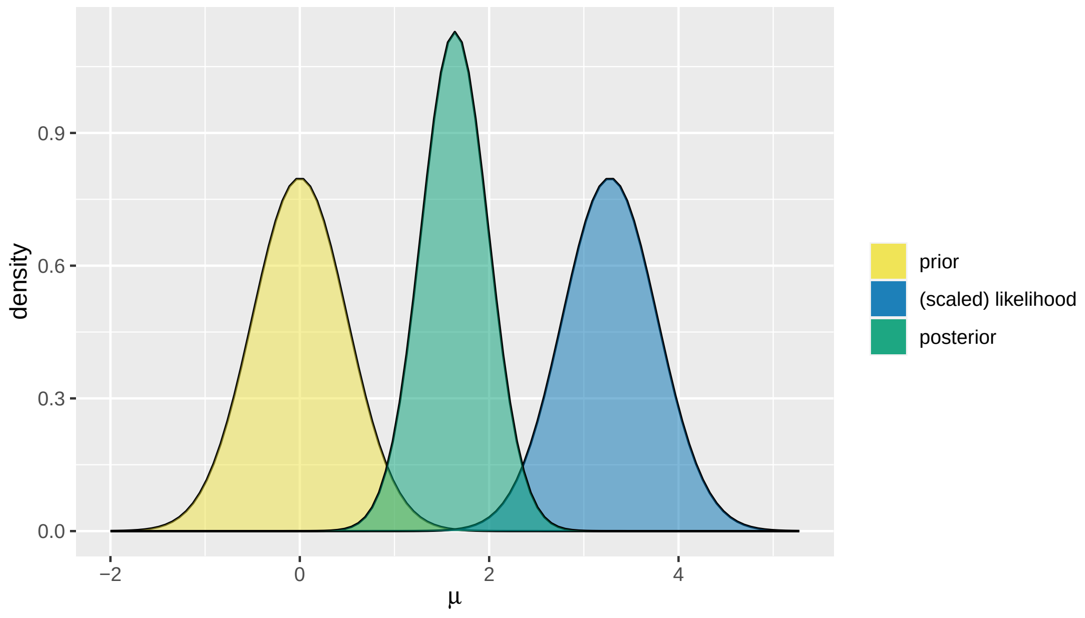
```r
# find reasonable plot boundaries
xrange <- c(
  min(c(ybar-4*(sqrt(sigma2/n)),
        m-4*sqrt(var.prior),
        mupost-4*sqrt(varpost))),
  max(c(ybar+4*(sqrt(sigma2/n)),
        m+4*sqrt(var.prior),
        mupost+4*sqrt(varpost)))
)


yrange <- c(0, max(dnorm(m, m, sqrt(var.prior)),
                   dnorm(mupost, mupost, sqrt(varpost)),
                   dnorm(ybar, ybar, sqrt(sigma2/n))))

# plot the likelihood N(theta, var/n)
```

```
curve(dnorm(x, ybar, sqrt(sigma2/n)), xlim=xrange, ylim=yrange,
      xlab = expression(theta), col=2, lwd=2, lty=3, ylab="",
      main="One Step")

# plot the prior
curve(dnorm(x,m, sqrt(var.prior)), col="dodgerblue4", lwd = 2, lty=2, add = TRUE)

# plot the posterior
curve(dnorm(x,mupost, sqrt(varpost)), col="turquoise2", lwd = 2, lty=1, add = TRUE)

legend("topleft", legend = c("Prior", "Likelihood", "Posterior"),
       lwd=c(2,2,3), lty=c(2,3,1), col=c("dodgerblue4", "red", "turquoise2"))
```



Plotting using **bayesrules**

```
library(bayesrules)
plot_normal_normal(mean = m,  sd = s, sigma = sig, y_bar = ybar, n = n)
```



As with the beta-binomial model, you can obtain summary statistics uings:

