CDT Module 1 - R Review

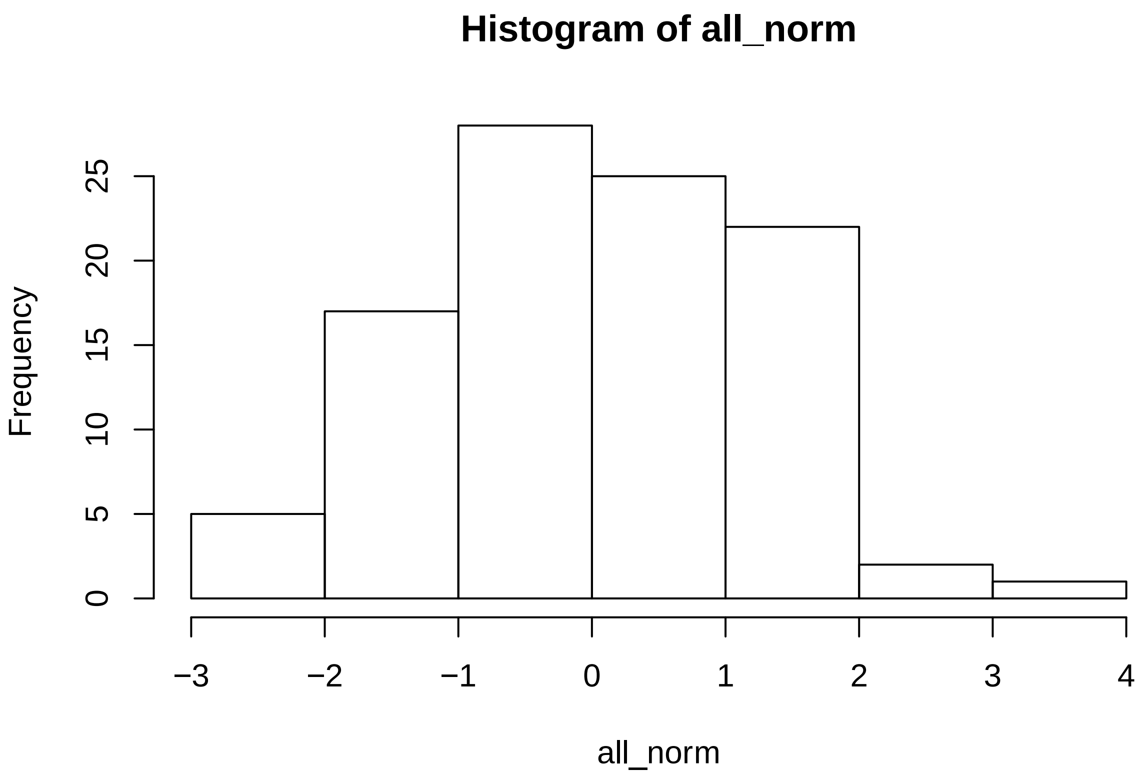
Valerie Bradley

9/19/2018

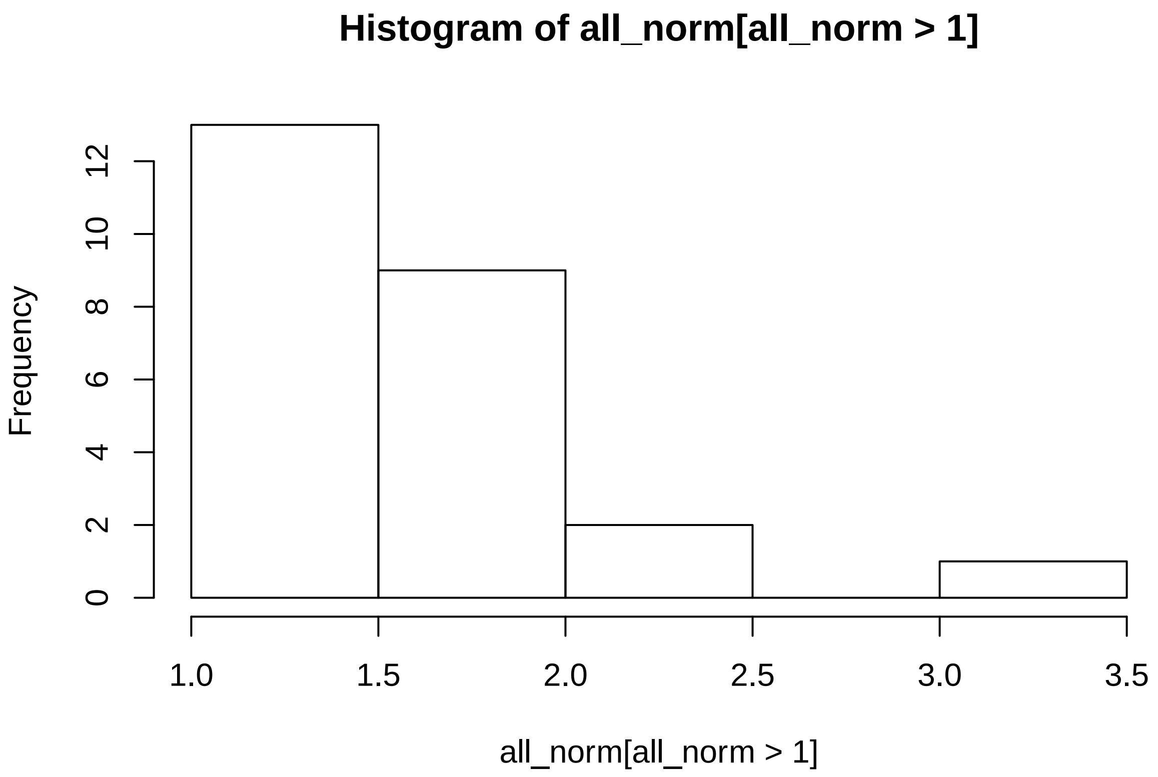
1. Vectors

a. Generate 100 standard normal random variables, and keep only the ones which are greater than 1. Don't use a loop!

```
all_norm = rnorm(100)
hist(all_norm)
```

