Chapter Thirteen

Ryan Honea

Exercise Two

In this exercise you will create a chain of Metropolis-Hastings samples, for a likelihood, $P(\theta|y)$, that is a standard Normal, using symmetric and asymmetric proposal densities. A new value in the chain is proposed by adding a randomly drawn value from the proposal density to the current value of the chain $\theta* = \theta^{(i)} + Q$. Use the likelihood ratio, equation (13.3), to assess the accepantance probability.

If you are using R you will find the following commands helpful. The sampled values are labeled `theta`.

(a) Create 1000 samples for θ using a Uniform proposal density $Q \sim U[-1, 1]$. Start the chain at $\theta^{(0)} = 0.5$. Monitor the total number of accepted moves (acceptance rate). Plot a history of the samples values and a histogram.
(b) Try the smaller proposal density of $Q \sim U[-0.1, 0.1]$ and the larger density of $Q \sim U[-10, 10]$. Explain the difference in the acceptance rates and chain histories.
(c) It has been suggested that an acceptance rate of arround 60% is ideal. Using a proposal density of $Q \sim U[-q, q]$, find the value of q that gives an acceptance rate of roughly 60%. Was it the most efficient value for q? How did you judge this?
(d) Plot the acceptance rate for $q = 1, ..., 20$.
(e) Using a Normal proposal density $Q \sim N(0, \sigma^2)$, write an algorithm that tunes the values of σ^2 after each iteration to give an acceptance rate of 60%. Base the acceptance rate on the last 30 samples.

Solution

Below, I write several functions for the completion of this question. The details are listed for each.

- `conditional(a, b, density, ...)` : This computes the conditional probability of two events occurring given some density.
- `MH_Sampler_Normal(n, theta_0, symmetric, sample_density, pmf, ...)`:

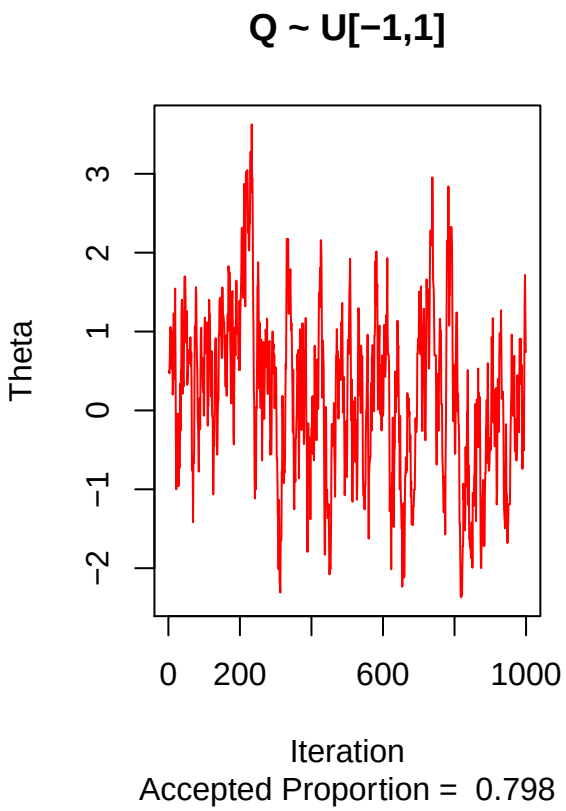
```
# not written y
conditional <- function(a, b, sample_density, ...) {
  return(1)
}


MH_Sampler_Normal <- function(n, theta_0, symmetric = TRUE,
                              sample_density = rnorm, pmf = dnorm, ...) {
  theta <- vector(n, mode = 'numeric')
  theta[1] <- theta_0
  theta.star <- vector(n, mode ='numeric')
  theta.star[1] <- theta[1] + sample_density(1, ...)
  accepted <- 0
  for (i in 2:n) {
    if (symmetric) {
      alpha = min(dnorm(theta.star[i-1])/dnorm(theta[i-1]),1)
    } else {
      alpha = min((dnorm(theta.star[i-1])/dnorm(theta[i-1]))*
                    (conditional(theta[i-1],theta.star[i-1],pmf,...)/
                       conditional(theta.star[i=1],theta[i-1],pmf,...)), 1)
```

