Investigating the effect of proposal distribution on Metropolis-Hastings sampling performance

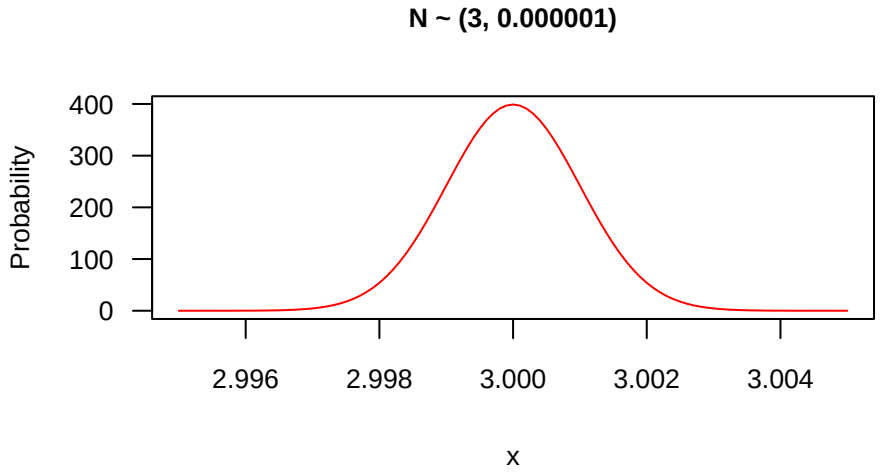
Dipika Gawande | 12 November 2021

Let's investigate how the performance of the Metropolis method can be affected by poor choices of the proposal distribution. We want to investigate the effect of changing the scale of the random walk type moves that we are proposing.

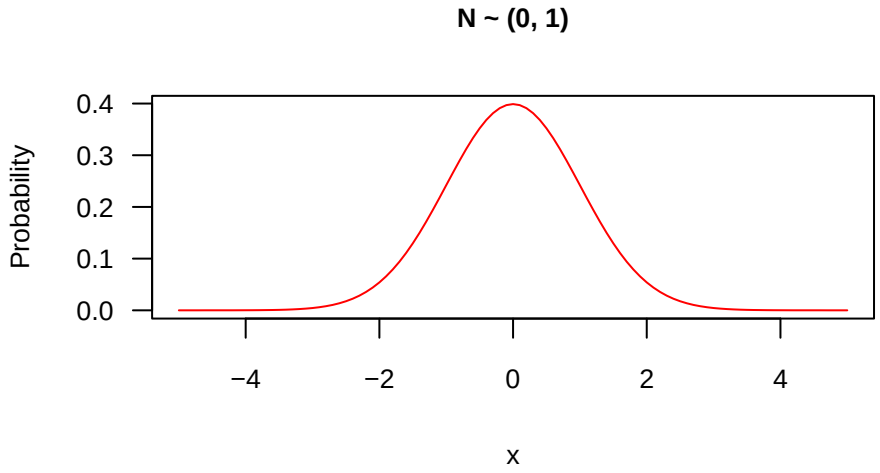
As one simple way to do this, in the existing script, find the place where we used the arbitrary choice of "1" in the command that generates a random candidate: `candidate <- runif(n=1, min = state-1 , max = state+1)` Change those 1 's to a variable; let's call it scale. So now we can influence the step size by changing the value of scale.

```
f1 <- function(x){
  (1/(0.001*sqrt(2*pi))) * exp(-0.5*((x-3)/0.001)^2)
}

#Standard Normal, for reference:
fnorm <- function(x){
  (1/sqrt(2*pi)) * exp(-0.5*(x^2))
}
```

```
plot(f1, 2.995, 3.005,
     las = 1, col = "red",
     main = "N ~ (3, 0.000001)",
     ylab = "Probability",
     cex.main = 0.8, cex.axis = 0.8, cex.lab = 0.8)
```



```r
#Standard Normal, for reference:
plot(fnorm, -5, 5,
     las = 1, col = "red",
     main = "N ~ (0, 1)",
     ylab = "Probability",
     cex.main = 0.8, cex.axis = 0.8, cex.lab = 0.8)
```