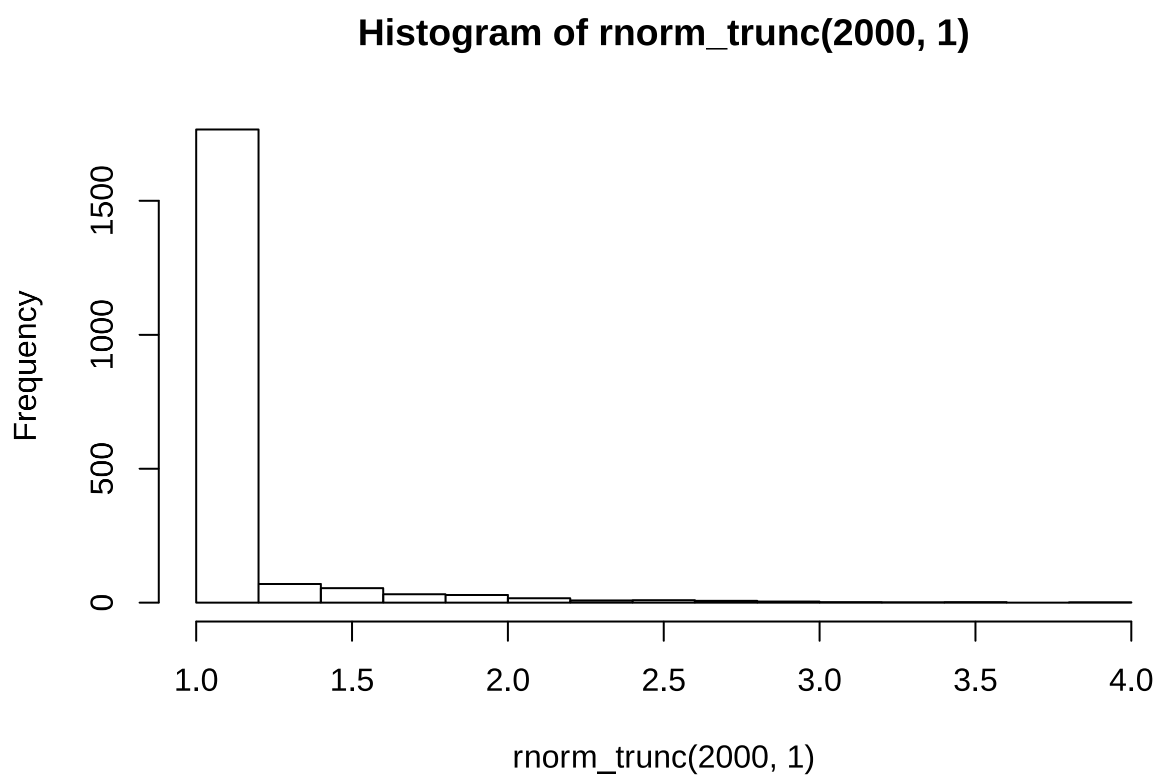


```
hist(all_norm[all_norm > 1])
```



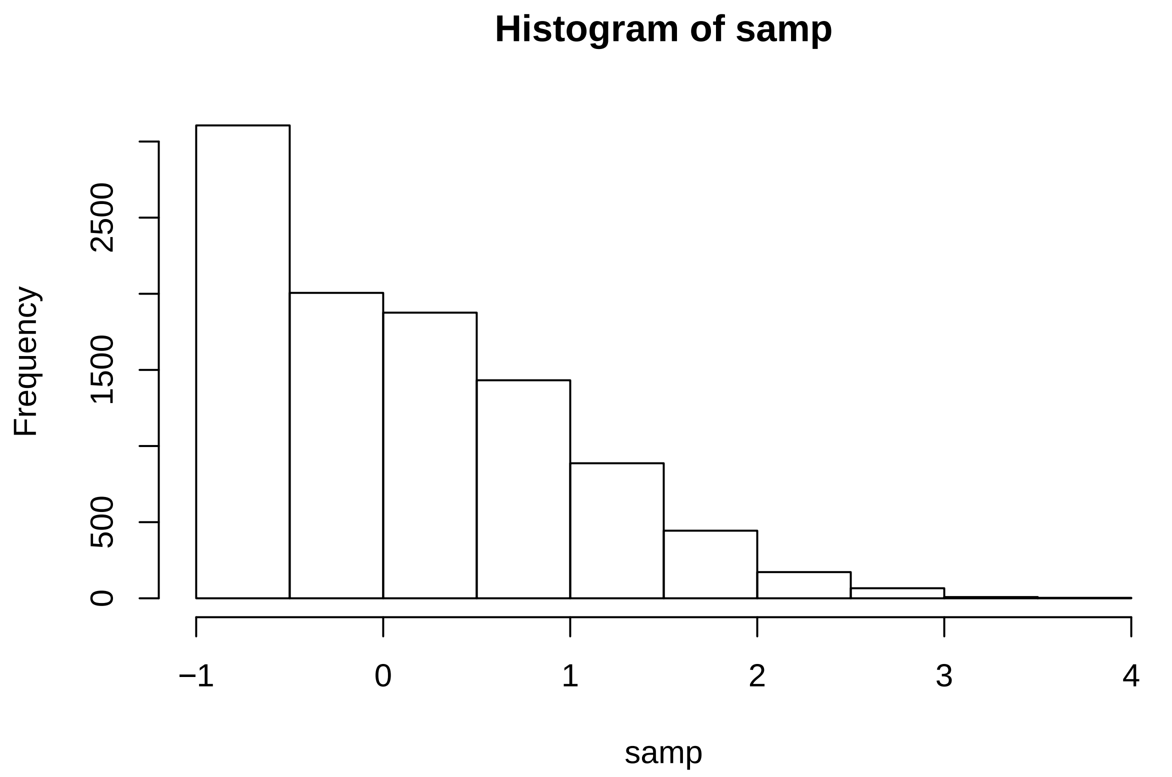
b. Write a function which takes two arguments n and min, and returns n independent random variables from a standard normal distribution truncated below by min. Let min default to 0.

```