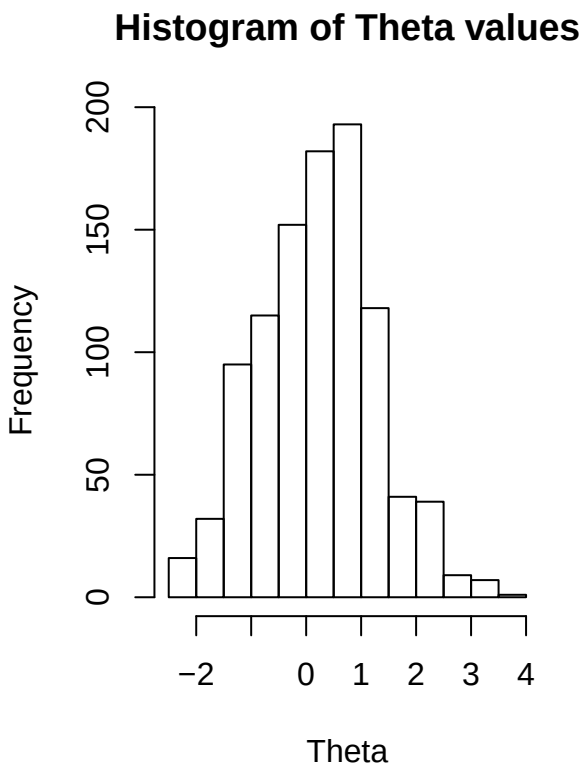
```
    }
    U <- runif(1,0,1)
    if (U < alpha) {
      theta[i] <- theta.star[i-1]
      accepted <- accepted + 1
    }
    else theta[i] <- theta[i-1]
    theta.star[i] <- theta[i] + sample_density(1, ...)
  }
  accepted_prop <- accepted/n
  df <- data.frame(theta = theta,
                   accepted = accepted,
                   proportion = accepted_prop)
  return(df)
}
```

Solution

(a): I now test with $Q \sim U[-1, 1]$ and plot the results.

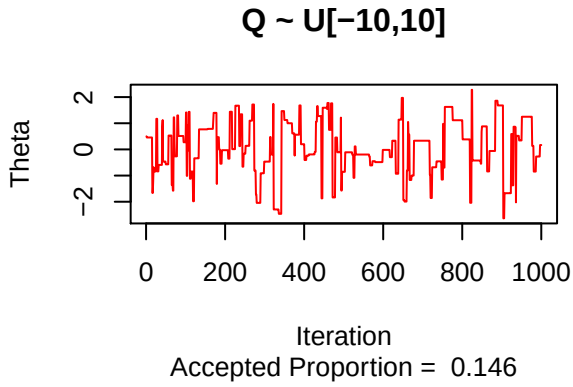
```
df <- MH_Sampler_Normal(1000, 0.5, sample_density = runif, pmf = dunif,
                        min = -1, max = 1)
par(mfrow = c(1,2))
plot(seq(1:1000),df$theta, type = "l", col = "red", xlab = "Iteration",
     main = paste("Q ~ U[-1,1]"), sub = paste("Accepted Proportion = ",df$proportion[1]),
     ylab = "Theta")
hist(df$theta, main = "Histogram of Theta values", xlab = "Theta")
```

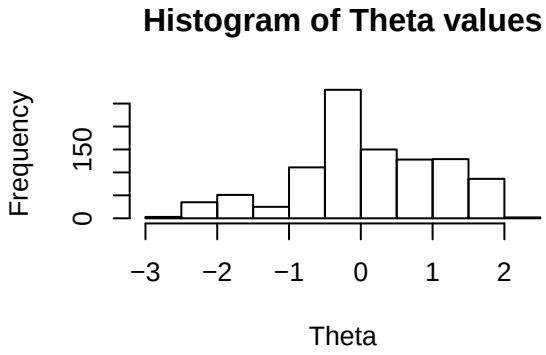


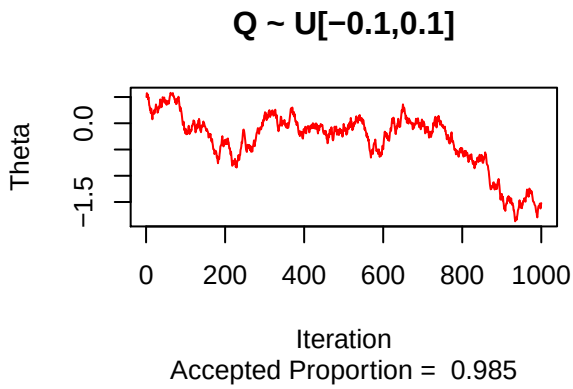


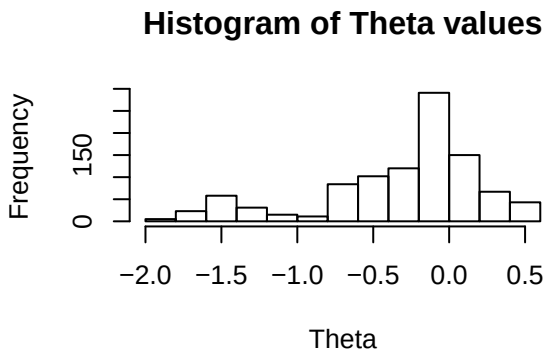
(b): We now do this with $Q \sim U[-10, 10]$ and $Q \sim U[-0.1, 0.1]$.

```
df <- MH_Sampler_Normal(1000, 0.5, sample_density = runif, pmf = dunif,
                        min = -10, max = 10)
df2 <- MH_Sampler_Normal(1000, 0.5, sample_density = runif, pmf = dunif,
                        min = -0.1, max = 0.1)
par(mfrow = c(2,2))
plot(seq(1:1000),df$theta, type = "l", col = "red", xlab = "Iteration",
     main = paste("Q ~ U[-10,10]"), sub = paste("Accepted Proportion = ",df$proportion[1]),
     ylab = "Theta")
hist(df$theta, main = "Histogram of Theta values", xlab = "Theta")

plot(seq(1:1000),df2$theta, type = "l", col = "red", xlab = "Iteration",
     main = paste("Q ~ U[-0.1,0.1]"), sub = paste("Accepted Proportion = ",df2$proportion[1]),
     ylab = "Theta")
hist(df2$theta, main = "Histogram of Theta values", xlab = "Theta")
```





