Lab 4: Metropolis-Hastings

Harry Bendekgey and Peter Brody-Moore
Math 153: Bayesian Statistics

April 5, 2018

Problem 1: Non-Bayesian/Educational

a) In problem 1, we propose two algorithms that estimate the value of π. The first algorithm is a random walk markov chain that always moves. If the random walk proposes a move outside of the stationary support, then it redraws a new point until it gets one within the support. The problem here is that you'll never stay in the outer edges of a square, even though there is a low probability of getting back there. As a result, you will spend too much time in the interior and overestimate pi, which we see in the results of the chain.

```
# The code to run a naive metropolis chain
naive.metropolis <- function(w0, epsilon, n) {
  locs <- matrix(w0, ncol=2)
  for(i in 1:n-1){
    x <- locs[nrow(locs),]
    newX <- runif(1,x[1]-epsilon,x[1]+epsilon)
    newY <- runif(1,x[2]-epsilon,x[2]+epsilon)
    while(newX < -1 || newX > 1|| newY < -1 || newY > 1){
      newX <- runif(1,x[1]-epsilon,x[1]+epsilon)
      newY <- runif(1,x[2]-epsilon,x[2]+epsilon)
    }
    locs <- rbind(locs,c(newX,newY))
  }
  rownames(locs) <- c()
  return(locs)
}
```

```
estimate.pi(naive.metropolis(c(0,0), .5, 10000))

## [1] 3.471
```

This estimate for pi using the naive metropolis chain is too high because the chain spent too much time in the interior of the space. Lets see if we can do better.

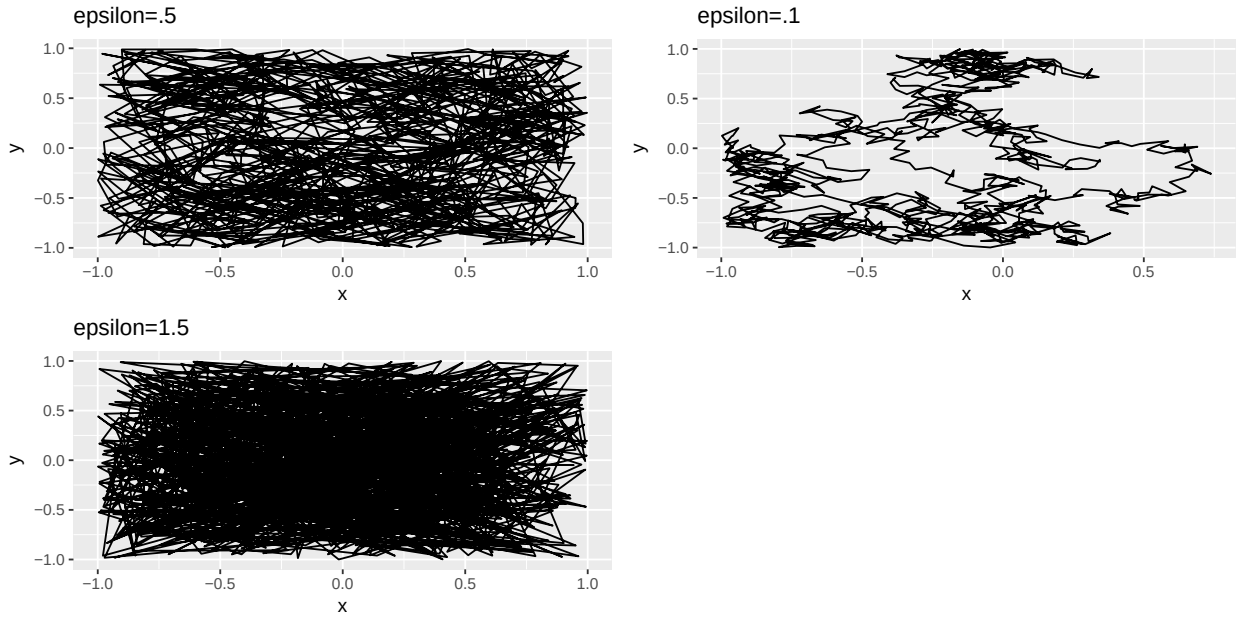
```
# The code to run a better metropolis chain
better.metropolis <- function(w0, epsilon, n) {
  locs <- matrix(w0, ncol=2)
  for(i in 1:n){
    x <- locs[nrow(locs),]
    newX <- runif(1,x[1]-epsilon,x[1]+epsilon)
    newY <- runif(1,x[2]-epsilon,x[2]+epsilon)
```

```
    if (newX >= -1 && newX <= 1 && newY >= -1 && newY <= 1) {
      locs <- rbind(locs,c(newX,newY))
    } else {
      locs <- rbind(locs, x)
    }
  }
  rownames(locs) <- c()
  return(locs)
}
```