rnorm_trunc = function(n, min = 0){
  X = rnorm(n)
  X = unlist(lapply(X, max, min))
  return(X)
}

hist(rnorm_trunc(2000, 1))
```



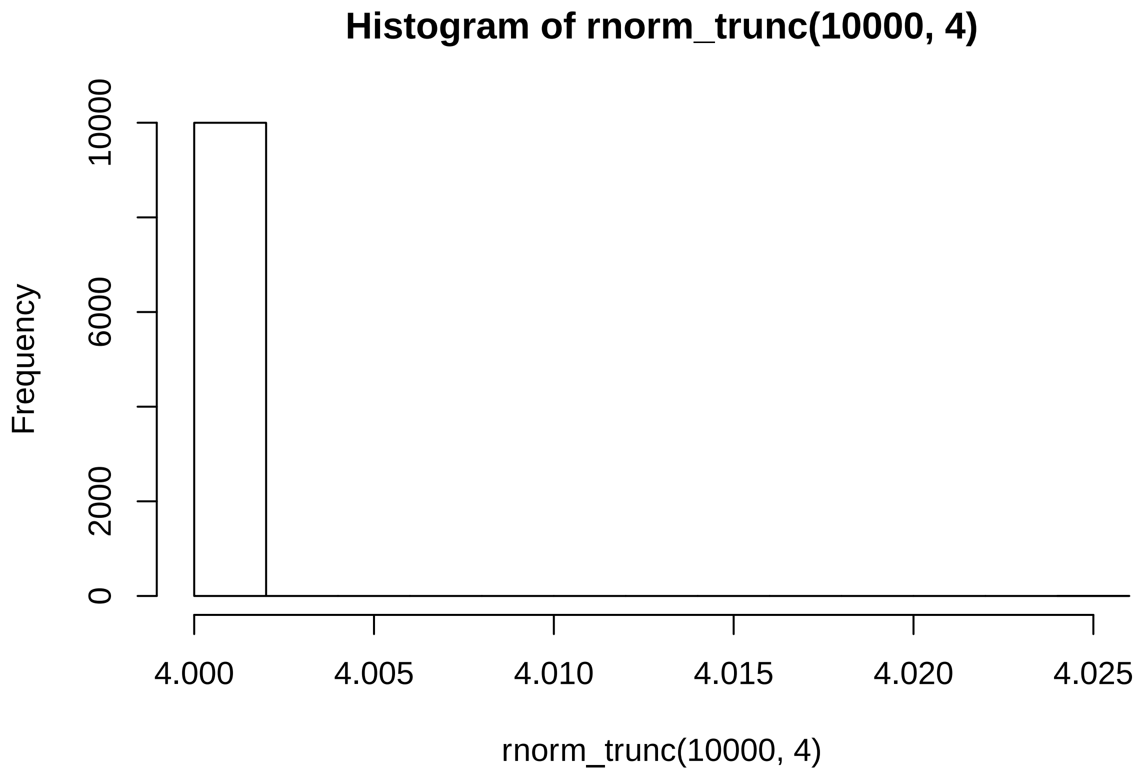
c. Generate 10k truncated normals with min set at -1 and plot as histogram

```r
samp = rnorm_trunc(10000, -1)
hist(samp, breaks  = 10)
```