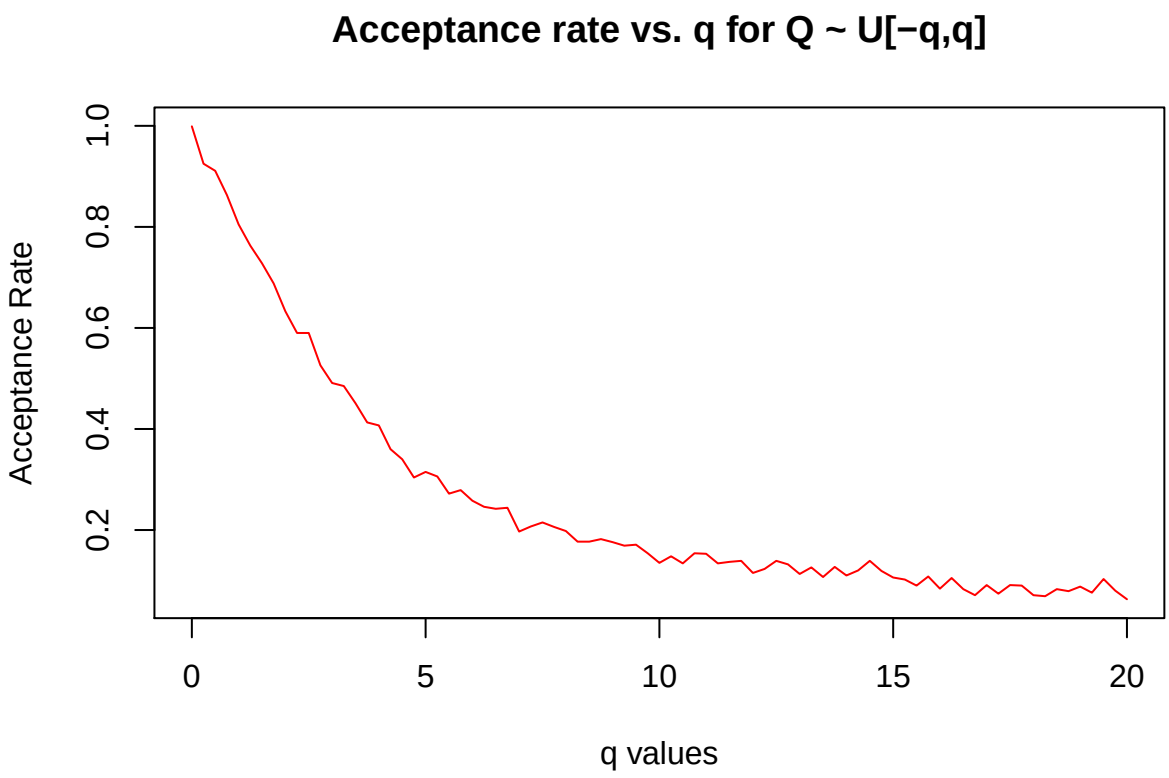




As would be expected, a wider range is less likely to be accepted than a lower range. Lower ranges will cause significantly less differences which would result in $\frac{P(\theta^*|y)}{P(\theta^{(i)}|y)}$ being low. Conversely, with high Q, that ratio is going to be high which would lead to unaccepted values. Both of these though don't necessarily approximate the standard distribution well.

(c) and (d): We now seek to find a q such that when $Q \sim U[-q, q]$ is our proposal density, the acceptance rate is approximately 60%. First, I'll graph acceptance rates with varying q. Although the text suggest $q = 1, 2, ..., 20$, I'd like to be more precise than that and do it from 0 to 20 in .25 intervals.

```
q <- seq(0,20,by=.25)
acceptance_rate <- vector(length(q), mode = "numeric")
for (i in 1:length(q)) {
  df <- MH_Sampler_Normal(1000, 0.5, sample_density = runif, pmf = dunif,
                          min = -q[i], max = q[i])
  acceptance_rate[i] <- df$proportion[1]
}
plot(q, acceptance_rate, type = "l", col = "red",
     ylab = "Acceptance Rate", xlab = "q values",
   main = "Acceptance rate vs. q for Q ~ U[-q,q]")
```