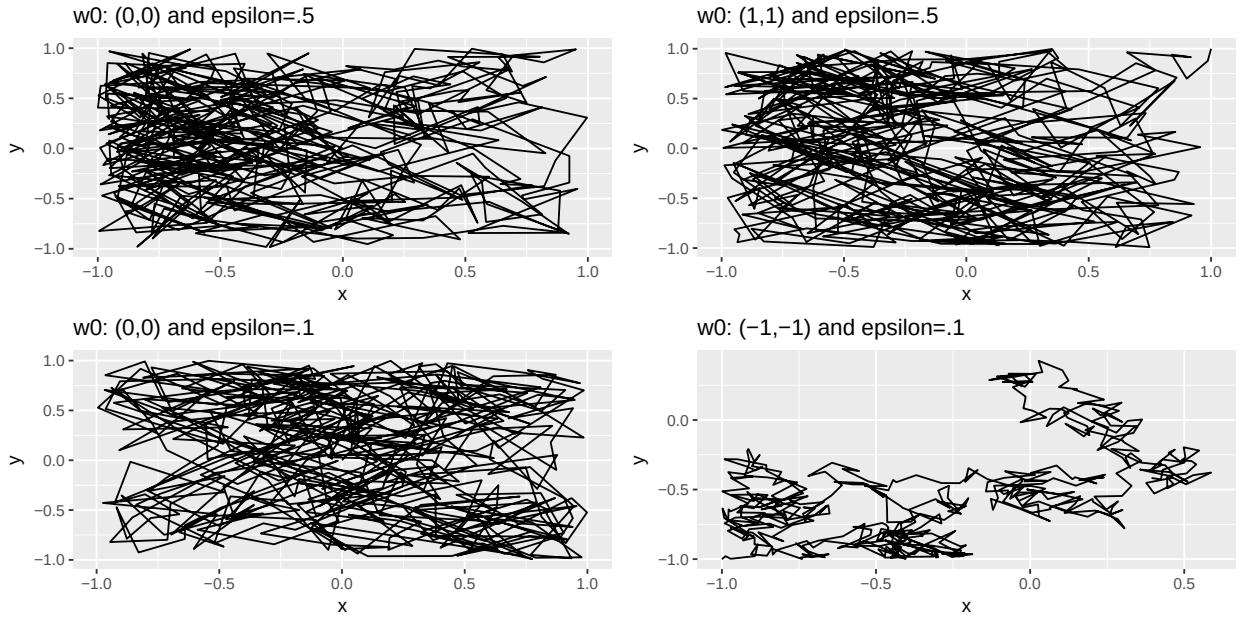
```
estimate.pi(better.metropolis(c(0,0), .5, 10000))

## [1] 3.132
```

The estimate using the better metropolis chain is much closer to the true value. Now let's visualize how changes in ϵ, w_0 and n change the movement of the chain and its estimate of π.

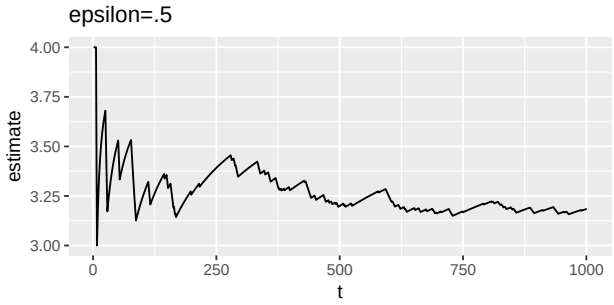
```
# Comparing movement of the better chain
graphBetterMovement <- function(w0,epsilon,n){
  naiveData <- data.frame(naive.metropolis(w0, epsilon, n))
  plot <- ggplot(naiveData,aes(x=X1,y=X2)) + geom_path() + xlab('x') + ylab('y')
  return(plot)
}

ggarrange(graphBetterMovement(c(0,0),.5,1000)+ggtitle('epsilon=.5'),
          graphBetterMovement(c(0,0),.1,1000)+ggtitle('epsilon=.1'),
          graphBetterMovement(c(0,0),1.5,1000)+ggtitle('epsilon=1.5'))
```