d. what happens if min is large?

```r
hist(rnorm_trunc(10000,4))
```



Point here is that this method of rejection sampling is inefficient

2. Data

```r
library(MASS)
data(hills)
summary(hills)
```

```
##       dist            climb           time       
##  Min.   : 2.000   Min.   : 300   Min.   : 15.95  
##  1st Qu.: 4.500   1st Qu.: 725   1st Qu.: 28.00  
##  Median : 6.000   Median :1000   Median : 39.75  
##  Mean   : 7.529   Mean   :1815   Mean   : 57.88  
##  3rd Qu.: 8.000   3rd Qu.:2200   3rd Qu.: 68.62  
##  Max.   :28.000   Max.   :7500   Max.   :204.62
```

a. what sort of object is hills?

```r
class(hills)
```

```
## [1] "data.frame"
```

b. how many columns?

```r
ncol(hills)
```

```
## [1] 3
```

c. Change "Two Breweries" to "Three Breweries"

```
hills[which(rownames(hills) == 'Two Breweries'),]
```

```
##                dist climb   time
## Two Breweries   18  5200 170.25
```

```
row.names(hills)[which(rownames(hills) == 'Two Breweries')] <-'Three Breweries'
hills[which(rownames(hills) == 'Two Breweries'),]
```

```
## [1] dist  climb time
## <0 rows> (or 0-length row.names)
```

```
hills[which(rownames(hills) == 'Three Breweries'),]
```

```
##                  dist climb   time
## Three Breweries   18  5200 170.25
```

d. Find the mean time for races iwth a climb greater than 1000ft

```
with(hills[hills$climb > 1000,], mean(dist))
```

```
## [1] 10.41176
```

e. What sort of object is Orthodont? How is it different from `hills`?

```
library(nlme)
data(Orthodont)
head(Orthodont)
```

```
## Grouped Data: distance ~ age | Subject
##   distance age Subject  Sex
## 1     26.0   8     M01 Male
## 2     25.0  10     M01 Male
## 3     29.0  12     M01 Male
## 4     31.0  14     M01 Male
## 5     21.5   8     M02 Male
## 6     22.5  10     M02 Male
```

```
#class(Orthodont)
```

It's grouped data.

```
head(methods(print))
```

```
## [1] "print.abbrev"    "print.acf"       "print.AES"       "print.anova"
## [5] "print.Anova"     "print.anova.lme"
```

```
#nlme:::print.groupedData(Orthodont)
```

3. Recursion

The nth Fibonacci number is defined by the recusion $F_n = F_{n-1} + F_{n-2}$ with $F_0 = F_1 = 1$

a. Write a recursive function with argument `n` which returns the nth Fibonacci number

```
fibonacci = function(n){
  if(n < 0) {
    #print(n)
    F_n = 0
  }else if(n <= 1){  #note this is different than the example given in Recall (they used <=2) - has to
    #print(n)
    F_n = 1
  }else{
    #print(n)
    F_n = Recall(n-1) + Recall(n-2)
  }
  F_n
}

fibonacci(2)   #evaluated 3 times
```

```
## [1] 2
```

```
fibonacci(0)   #evaluated 1 time
```

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

```
fibonacci(1)   #evaluated 1 time
```

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

```
fibonacci(3)   #evaluated 5 times
```

```
## [1] 3
```

```
fib_2 = function(n){

  f_n_minus_1 = 1
  f_n_minus_2 = 1

  if(n < 0){
    return(0)
  }else if(n <= 1){
    return(1)
  }else{
    #initialize f_i at the first lag
    f_i = f_n_minus_1

     for(i in 2:n){

       #save old value
       f_i_old = f_i

       #get new value of f_i
       f_i = f_n_minus_1 + f_n_minus_2

       #update lag counters
       f_n_minus_2 = f_n_minus_1
       f_n_minus_1 = f_i_old

       print(c(i, f_i, f_n_minus_1, f_n_minus_2))
     }
    return(f_i)
  }
}
fib_2(1)
```

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

```r
fib_2(2)
```

```
## [1] 2 2 1 1
```

```
## [1] 2
```

```r
fib_2(3)
```

```
## [1] 2 2 1 1
## [1] 3 2 2 1
```

```
## [1] 2
```

```r
fib_2(4)
```

```
## [1] 2 2 1 1
## [1] 3 2 2 1
## [1] 4 3 2 2
```

```
## [1] 3
```

```r
fib_2(5)
```

```
## [1] 2 2 1 1
## [1] 3 2 2 1
## [1] 4 3 2 2
## [1] 5 4 3 2
```

```
## [1] 4
```