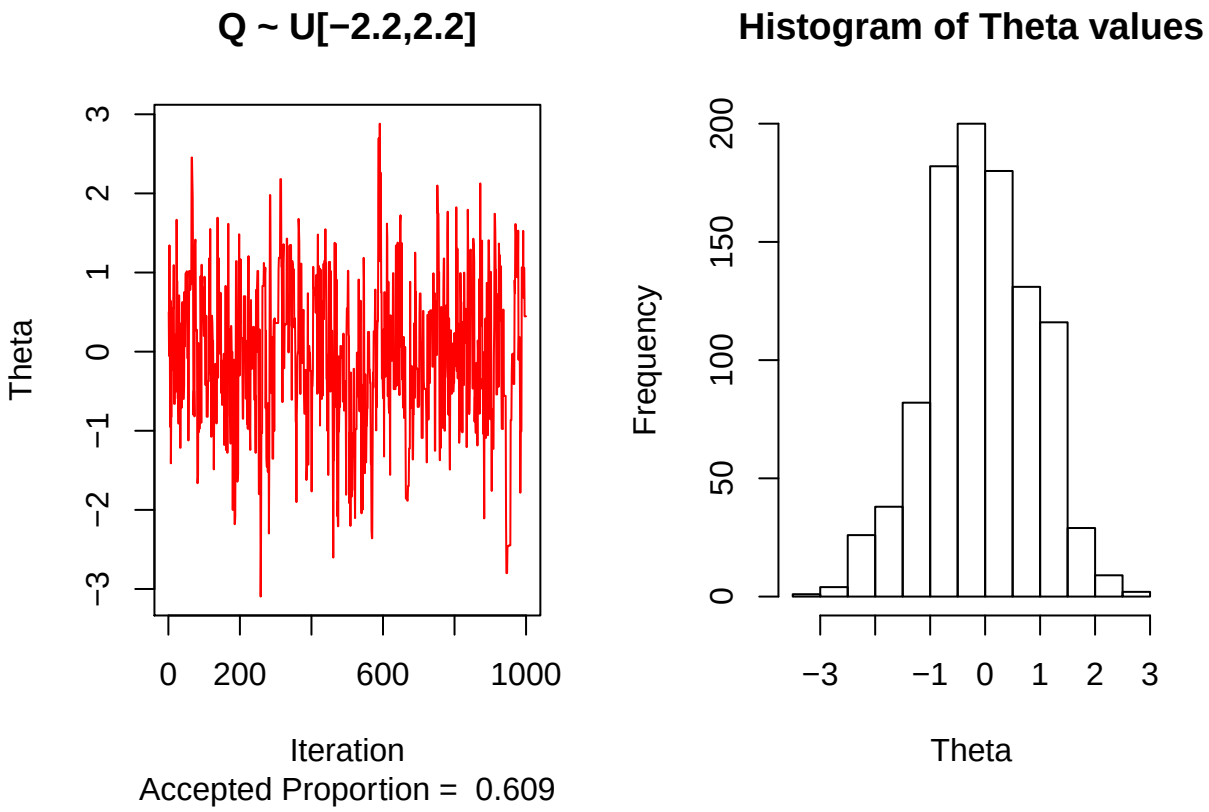


It appears that when $q \approx 2.3$, the acceptance rate is approximately 60%. We will attempt to find a more precise location within 1% error.

```
q <- seq(2,3,by = .01)
for (i in 1:length(q)) {
  df <- MH_Sampler_Normal(1000, 0.5, sample_density = runif, pmf = dunif,
                          min = -q[i], max = q[i])
  acceptance_rate <- df$proportion[1]
  if (abs(acceptance_rate - .60) < .01) cat("Acceptance is approximately 60% for q =", q[i], "\n")
}
```

```
## Acceptance is approximately 60% for q = 2.03
## Acceptance is approximately 60% for q = 2.05
## Acceptance is approximately 60% for q = 2.08
## Acceptance is approximately 60% for q = 2.09
## Acceptance is approximately 60% for q = 2.1
## Acceptance is approximately 60% for q = 2.12
## Acceptance is approximately 60% for q = 2.13
## Acceptance is approximately 60% for q = 2.14
## Acceptance is approximately 60% for q = 2.15
## Acceptance is approximately 60% for q = 2.16
## Acceptance is approximately 60% for q = 2.19
## Acceptance is approximately 60% for q = 2.22
## Acceptance is approximately 60% for q = 2.23
## Acceptance is approximately 60% for q = 2.26
## Acceptance is approximately 60% for q = 2.31
## Acceptance is approximately 60% for q = 2.46
```