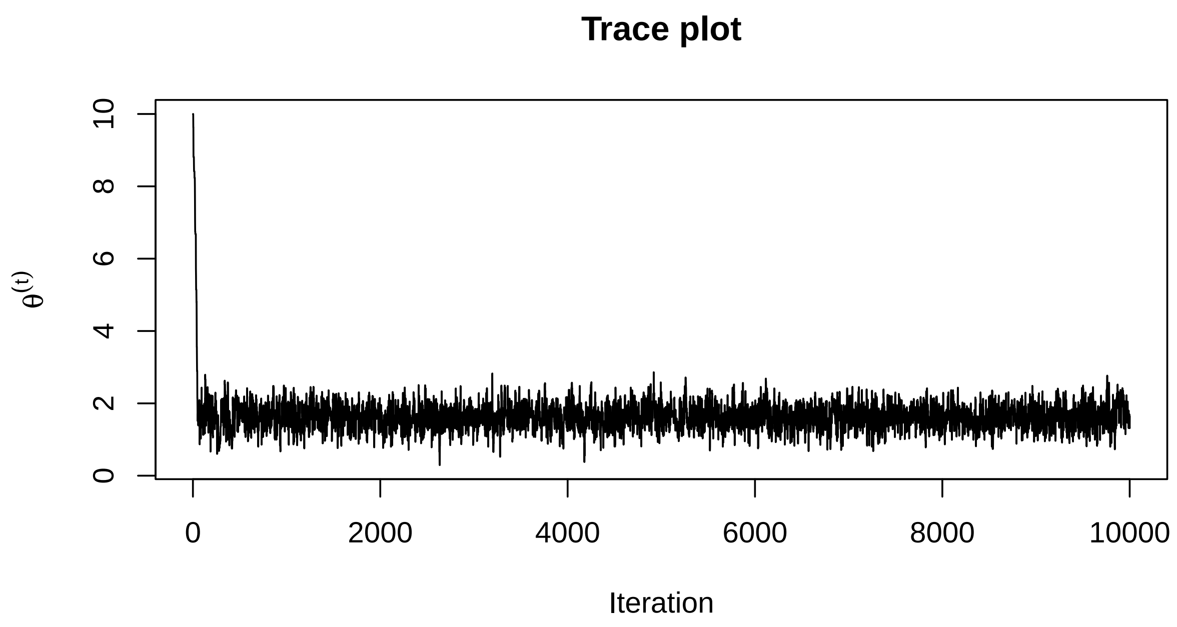
```
summarize_normal_normal(mean = m,  sd = s, sigma = sig, y_bar = ybar, n = n)

##       model   mean   mode   var        sd
## 1     prior 0.0000 0.0000 0.250 0.5000000
## 2 posterior 1.6375 1.6375 0.125 0.3535534
```

To sample from this posterio using an MCMC instead, we could use the code saved in **Normal-NormalMCMC.R**

```
source("Normal-NormalMCMC.R")
```