4. MCMC

a. X ~ Gamma(alpha, beta) alpha, beta ~ Exp(1)

```r
#vector of data x
alpha_true = 1
beta_true = 2

x = rgamma(100, shape = alpha_true, rate = beta_true)


# evaluate the posterior distribution of alpha given a vector of data
posterior = function(x, alpha, beta){

  # get prior density
  prior_alpha = dexp(alpha, rate = 1, log = T)  # alpha ~ exp(1)
  prior_beta = dexp(beta, rate = 1, log = T)  # beta ~ exp(1)

  likelihood = dgamma(x, alpha, beta, log = T)

  posterior = prior_alpha + prior_beta + sum(likelihood)

  return(posterior)
}

posterior(x, alpha = 1.1, beta = 2.3)
```

```
## [1] -15.64293
```

```r
posterior(x, alpha = 2, beta = 2)
```

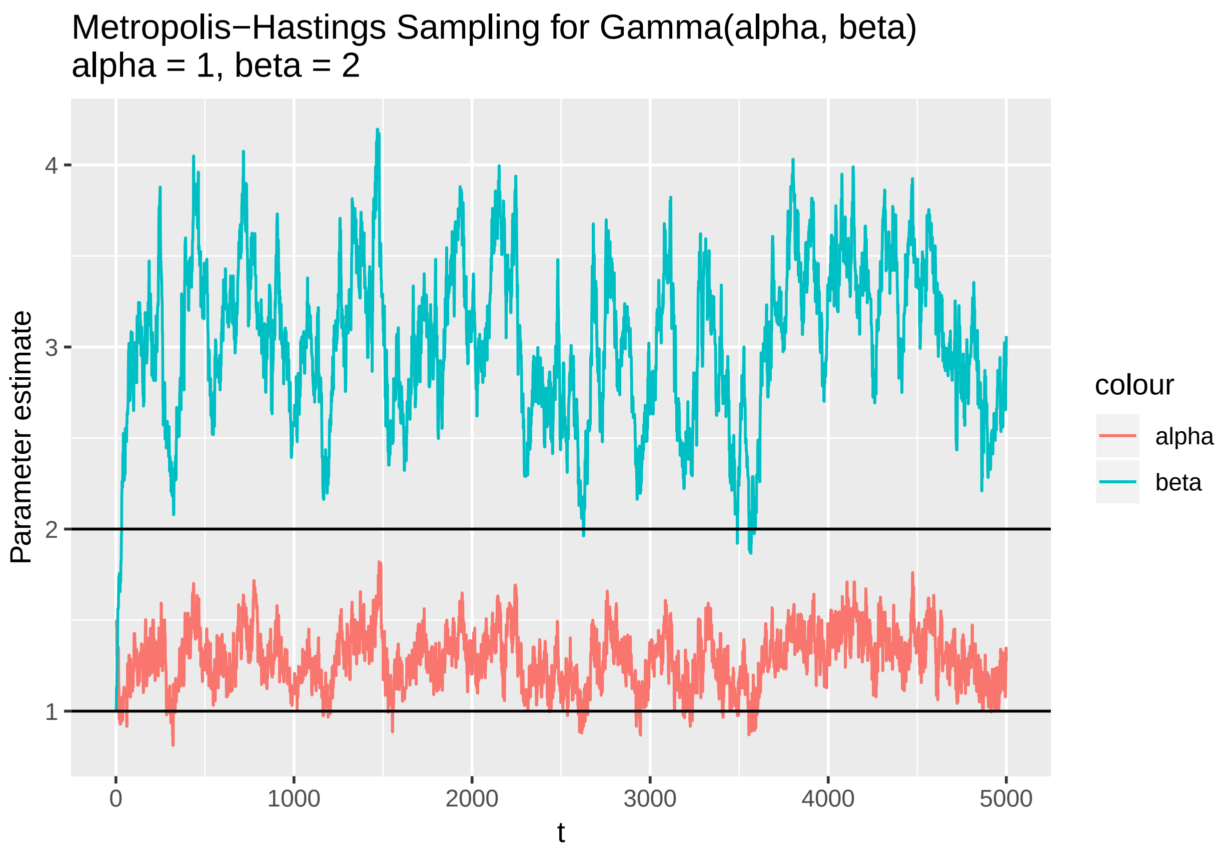
```
## [1] -78.01093
```

b. single metropolis hasting step $\alpha' = \alpha + \sigma Z_1$ $\beta' = \beta + \sigma Z_1$ where Z_1, Z_2 are iid Standard Normal $q'(\alpha'|\alpha)$ $N(\alpha, \sigma^2)$