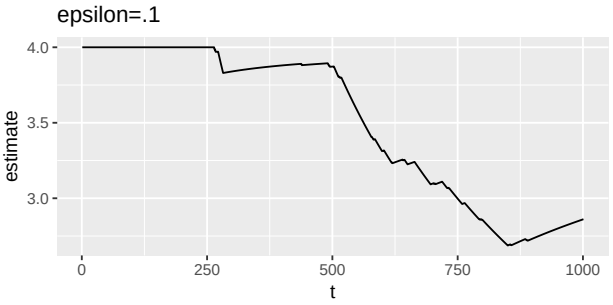


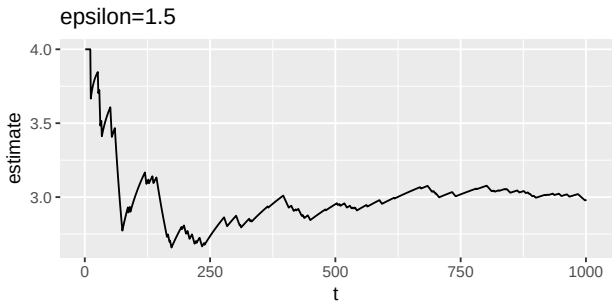
```
ggarrange(graphBetterMovement(c(0,0),.5,500)+ggtitle('w0: (0,0) and epsilon=.5'),
          graphBetterMovement(c(1,1),.5,500)+ggtitle('w0: (1,1) and epsilon=.5'),
          graphBetterMovement(c(0,0),.5,500)+ggtitle('w0: (0,0) and epsilon=.1'),
          graphBetterMovement(c(-1,-1),.1,500)+ggtitle('w0: (-1,-1) and epsilon=.1'))
```



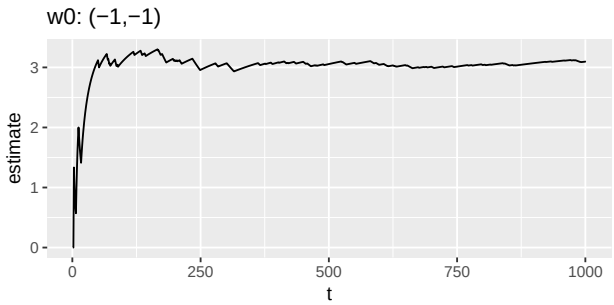

```
#Comparing Estimates of the better chain
graphBetterEstimates <- function(w0,epsilon,n){
  estimate <- c()
  t <- c()
  data <- data.frame(better.metropolis(w0,epsilon,n))
  for(i in 1:nrow(data)) {
    subset <- data[1:i,]
    estimate[i] <- estimate.pi(subset)
    t[i] <- i
  }
  piEstimates <- data.frame(t,estimate)
  return(ggplot(piEstimates,aes(x=t,y=estimate)) + geom_line())
}
ggarrange(graphBetterEstimates(c(0,0),.5,1000)+ggtitle('epsilon=.5'),
          graphBetterEstimates(c(0,0),.1,1000)+ggtitle('epsilon=.1'),
          graphBetterEstimates(c(0,0),1.5,1000)+ggtitle('epsilon=1.5'))
```