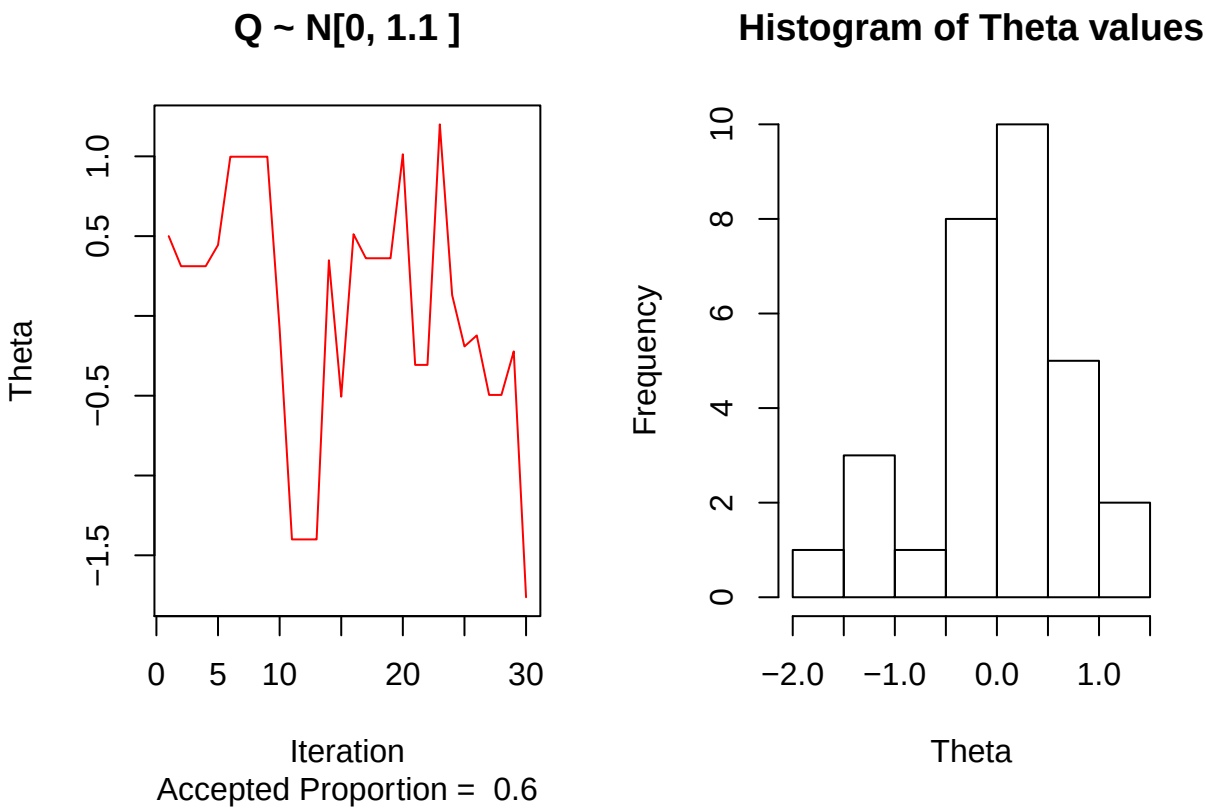
So from about 2 to 2.4, we can find that acceptance rate. Below is a plot where $q = 2.2$.



Of all the values for q so far, this one results in the most "standard normal" looking histogram.

(e): Now, we will let $Q \sim N(0, \sigma^2)$ and try to tune σ^2 to give an acceptance rate of approximately 60%. I'm going to write a brute force algorithm because I have the computational power to handle that.

```
sigmasqr <- seq(0, 5, by = .05)
for (i in 1:length(sigmasqr)) {
  df <- MH_Sampler_Normal(30, 0.5, sample_density = rnorm, pmf = dnorm,
                          mean = 0, sd = sigmasqr[i])
  if (abs(df$proportion[1] - .6) < 0.01) {
    par(mfrow = c(1,2))
    plot(seq(1:30),df$theta, type = "l", col = "red", xlab = "Iteration",
        main = paste("Q ~ N[0,", sigmasqr[i],"]"), sub = paste("Accepted Proportion = ",df$proportion[1]
        ylab = "Theta")
    hist(df$theta, main = "Histogram of Theta values", xlab = "Theta")
    break
  }
}
```



After several runs of the code above, it is found that the acceptance rate is approximately 0.6 when $\sigma^2 = 1$ or around that general neighborhood.

Exercise Three

This exercise is an introduction to the BRugs package in R. BRugs is an add-on package to R and you might need to download it (see https://www.r-project.org/). BRugs is the R interface for OpenBUGS, which is the open source version of WinBUGS (http://mathstat.helsinki.fi/openbugs/).

This exercise uses the beetles data which is below.

```
x[] n[] y[]
1.6907 59 6
1.7242 60 13
1.7552 62 18
1.7842 56 28
1.8113 63 52
1.8369 59 53
1.8610 62 61
1.8839 60 60
END
```

Solution

First, we load our the library of information and set the working directory.