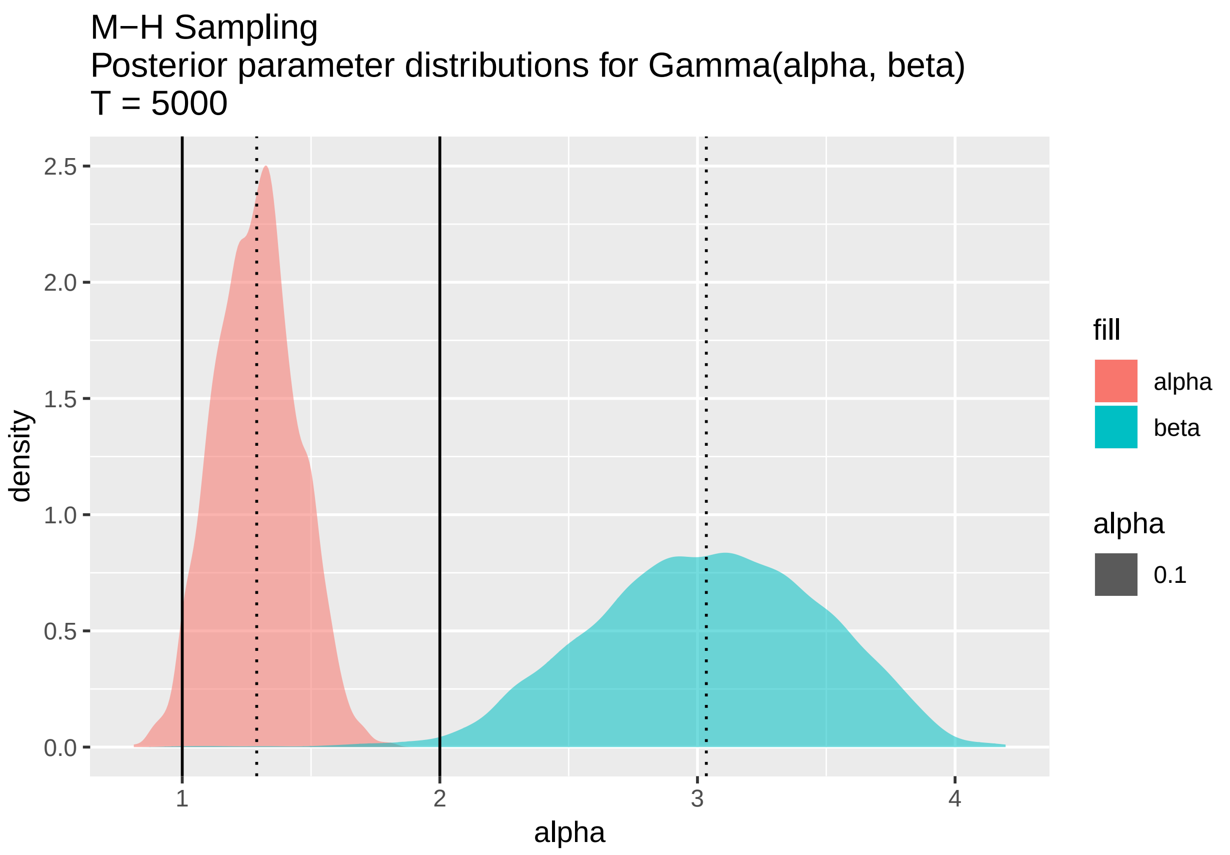
```r
step_MH = function(x, alpha, beta, sigma){

  alpha_prime = rnorm(n = 1, alpha, sigma)
  beta_prime = rnorm(n = 1, beta, sigma)

  a = exp(posterior(x, alpha = alpha_prime, beta = beta_prime) - posterior(x, alpha = alpha, beta = beta
  u = runif(1)
  if(u < a){
    data.frame(alpha = alpha_prime, beta = beta_prime)
  }else{
    data.frame(alpha = alpha, beta = beta)
  }
}

step_MH(x, alpha = 1.1, beta = 2.3, sigma = 0.01)
```

```
##      alpha    beta
## 1 1.092917 2.30664
```

```r
run_MH = function(N, x, alpha, beta, sigma){
  chains = data.frame(t = 0, alpha = alpha, beta = beta)
  for(t in 1:N){
    step = step_MH(x, alpha = chains[t,]$alpha, beta = chains[t,]$beta, sigma)
    chains[t+1, ] = c(t, step$alpha, step$beta)
  }
  return(chains)
}
chains = run_MH(5000, x = x, alpha = 1.5, beta = 1, sigma = 0.1)


ggplot(chains, aes(x = t, y = alpha)) + geom_line(aes(color = 'alpha')) + geom_line(aes(x = t, y = beta
  ggtitle(paste0("Metropolis-Hastings Sampling for Gamma(alpha, beta)\n", "alpha = ", alpha_true, ", be
  geom_hline(yintercept = alpha_true) +
  geom_hline(yintercept = beta_true) +
  ylab("Parameter estimate")
```



```r
ggplot(chains) + stat_density(aes(x = alpha, fill = 'alpha', alpha = 0.1)) +
  stat_density(aes(x = beta, fill = 'beta', alpha = 0.1)) +
  geom_vline(xintercept = mean(chains$alpha), linetype = 'dotted') +
  geom_vline(xintercept = alpha_true) +
  geom_vline(xintercept = mean(chains$beta), linetype = 'dotted') +
  geom_vline(xintercept = beta_true) +
  ggtitle(paste0("M-H Sampling\nPosterior parameter distributions for Gamma(alpha, beta)\nT = ", nrow(c
```