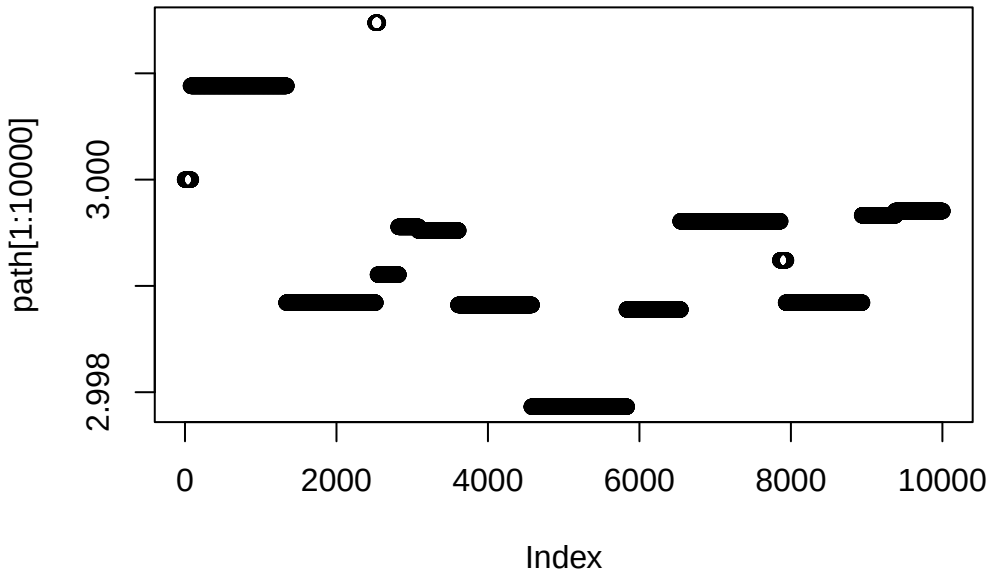
(Part a)

For now let scale keep the value 1, but suppose you want to use the Metropolis method to simulate from the Normal distribution with mean 3 and SD 0.001. You just need to modify the definition of the function f to specify that Normal density. Taking the initial state as 3 (to give the simulation the best chance to succeed), do a run of 10,000 iterations and make some plots to look at your results. Comment on the performance of the method. Why did this happen?

```r
set.seed(1)
f <- f1

nit <- 100000
path <- rep(0, nit)

state <- 3  # initial state
path[1] <- state

for(i in 2:nit){
  candidate <- runif(n = 1, min = state-1, max = state+1)
  ratio <- f(candidate) / f(state)
  u <- runif(n = 1, min = 0, max = 1)
  if(u < ratio){state <- candidate}
  path[i] <- state
}
#path[1:100]
```

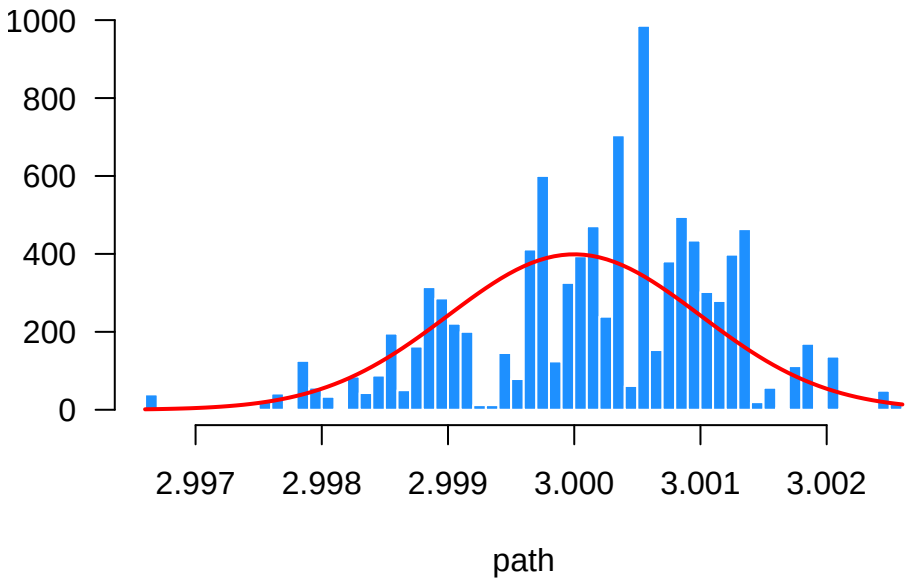
- **Plot of the sample path**

```
plot(path[1:10000])
```



- **Histogram of the density**

```
library(MASS) #for truehist
truehist(path,
         col = "dodgerblue", border = "white", las = 1)
curve(f1, add=T, col="red", lwd=2)
```