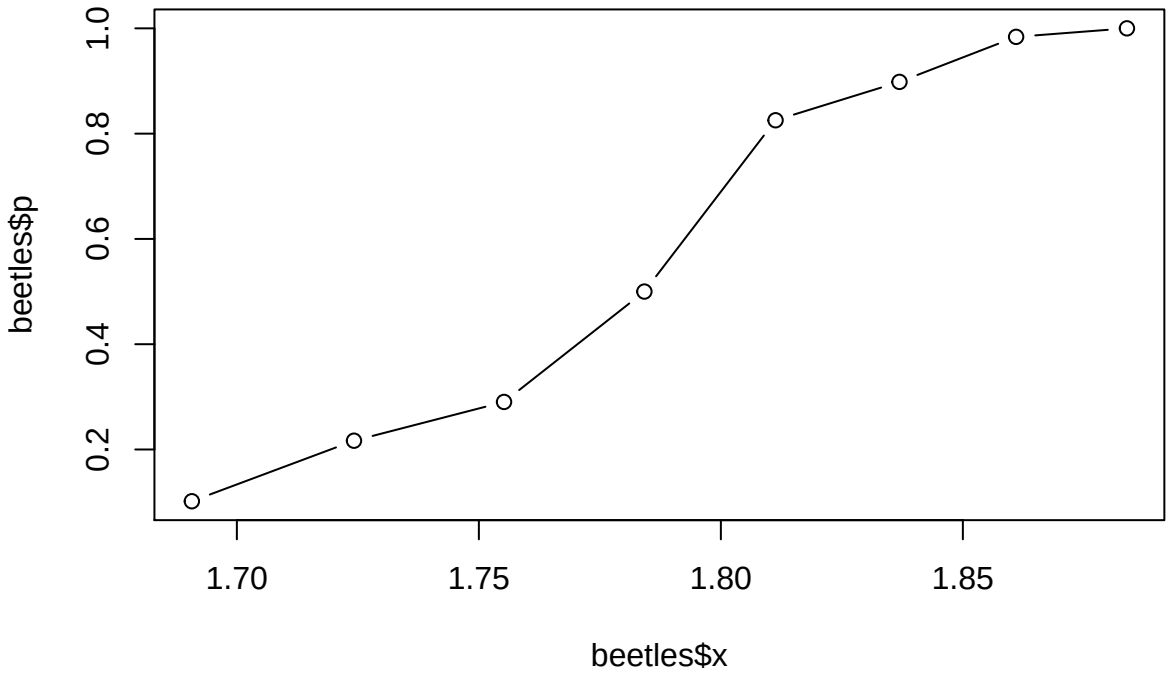
```
library("BRugs")
```

```
## Welcome to BRugs connected to OpenBUGS version 3.2.3
```

```
setwd("C:/Users/honeabear/Documents")
```

Next, we load the relevant data and then plot proportions.

```
beetles <- read.table("BeetlesData.txt", header = FALSE, nrows = 8, skip = 1, col.names = c("x", "n", "
beetles$p <- beetles$y/beetles$n
plot(beetles$x, beetles$p, type = "b")
```



Then we write code in OpenBUGS that specifies the model.

```
model
{   # likelihood
    for (i in 1:8) {
        y[i] ~ dbin(pi[i], n[i]); # Binomial
        pi[i] <- 1 - exp(-exp(pi.r[i])); # inverse link function
        pi.r[i] <- beta[1] + (beta[2] * x[i]); # Regression
        fitted[i] <- n[i] * pi[i]; # fitted values
    }
    # Priors
    beta[1] ~ dnorm(0,1.0E-6); # Intercept
    beta[2] ~ dnorm(0,1.0E-6); # Slope
}
```

Using the `BRugs` library, we can now run a script to obtain information about the data. Note that 'Initials.txt' contains the following information.

```
list(beta = c(0,0))
```

So we run thse script below.

```
# Checks the model file
modelCheck("BeetlesExtreme.odc")
```

```
## model is syntactically correct
```

```
# Reads the model data
modelData("BeetlesData.txt")
```

```
## data loaded
```

```
# Compiles the model with 1 chain
modelCompile(numChains = 1)
```

```
## model compiled
```

```
# Loads the initial values
modelInits(rep('Initials.txt', 1))
```

```
## Initializing chain 1:
```

```
## model is initialized
```

```
# burn-in
modelUpdate(5000)
```

```
## 5000 updates took 0 s
```

```
#monitor the intercept and slope
tosample <- c('beta')
samplesSet(tosample)
```

```
## monitor set for variable 'beta'
```