Let's see how this converges over time

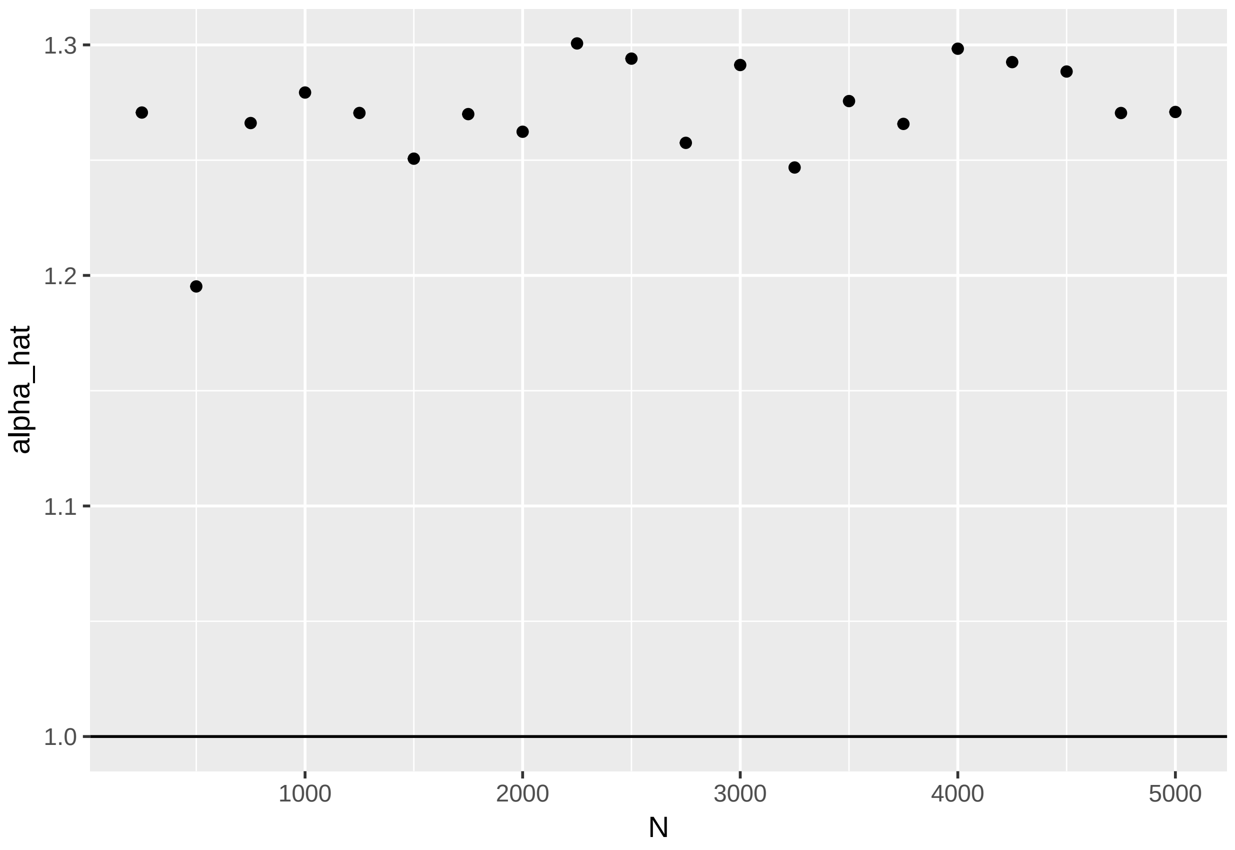
```r
Ts = seq(250, 5000, by = 250)

all_chains = lapply(Ts, FUN = run_MH, x = x, alpha = 1.5, beta = 1, sigma = 0.2)

all_chains = rbindlist(all_chains)
all_chains = as.data.table(all_chains)
all_chains[, N := rep(Ts, times = Ts + 1)]


all_chains[, .(mean(alpha), .N), N]
```

```
##        N       V1    N
##  1:  250 1.270656  251
##  2:  500 1.195231  501
##  3:  750 1.266099  751
##  4: 1000 1.279338 1001
##  5: 1250 1.270465 1251
##  6: 1500 1.250645 1501
##  7: 1750 1.269973 1751
##  8: 2000 1.262321 2001
##  9: 2250 1.300630 2251
## 10: 2500 1.294036 2501
## 11: 2750 1.257512 2751
## 12: 3000 1.291271 3001
## 13: 3250 1.246820 3251
## 14: 3500 1.275628 3501
## 15: 3750 1.265723 3751
## 16: 4000 1.298344 4001
## 17: 4250 1.292533 4251
## 18: 4500 1.288440 4501
## 19: 4750 1.270453 4751
## 20: 5000 1.270900 5001
```