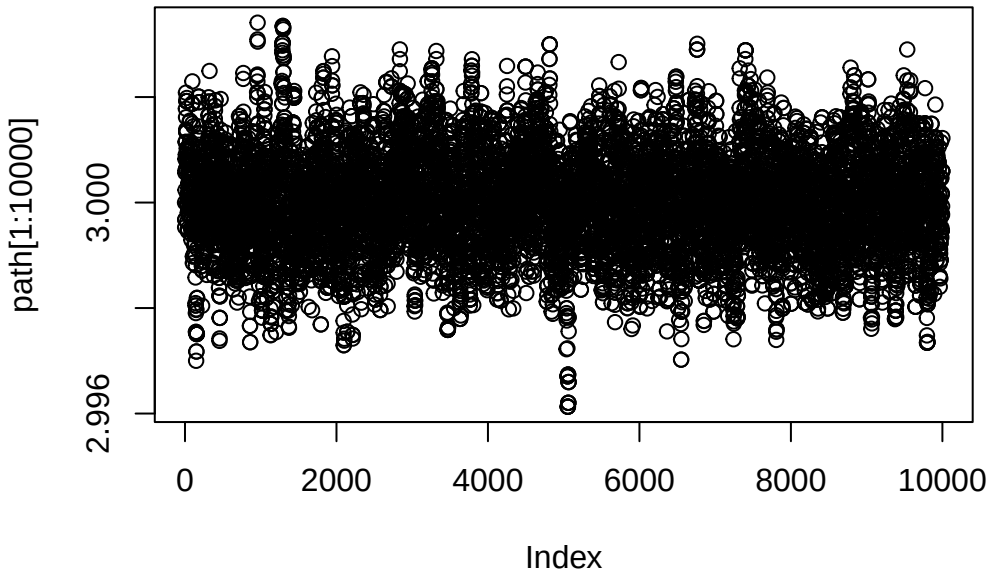
Ans: The method performed badly. A scale of 1 gave way too wide of a proposal distribution, and so the proposed y's could range from 2 to 4. Meanwhile the probability mass under our desired density drops to near zero outside of the range [2.997, 3.002]. The chain would need much much much more time to be able to traverse the state space and spend the appropriate amount of time at each state, if the range of proposed y values is so wide.

(Part b)

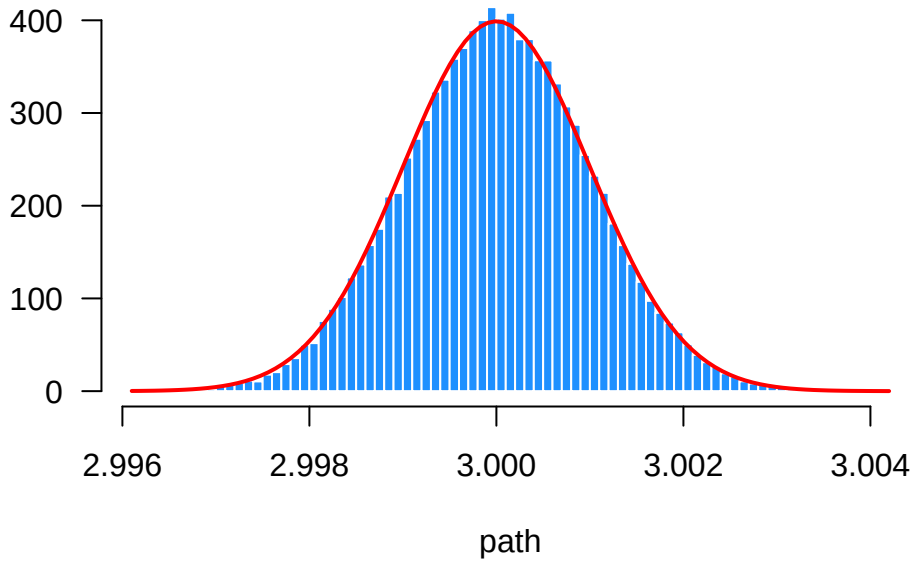
Repeat (Part a), except now take scale to be 0.001.

```
set.seed(1)
f <- f1

nit <- 100000
path <- rep(0, nit)

state <- 3  # initial state
path[1] <- state

for(i in 2:nit){
  candidate <- runif(n = 1, min = state-0.001, max = state+0.001)
  ratio <- f(candidate) / f(state)
  u <- runif(n = 1, min = 0, max = 1)
  if(u < ratio){state <- candidate}
  path[i] <- state
}
#path[1:100]
```

```
plot(path[1:10000])
```




```
truehist(path,
         col = "dodgerblue", border = "white", las = 1)
curve(f1, add=T, col="red", lwd=2)
```