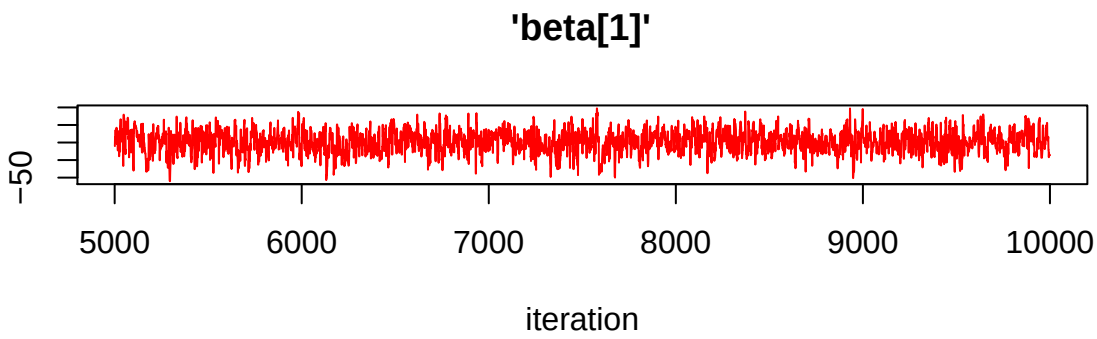
```
# sample
modelUpdate(5000)
```

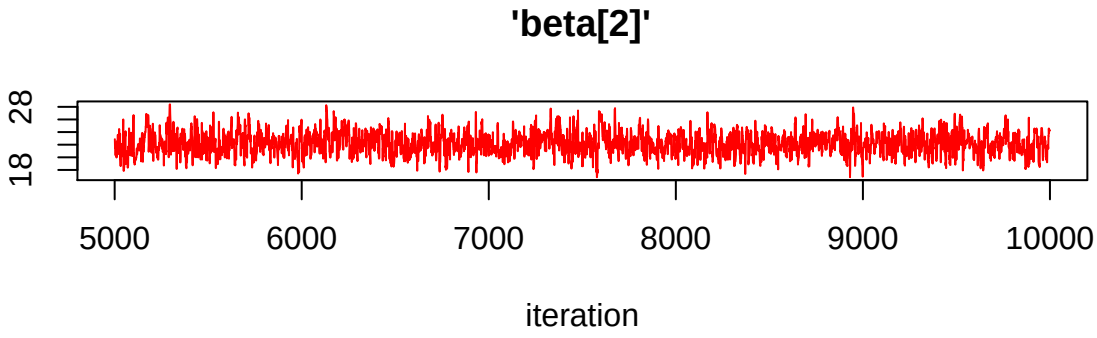
```
## 5000 updates took 0 s
```

```
#display the summary statistics
samplesStats(tosample)
```

```
##           mean    sd MC_error val2.5pc median val97.5pc start sample
## beta[1] -39.84 3.215  0.12240   -46.58 -39.76    -33.69  5001   5000
## beta[2]  22.19 1.786  0.06819    18.79  22.14     25.96  5001   5000
```

```
# plot the histories
par(mfrow = c(2,1))
plotHistory('beta[1]')
plotHistory('beta[2]')
```



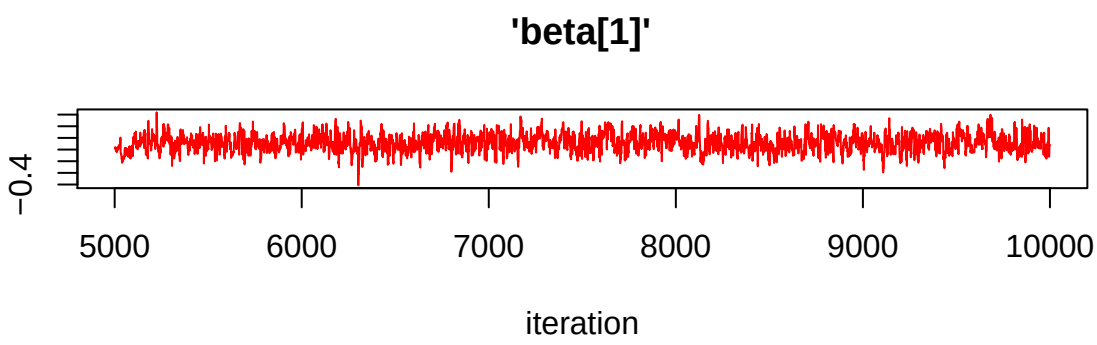


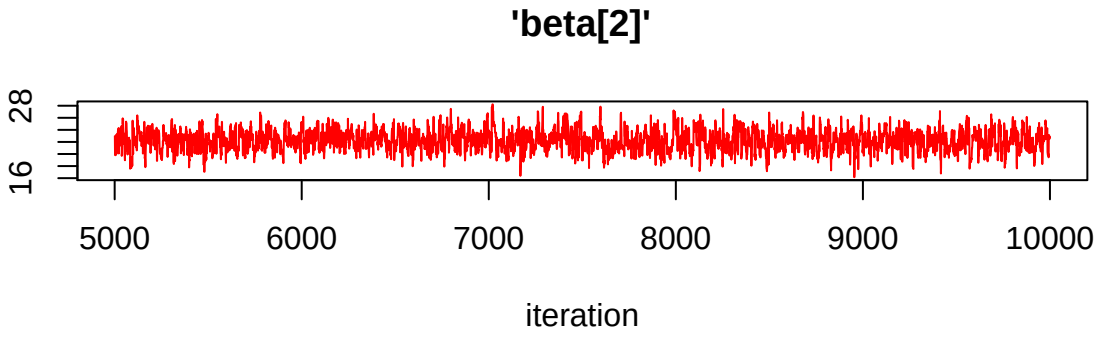
In order to improve the mixing of chains, we can change the regression line in the WinBUGS model to read

```
pi.r[i] <- beta[1] + (beta[2]*(x[i] - mean.x)); # Regression
```

and add the mean dose (mean.x) to a data file saved as BeetlesMean.txt.

```
## model is syntactically correct

## data loaded
## data loaded

## model compiled

## Initializing chain 1:

## model is initialized

## 5000 updates took 0 s

## monitor set for variable 'beta'

## 5000 updates took 0 s

##           mean      sd MC_error val2.5pc   median val97.5pc start sample
## beta[1] -0.043 0.07622 0.003035  -0.1909 -0.04379    0.1129  5001   5000
## beta[2] 22.200 1.78400 0.053740  18.7600 22.17000   25.7200  5001   5000
```





This addition has led to an improvement in the chains. β_1 and β_2 have significantly thinner intervals in the second versions of the regressions.