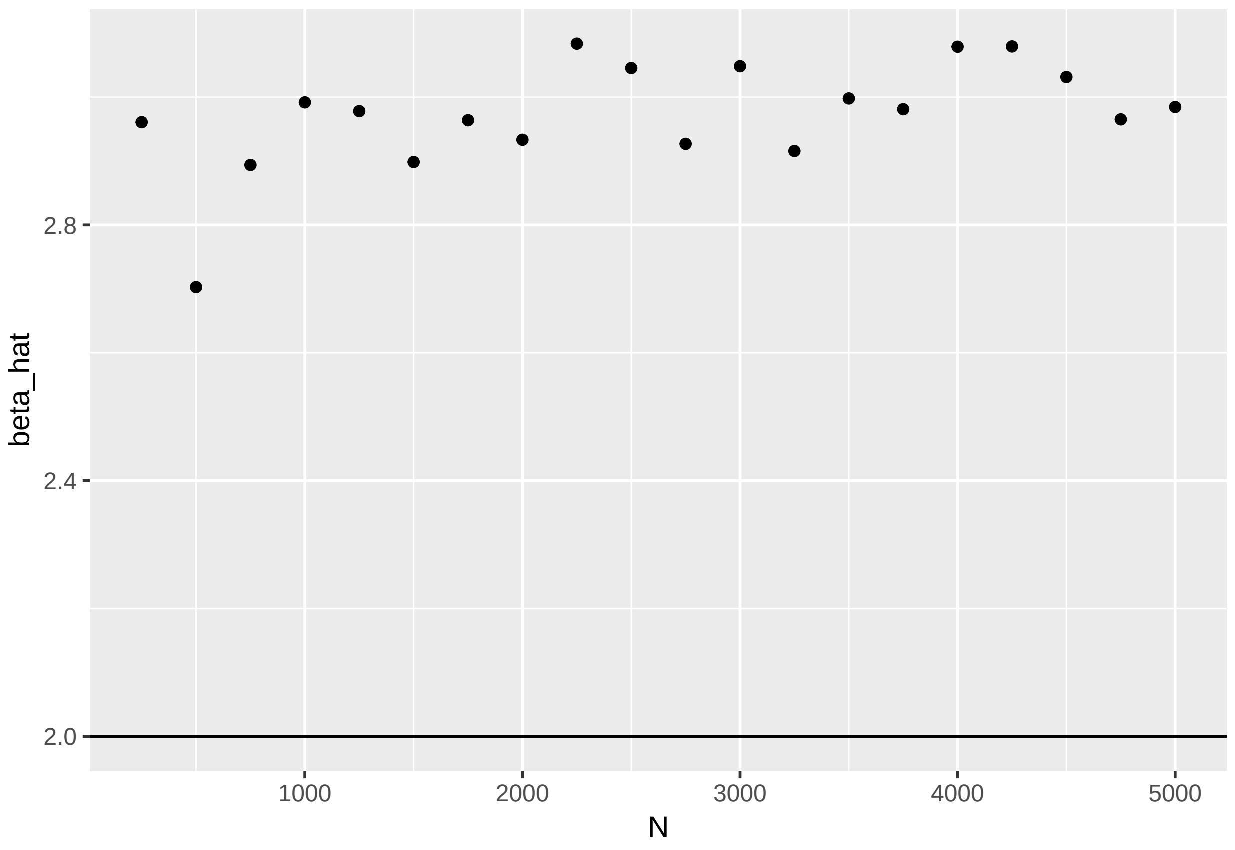
```
all_chains[, .(mean(beta), .N), N]
```

```
##        N       V1    N
##  1:  250 2.960724  251
##  2:  500 2.702690  501
##  3:  750 2.893831  751
##  4: 1000 2.991720 1001
##  5: 1250 2.978000 1251
##  6: 1500 2.898417 1501
##  7: 1750 2.963744 1751
##  8: 2000 2.933248 2001
##  9: 2250 3.083511 2251
## 10: 2500 3.045374 2501
## 11: 2750 2.926844 2751
## 12: 3000 3.048239 3001
## 13: 3250 2.915641 3251
## 14: 3500 2.997848 3501
## 15: 3750 2.981003 3751
## 16: 4000 3.078727 4001
## 17: 4250 3.079169 4251
## 18: 4500 3.031384 4501
## 19: 4750 2.965151 4751
## 20: 5000 2.984470 5001
```

```
ggplot(all_chains[, .(alpha_hat = mean(alpha)), N]) + geom_point(aes(x = N, y = alpha_hat)) + geom_hlin
```



```
ggplot(all_chains[, .(beta_hat = mean(beta)), N]) + geom_point(aes(x = N, y = beta_hat)) + geom_hline(y
```


beta_hat
2.8
2.4
2.0
1000
2000
3000
4000
5000
N