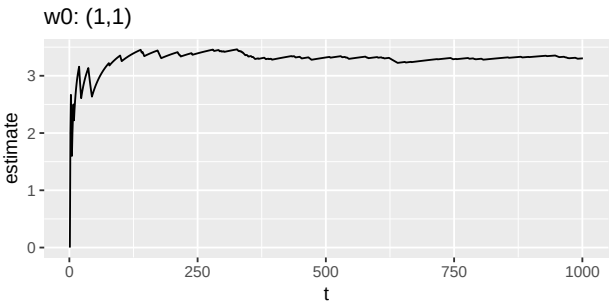


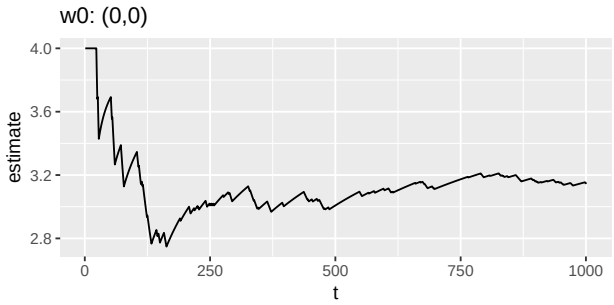


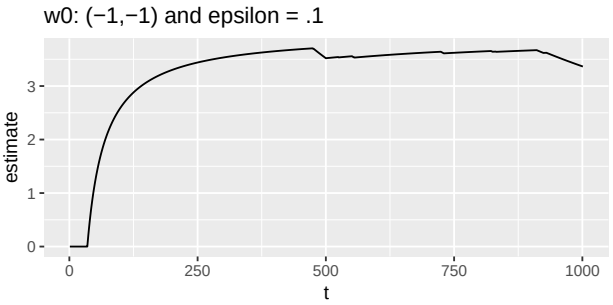



```
ggarrange(graphBetterEstimates(c(-1,-1),.5,1000)+ggtitle('w0: (-1,-1)'),
          graphBetterEstimates(c(1,1),.5,1000)+ggtitle('w0: (1,1)'),
          graphBetterEstimates(c(0,0),.5,1000)+ggtitle('w0: (0,0)'),
          graphBetterEstimates(c(-1,-1),.1,1000)+ggtitle('w0: (-1,-1) and epsilon = .1'))
```









(i) The choice of ϵ can absolutely affect the behavior of the chain. As we can see in the first group of plots, for $\epsilon = .1$ (small epsilon), the chain fails to explore the entire space because it's step size hinders its movement. Although this may work itself out for very large values of n, this adds uncessary computation. For $\epsilon = 1.5$ (large epsilon), the chain will stay where it is very often because the large step size will push it out of the boundaries often. Again, this is inefficient computation and affects its ability to explore the full space. Therefore, $\epsilon = .5$ is near the sweet spot where it will fully explore the space, but won't waste

computation staying the same spot too often. This can be seen in the estimation of π vs t, as $\epsilon = .5$ converges to the true value quicker while $\epsilon = .1$ and $\epsilon = 1.5$ bounce around much more.
(ii) The choice of w_0 also affects the behavior of the change. Starting in the corner of the space like (1,1) or (-1,-1) can bias where the chain explores. This is especially problematic for low ϵ because it will tend to stay around where it starts and small n where it doesn't have time to explore the full space with a small ϵ.
(iii) Finally the length of the run affects our ability to estimate pi. Even if there are problems with a low or high ϵ or a corner w_0, if you run the chain long enough it will probably return a good estimate. This is shown in our estimate plots where even after 1000 steps, our estimate starts to concern. However, there are concerns with runtime, especially as our chains get more complex and computionally intensive.
d) Burn-in is the idea that you should run your chain for a certain number of steps before you start collecting data. This is important because it allows the chain to stabilize to the stationary distribution and find a good starting point. This is especially important if you have a bad starting point because it allows the chain to start in a good location before you collect points. However, choosing your burn-in is an inexact art, and running your chain for a long time can achieve similar benefits.

Problem 2: Bayesian but Unncessary

We want to use Metropolis-Hastings to estimate something we already know a lot about: a beta posterior. For our candidate distribution, we will use our beta prior, making this an independence chain.
We note that the Metropolis-Hastings ratio is given by

$$R(x, y) = \min(1, \frac{f(y)g(x)}{f(x)g(y)})$$

Where g is the prior density and f is the posterior density. Let's examine the second term. If we expand the two posteriors using Bayes' rule, the marginals will cancel, giving us:

