CDT Module 1 - R Review

Valerie Bradley

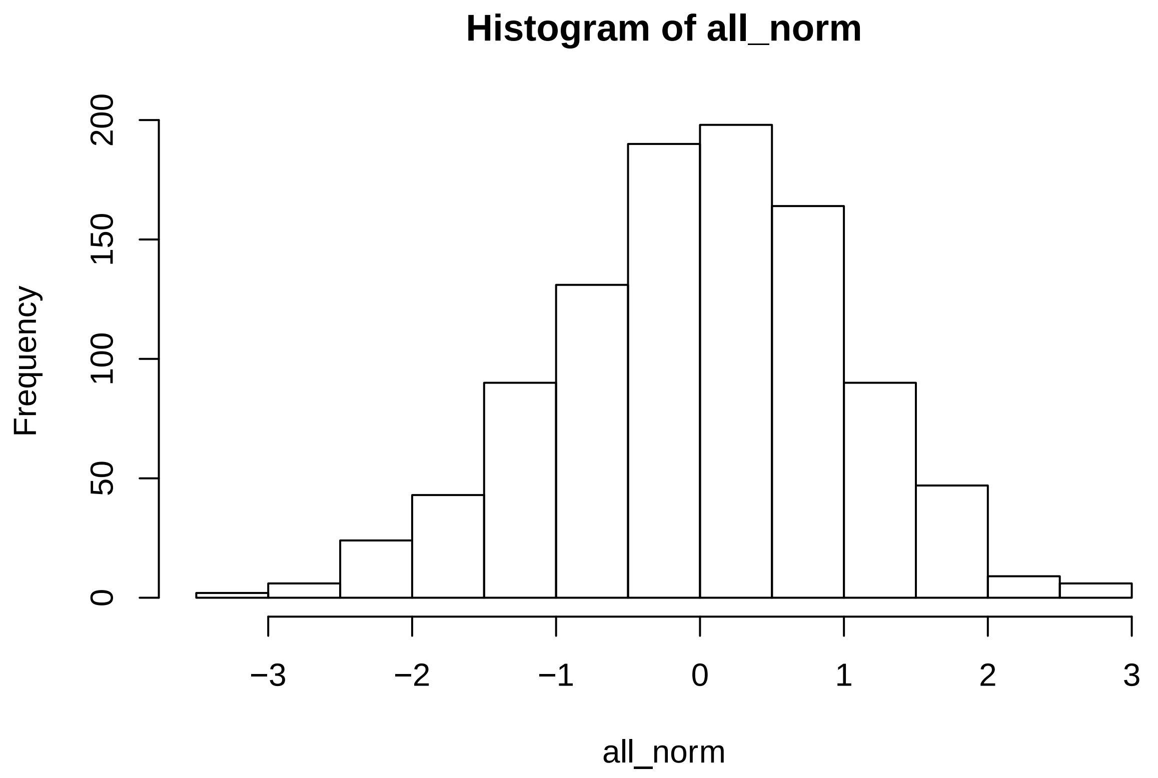
9/19/2018

1. Vectors

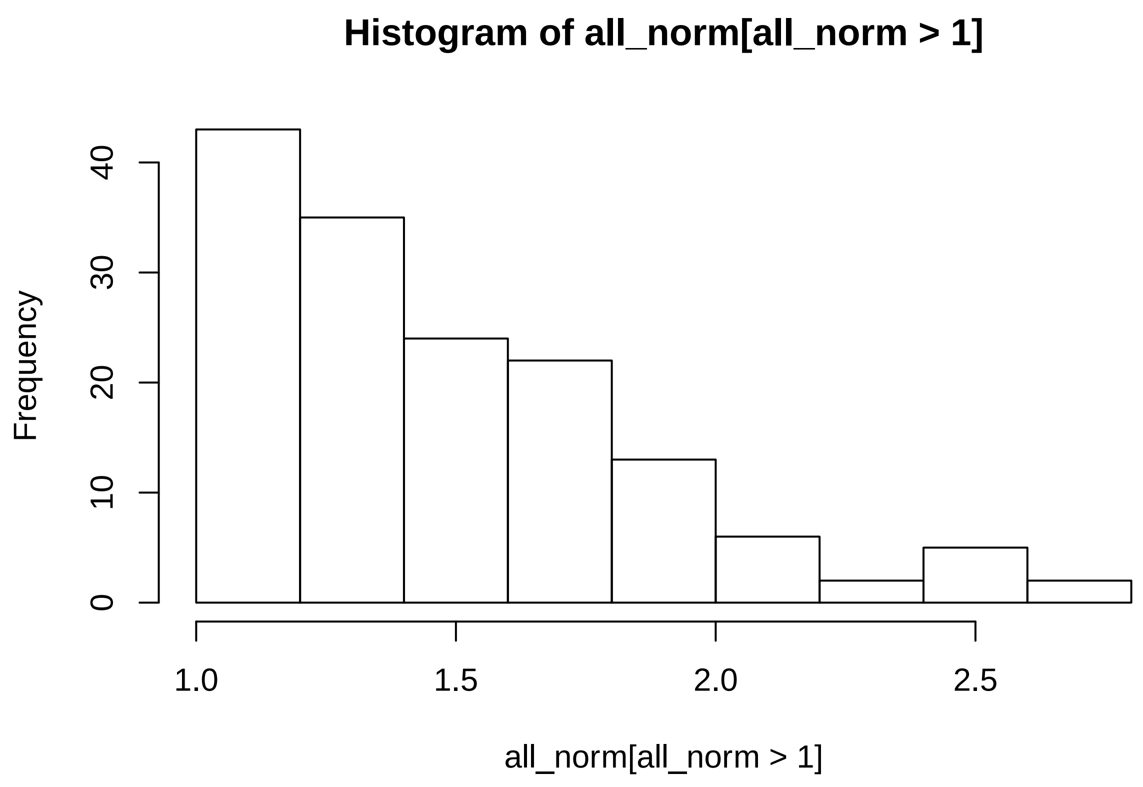
a. Generate 100 standard normal random variables, and keep only the ones which are greater than 1. Don't use a loop!