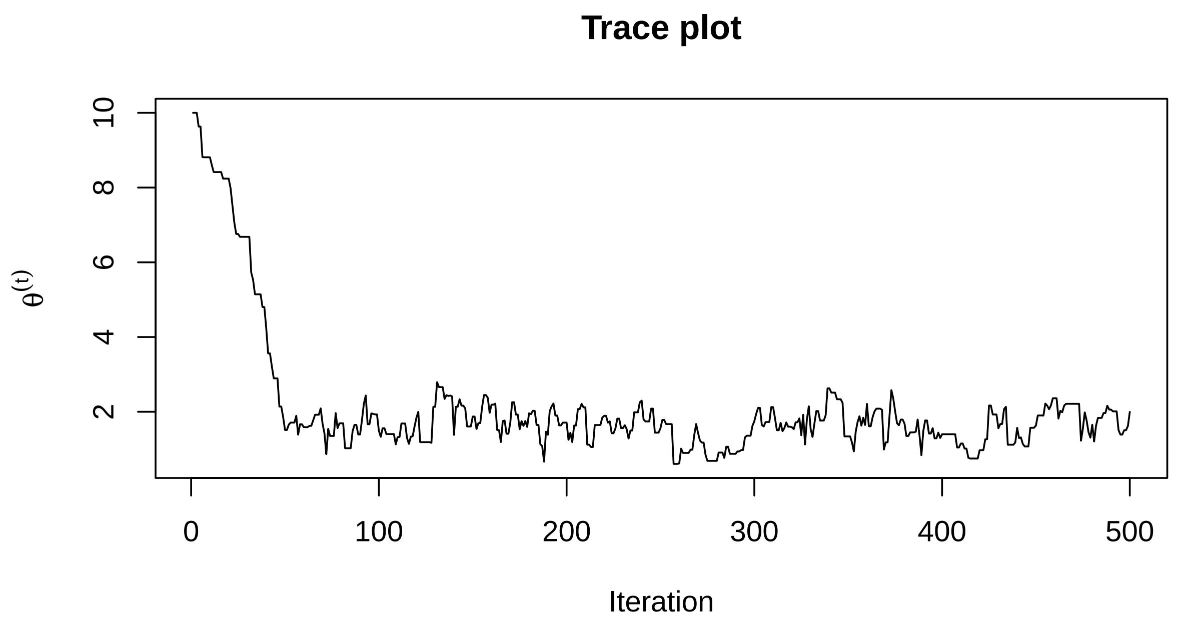
After running the above, you should have an MCMC chain stored in the vector called `chain`. Let's see if it looks like a fat and hairy caterpillar...

```
plot(chain, type="l", ylab=expression(theta^(t)), xlab="Iteration", main ="Trace plot")
```



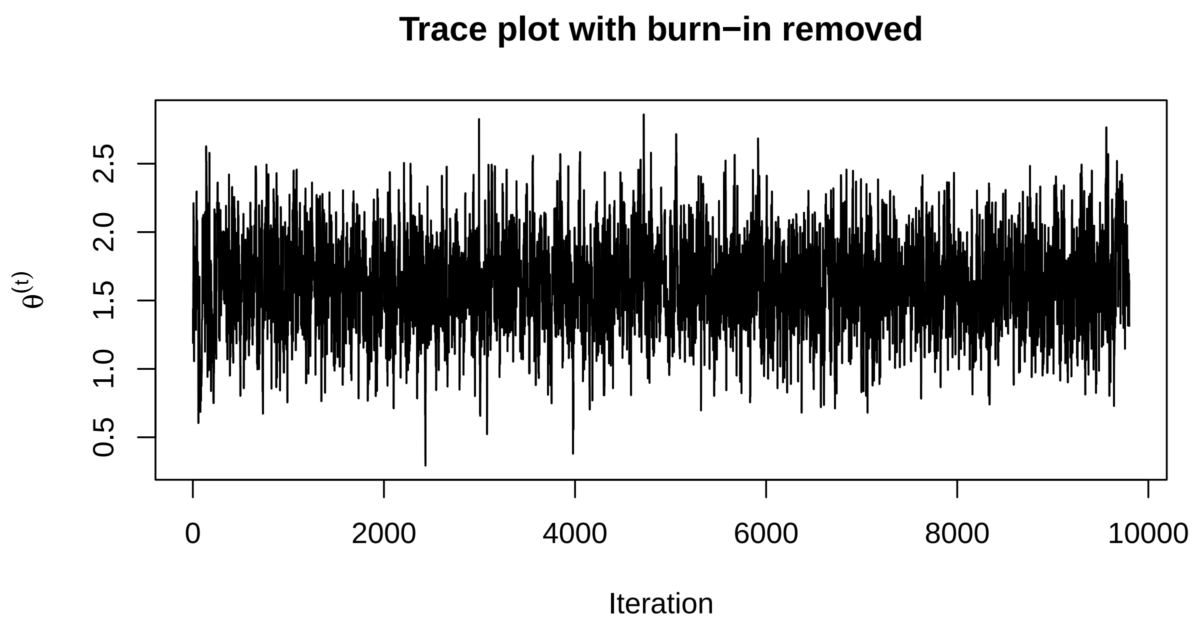
Looks good! Let's zoom in to see how much burnin we need to get rid of:

```
plot(chain[1:500], type="l", ylab=expression(theta^(t)), xlab="Iteration", main ="Trace
```