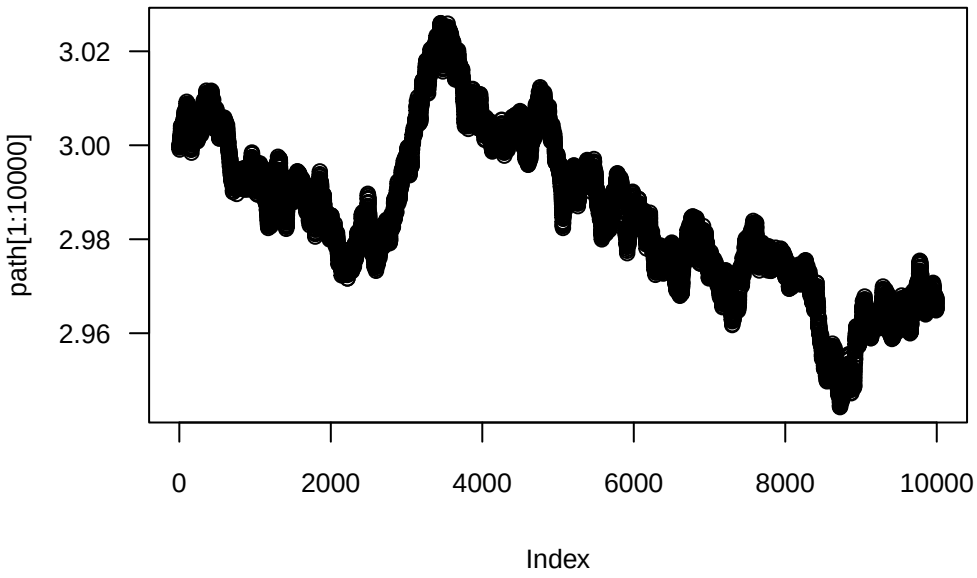
Ans: This is much better. When the proposed y's come from a reasonable distribution, the chain can spend the appropriate amount of time at each state within 100,000 steps.

(Part c)

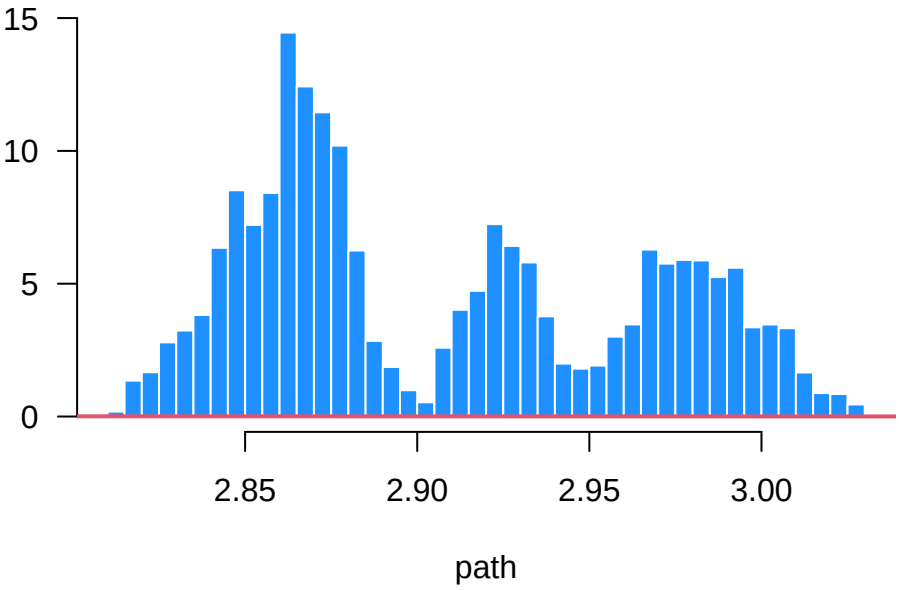
Repeat (Part a) so scale is still 0.001 and your initial state is still 3, except now take f to be the standard Normal density, with mean 0 and SD = 1.

```
set.seed(1)
f <- fnorm

nit <- 100000
path <- rep(0, nit)

state <- 3  # initial state
path[1] <- state

for(i in 2:nit){
  candidate <- runif(n = 1, min = state-0.001, max = state+0.001)
  ratio <- f(candidate) / f(state)
  u <- runif(n = 1, min = 0, max = 1)
  if(u < ratio){state <- candidate}
  path[i] <- state
}
#path[1:100]
```

```
plot(path[1:10000],
    las = 1, cex.main = 0.8, cex.axis = 0.8, cex.lab = 0.8)
```




```
truehist(path,
         col = "dodgerblue", border = "white", las = 1)
#con <- integrate(fnorm, -1000, 1000)$value
#fpset_normalized <- function(x){fnorm(x)/con}
plot(fnorm, -5, 5, n=1000, add = T, lwd=2, col=2)
```



The method does poorly again because the proposal distribution is too narrow! It's centered at 3, which is far in the right tail of our target distribution (the standard normal), but that alone wouldn't lead to such poor performance. Likely the biggest issue is that the proposal distribution only spans 0.001 on either side of the current state. It's the opposite problem from part (a) – now the proposed y's have almost no variation in them. The chain doesn't have enough time to traverse the full state space properly, because every new proposed y is so close to where we started (3 standard deviations out from the center of the target distribution) that the chain's movement is restricted to this small range within a few decimal places on either side of 3.