Problem with this is that the number of R.V.s generated

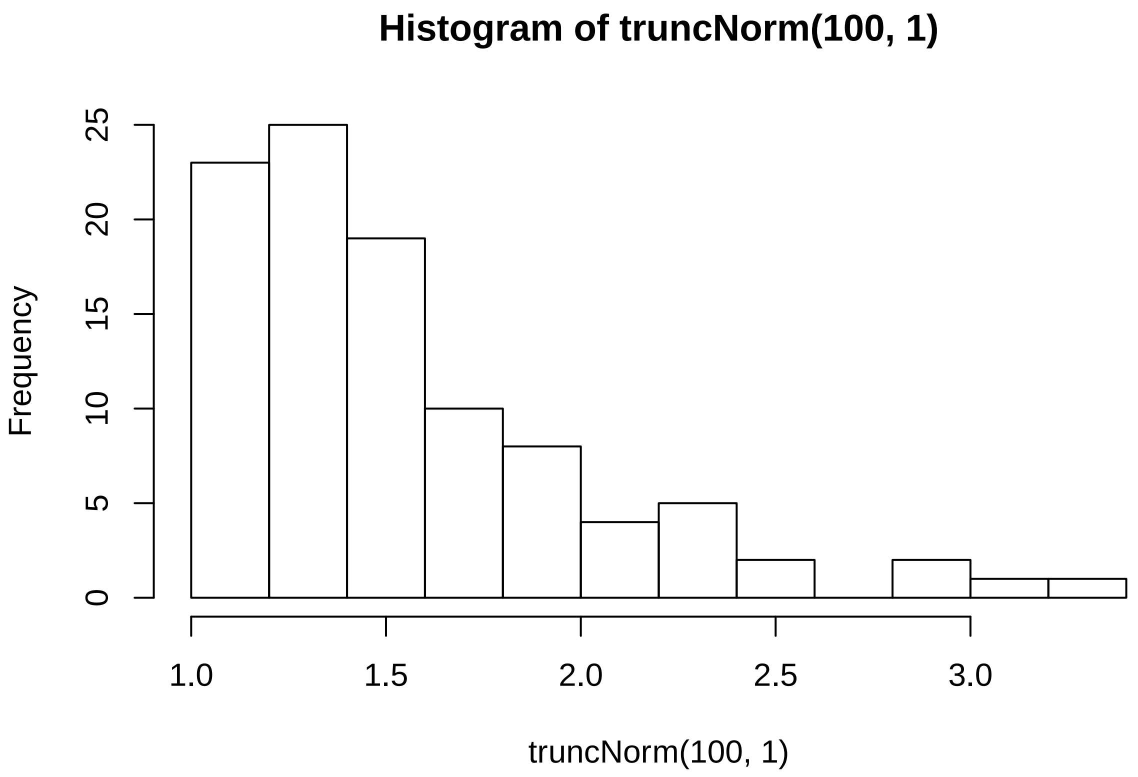
```
all_norm = rnorm(1000)
hist(all_norm)
```



```
hist(all_norm[all_norm > 1], breaks = 10)
```