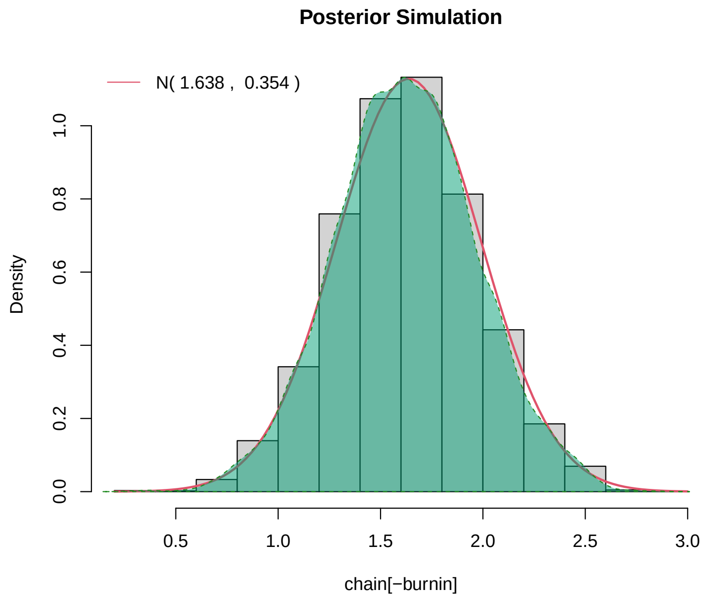
This chain reached the converged state pretty quickly, say in 50 iterations? Just to be on the safe side, let's remove the first 200 states, thus our burn-in period is iterations 1-200.

```
burnin = 1:200
plot(chain[-burnin], type="l", ylab=expression(theta^(t)), xlab="Iteration",
     main ="Trace plot with burn-in removed")
```



Let's plot the values in the chain and compare them with the analogical solutions summarized in (13):

```
hist(chain[-burnin], freq = FALSE, main="Posterior Simulation")
mupost = m*sig^2/(n*s^2 + sig^2) + ybar*n*s^2/(n*s^2 + sig^2)
sigpost = sqrt(s^2*sig^2/(n*s^2 + sig^2))
varpost = sigpost^2
curve(dnorm(x, mupost, sigpost), add =TRUE, col = 2, lwd = 2)
legend("topleft", lty=1, col =2 , bty = "n",
       legend = paste("N(", round(mupost,3), ", ", round(sigpost,3),")"))
d <- density(chain[-burnin])
polygon(d, col=adjustcolor("#009e74", alpha.f=0.5), border='ForestGreen', lty = 2)
```



6 Coding the MCMC

6.1 Comment

The likelihood generally involves the product of many small numbers, which can lead to numeric underflow. For instance if you attempt to initialize the **Normal-NormalMCMC.R** algorithm at `theta.t = 100` rather than `theta.t = 10` you will run into issues since the ratio used to find the acceptance probability α will produce `NaN` since we are dividing by 0. That's because the calculation of the likelihood and prior at such extreme values produce a value of 0. For instance with $\theta^{(t)} = 100$ and a proposed value of $\theta_{new} = 100.5$ we get:

$$\begin{aligned} r = \frac{f(\theta_{new})}{f(\theta^{(t)})} &= \frac{\mathcal{L}(100.5) * prior(100.5)}{\mathcal{L}(100) * prior(100.5)} \\ &= \frac{\texttt{dnorm}(100.5,\ \texttt{mean = } \bar{y},\ \texttt{sd = } \sigma/\sqrt{n}) * \texttt{dnorm}(100.5,\ \texttt{mean = } m,\ \texttt{sd = } s)}{\texttt{dnorm}(100,\ \texttt{mean = } \bar{y},\ \texttt{sd = } \sigma/\sqrt{n}) * \texttt{dnorm}(100,\ \texttt{mean = } m,\ \texttt{sd = } s)} \\ &= \frac{\texttt{dnorm}(100.5,\ \texttt{mean = } 3.275,\ \texttt{sd = } 0.5) * \texttt{dnorm}(100.5,\ \texttt{mean = } 0,\ \texttt{sd = } 0.5)}{\texttt{dnorm}(100,\ \texttt{mean = } 3.275,\ \texttt{sd = } 0.5) * \texttt{dnorm}(100,\ \texttt{mean = } 0,\ \texttt{sd = } 0.5)} \\ &= \frac{0 * 0}{0 * 0} = \texttt{NaN} \end{aligned}$$