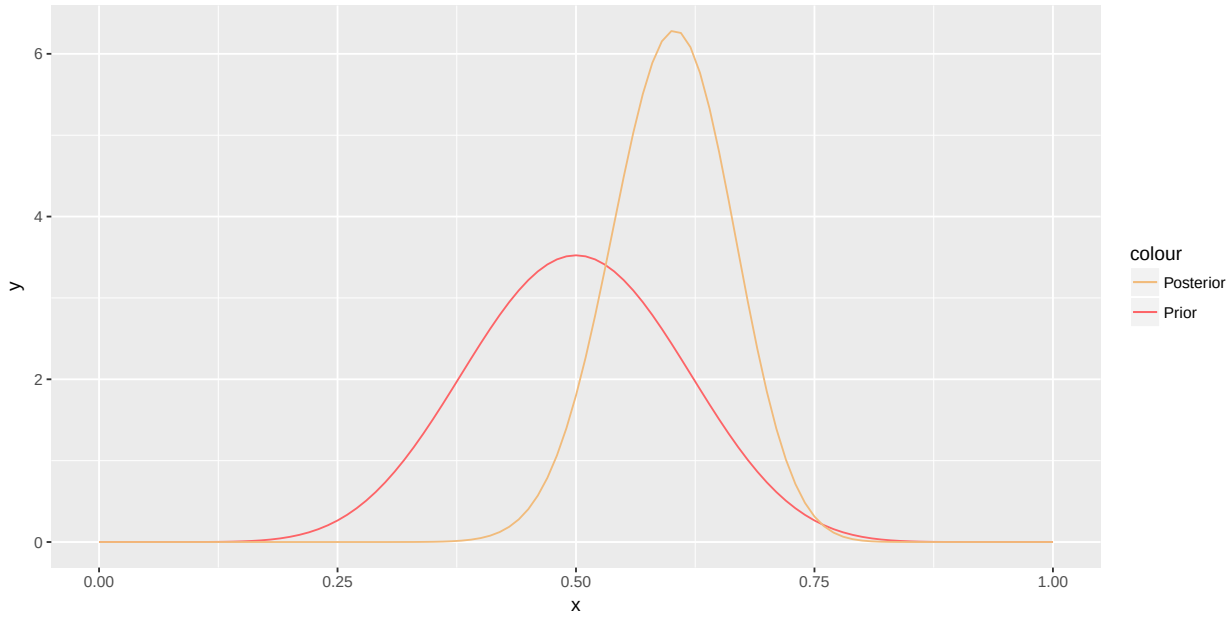
$$\frac{L(d|y)g(y)g(x)}{L(d|x)g(x)g(y)}$$

We notice that the priors cancel and we get the ratio of the likelihoods. Thus we get

$$R(x, y) = \min(1, \frac{L(d|y)}{L(d|x)}$$

Let's see if sampling using this method gives us something close to the Beta posterior we want. First, we consider using a smart prior: let the prior be Beta(10,10) and the data is drawn from a binomial experiment with $n = 40, p = 0.5$. First let's make the draw and visualize the target and candidate distributions:

```
require(wesanderson)
set.seed(47)
data <- rbinom(1,40,.5)
a <- data + 10 #posterior parameters
b <- 10 + 40 - data
ggplot(data.frame(x = c(0, 1)), aes(x)) +
    stat_function(fun=function(x) dbeta(x,10,10), aes(col="Prior")) +
    stat_function(fun=function(x) dbeta(x,a,b), aes(col="Posterior")) +
    scale_color_manual(values=wes_palette(n=2, name="GrandBudapest"))
```