Finally, we are interested in what the deviance might look like as the model progresses, so we will eliminate the burn in part of the model and observe results.

```
# Checks the model file
modelCheck("BeetlesExtreme2.odc")
```

```
## model is syntactically correct
```

```
# Reads the model data
modelData("BeetlesData.txt")
```

```
## data loaded
```

```
modelData("BeetlesMean.txt")
```

```
## data loaded
```

```
# Compiles the model with 1 chain
modelCompile(numChains = 1)
```

```
## model compiled
```

```
# Loads the initial values
modelInits(rep('Initials.txt', 1))
```

```
## Initializing chain 1:
```

```
## model is initialized
```

```r
#monitor the intercept and slope
tosample <- c('beta', 'deviance')
samplesSet(tosample)
```

```
## monitor set for variable 'beta'
```

```
## monitor set for variable 'deviance'
```

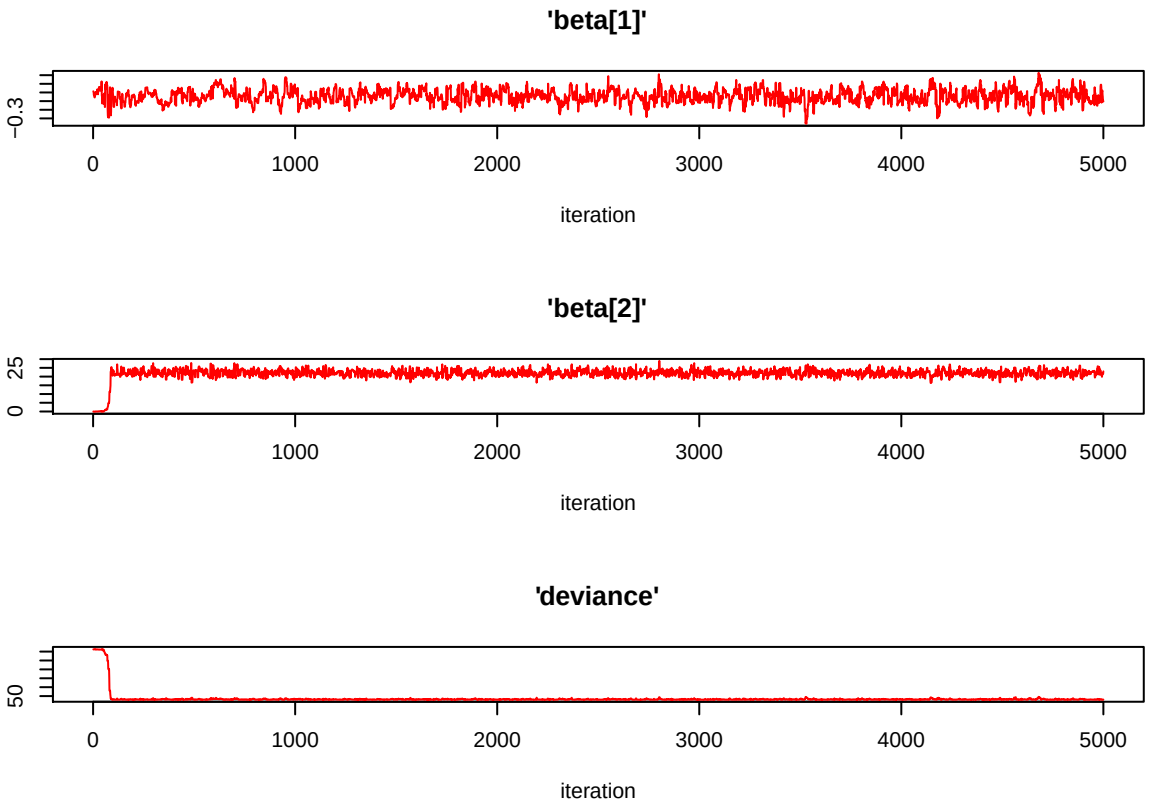
```r
# sample
modelUpdate(5000)
```

```
## 5000 updates took 0 s
```

```r
#display the summary statistics
samplesStats(tosample)
```

```
##              mean       sd MC_error val2.5pc   median val97.5pc start
## beta[1]  -0.04253  0.08037 0.003599   -0.201 -0.04322    0.1122     1
## beta[2]  21.73000  3.22300 0.313500   18.080 21.98000   25.7400     1
## deviance 35.83000 33.33000 3.867000   29.690 30.97000   40.1800     1
##          sample
## beta[1]    5000
## beta[2]    5000
## deviance   5000
```

```r
# plot the histories
par(mfrow = c(3,1))
plotHistory('beta[1]')
plotHistory('beta[2]')
plotHistory('deviance')
```



Now, we can see how quickly the model actually converges compared to just viewing it after convergence. The deviance drops very quickly (around the 50th) iteration.