For numerical stability, we will almost always work on the log-scale so that we are not multiplying really small numbers together and potentially dividing by 0. Consequently, we will calculate a **log-acceptance ratio** and compare it to the **log of a uniform r.v.**.

$$\begin{aligned}\log \alpha(\theta_{new} \mid \theta^t) &= \min\left\{\log\left(\frac{f(\theta_{new})}{f(\theta_t)}\right), \log(1)\right\} \\ &= \min\left\{\log\left(f(\theta_{new})\right) - \log\left(f(\theta_t)\right), 0\right\}\end{aligned} \tag{15}$$

We then generate a random number $u \sim \mathcal{U}(0,1)$ and compare it with the acceptance probability:

$$\begin{cases}\log(u) \leq \log \alpha(\theta_{new} \mid \theta^t) & \text{accept move} \\ \text{otherwise} & \text{reject}\end{cases}$$

Once we adopt this change, the `NaN` result from above becomes:

$$\begin{aligned}\log r = \log &\left(\frac{\texttt{dnorm}(100.5,\ \texttt{mean = }3.275,\ \texttt{sd = }0.5) * \texttt{dnorm}(100.5,\ \texttt{mean = }0,\ \texttt{sd = }0.5)}{\texttt{dnorm}(100,\ \texttt{mean = }3.275,\ \texttt{sd = }0.5) * \texttt{dnorm}(100,\ \texttt{mean = }0,\ \texttt{sd = }0.5)}\right) \\ &= \log\left(\texttt{dnorm}(100.5,\ \texttt{mean = }3.275,\ \texttt{sd = }0.5)\right) + \log\left(\texttt{dnorm}(100.5,\ \texttt{mean = }0,\ \texttt{sd = }0.5)\right) \\ &\quad - \left[\log\left(\texttt{dnorm}(100,\ \texttt{mean = }3.275,\ \texttt{sd = }0.5)\right) + \log\left(\texttt{dnorm}(100,\ \texttt{mean = }0,\ \texttt{sd = }0.5)\right)\right] \\ &= \texttt{dnorm}(100.5,\ \texttt{mean = }3.275,\ \texttt{sd = }0.5, \texttt{log = TRUE}) \\ &\quad + \texttt{dnorm}(100.5,\ \texttt{mean = }0,\ \texttt{sd = }0.5, \texttt{log = TRUE}) \\ &\quad - \left[\texttt{dnorm}(100,\ \texttt{mean = }3.275,\ \texttt{sd = }0.5, \texttt{log = TRUE})\right. \\ &\quad \left.+ \texttt{dnorm}(100,\ \texttt{mean = }0,\ \texttt{sd = }0.5, \texttt{log = TRUE})\right] \\ &= -39103.18 - (-38711.9) = -391.2777\end{aligned}$$

For ease of demonstration, I have worked with the regular (non-log) acceptance ratios in the above examples. If you want to see an example of the same Normal-Normal model being coded up in the log-scale, I invite you to try the **normal-normal-logscale.R**. As you can see in that code, starting the algorithm at $\theta^{(t)}$ does not produce any numerical instabilities. N.B. When we initialize at 100, however, the chain takes longer to converge. Hence our burn-in period will be longer.

6.2 Relevant Packages

Alternatively we could have used RStan, the R interface for Stan. Stan does much of the heavy lifting for us, so you can build your model with relative small amounts of code. Follow the RStan Quick Start Guide. N.B. I was unable to install the MacOS R toolchain, so I had to follow the instructions found here: `https://thecoatlessprofessor.com/programming/cpp/r-compiler-tools-for-rcpp-on-macos/`