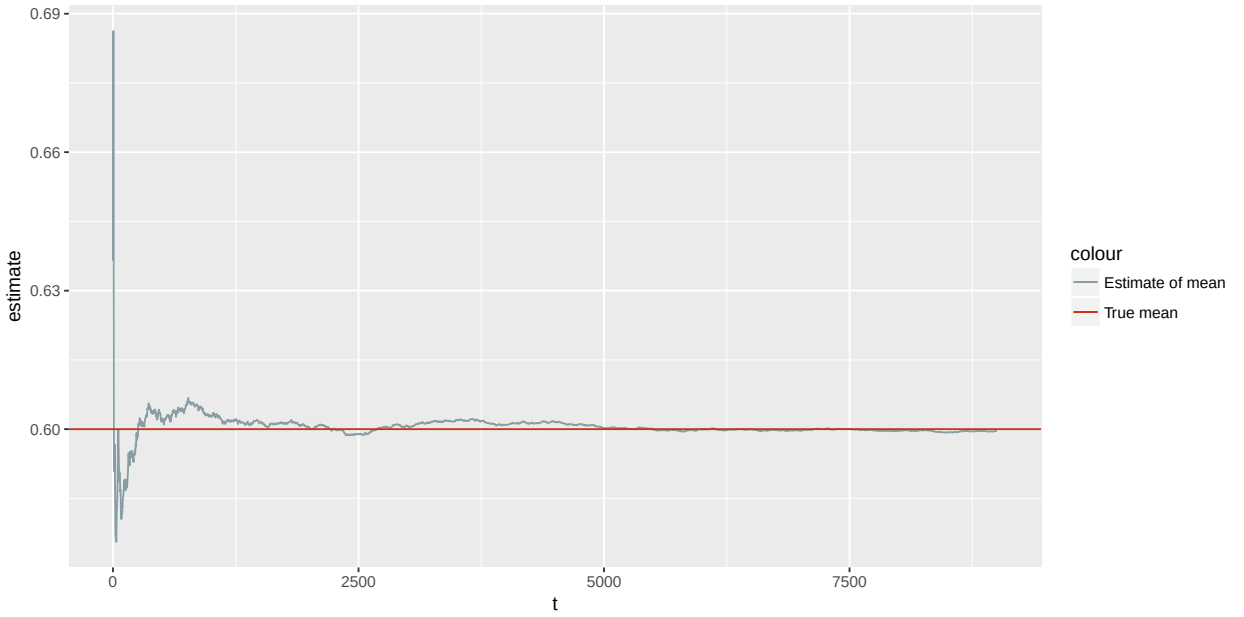


Now, we simulate:

```
samples <- metropolis.Bayes(10000,10,10,data,40)
compare.results(samples,a,b,data,40)

## $estimate.mean
## [1] 0.5996
##
## $true.mean
## [1] 0.6
##
## $estimate.quantiles
##     5%    50%    95%
## 0.4942 0.5998 0.7053
##
## $true.quantiles
## [1] 0.4948 0.6011 0.7013
```

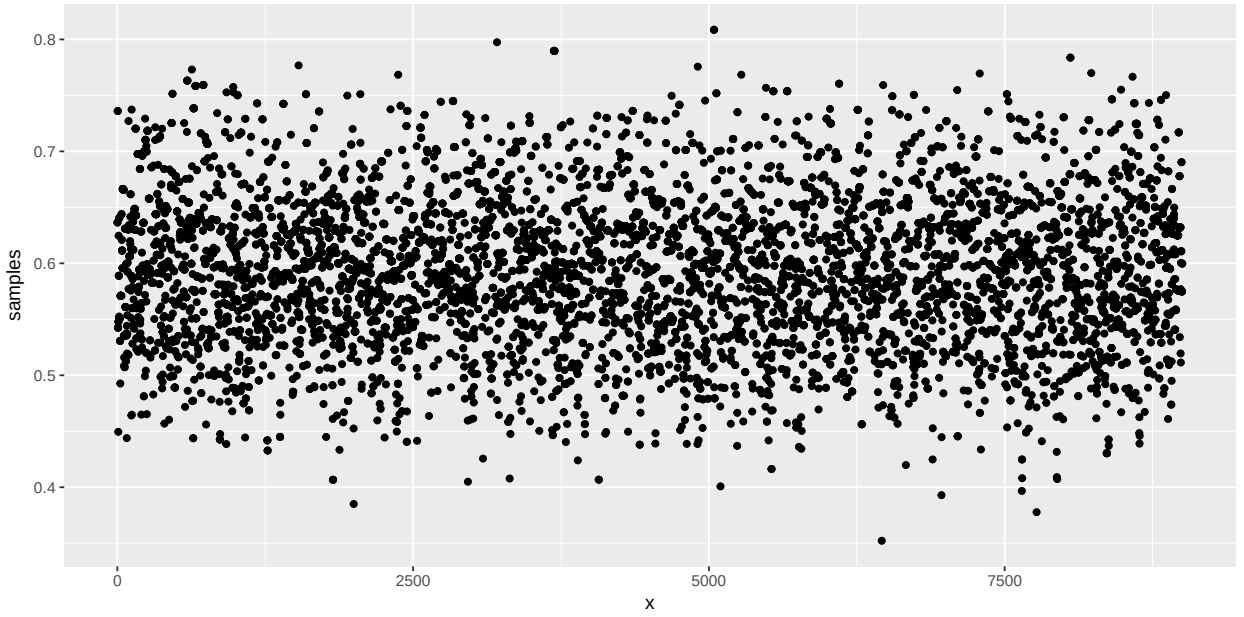
These are pretty good estimates of the mean, median, and 90% credible interval.

```
graph.estimates(samples,a,b) +
  scale_color_manual(values=wes_palette(n=2, name="Royal1"))
```



Here, we can see the estimate for the mean of the distribution converging over time to the true answer. This looks good. This seems to be a quick-moving chain, but we can confirm this by looking at movement over time:

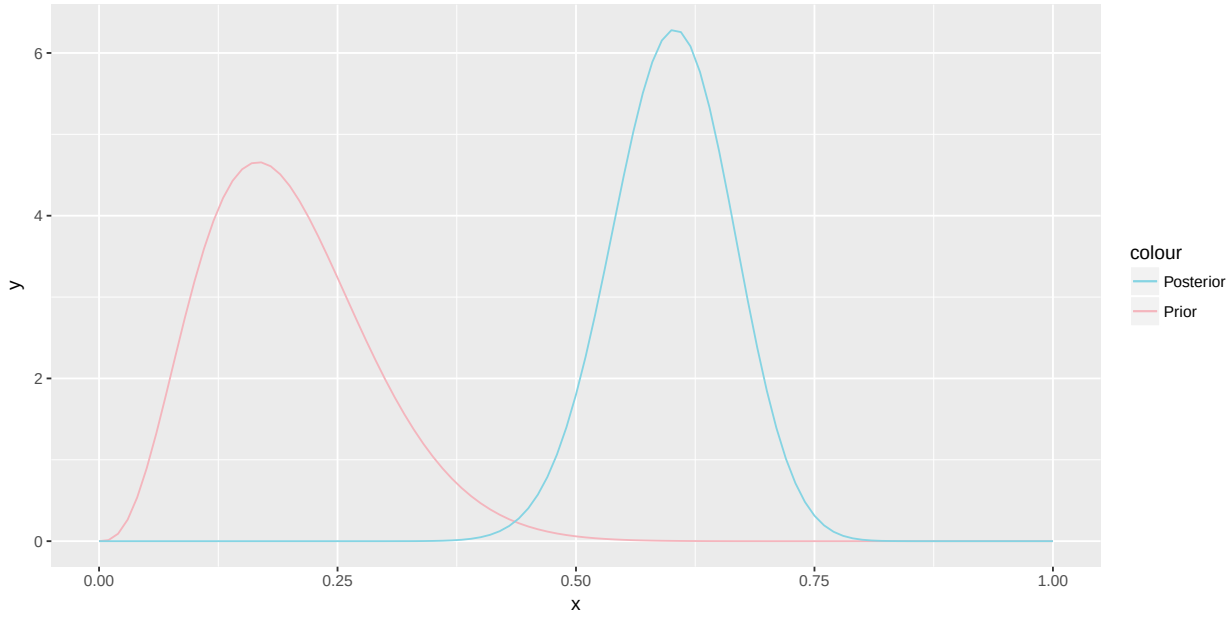
```
data.frame(x=c(1:length(samples)),samples=samples) %>%
  ggplot(aes(x,samples)) + geom_point()
```



Now let's try with a bad prior. If we use a prior of Beta(4,16), but the true parameter value is 0.8, we see:

```
data <- rbinom(1,40,.8)
a <- data + 4 #posterior parameters
```

```
b <- 16 + 40 - data
ggplot(data.frame(x = c(0, 1)), aes(x)) +
    stat_function(fun=function(x) dbeta(x,4,16), aes(col="Prior")) +
    stat_function(fun=function(x) dbeta(x,a,b), aes(col="Posterior")) +
    scale_color_manual(values=wes_palette(n=2, name="Moonrise3"))
```

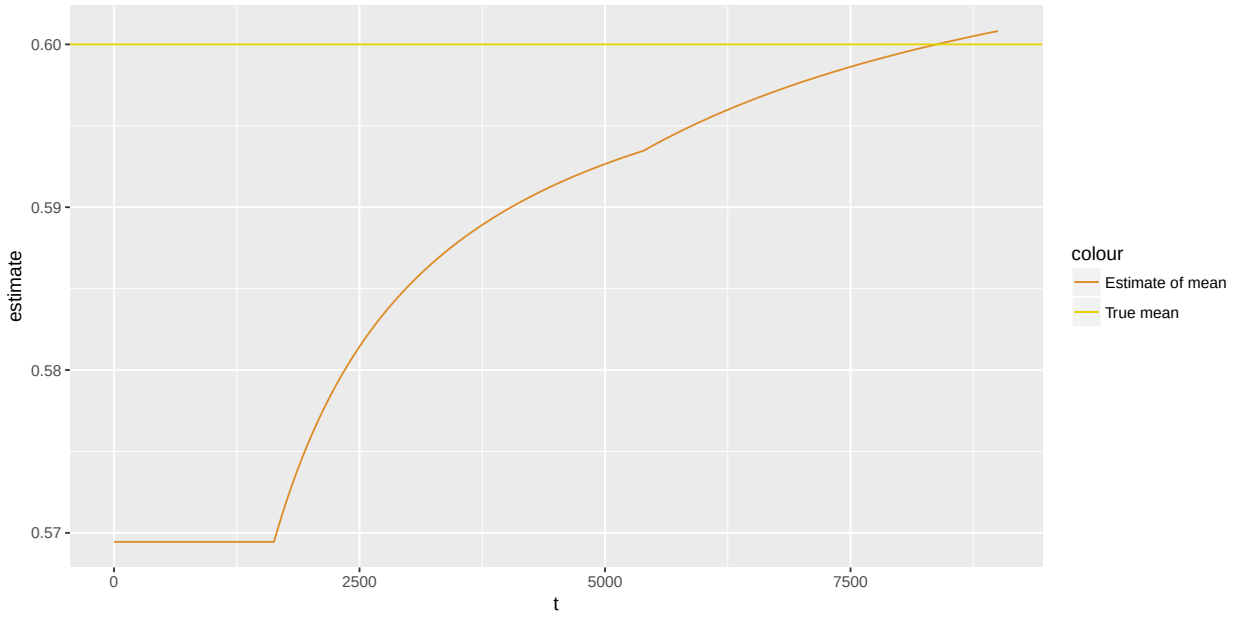


This is pretty bad candidate for that target. Let's see what happens:

```
samples <- metropolis.Bayes(10000,4,16,data,40)
compare.results(samples,a,b,data,40)

## $estimate.mean
## [1] 0.6008
##
## $true.mean
## [1] 0.6
##
## $estimate.quantiles
##     5%    50%    95%
## 0.5694 0.6039 0.6118
##
## $true.quantiles
## [1] 0.4948 0.6011 0.7013
```

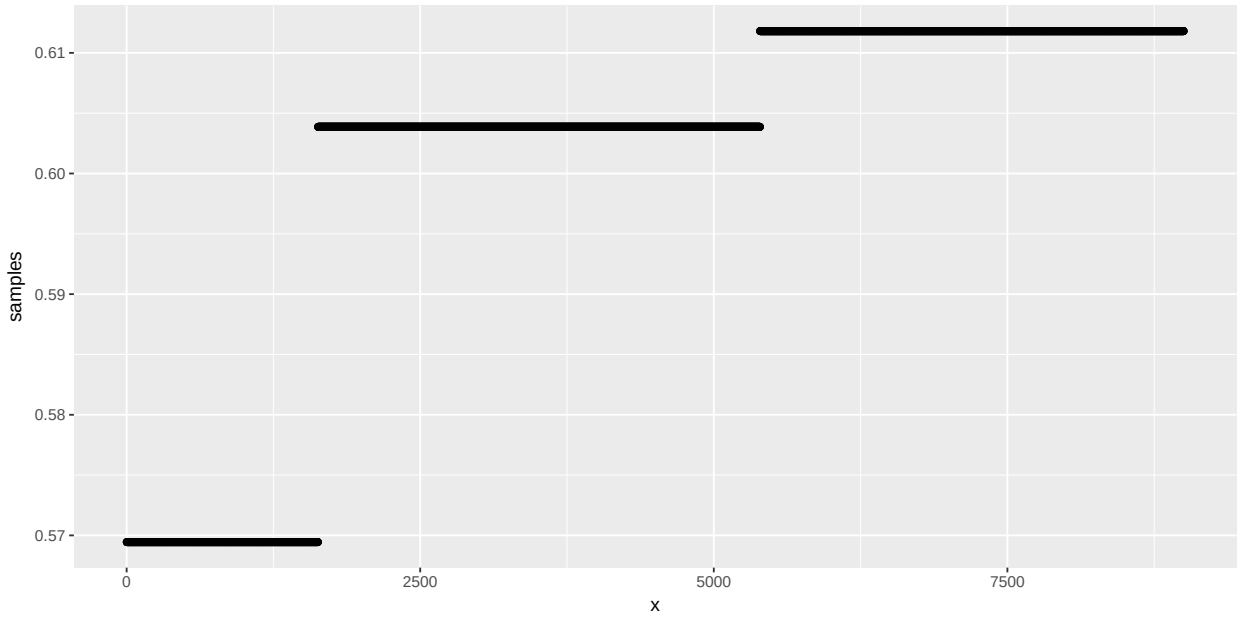
The mean estimate isn't bad, but then we notice that the estimate for the 95th percentile is roughly the same as the mean. Let's take a look:

```
graph.estimates(samples,a,b) +
  scale_color_manual(values=wes_palette(n=2, name="FantasticFox"))
```



Weirdly at $n = 10000$ we're not too far off on our estimate, but the rigidity of this graph tells us that there isn't much movement through the chain. If we look at the points:

```
df <- data.frame(x=c(1:length(samples)),samples=samples)
ggplot(df,aes(x,samples)) + geom_point()
```



Dear lord. It got that estimate by only taking on 3 values the entire time. The three quantiles printed above are the only three values obtained by the chain. This was an exceptional case of terrible Metropolis-Hastings-ing