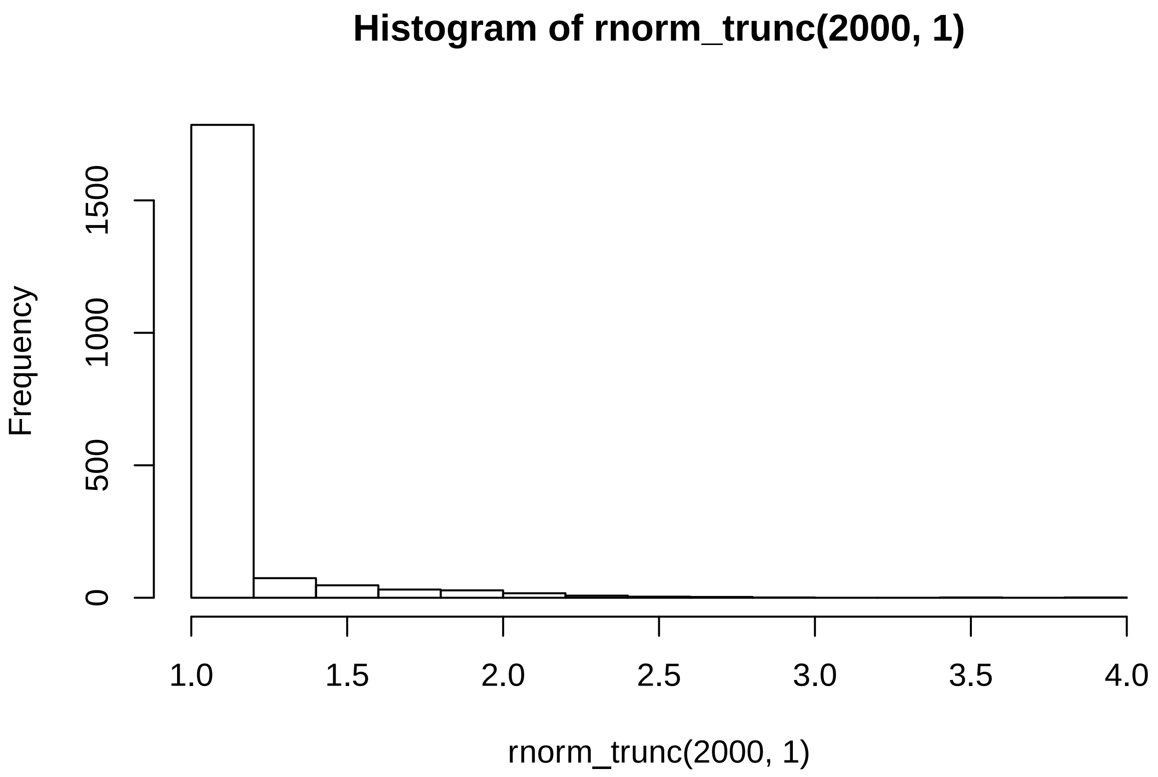


```
truncNorm <- function(n, min){
  out <- c()

  while(length(out) < n){
    tmp <- rnorm(n)
    tmp <- tmp[tmp > min]
    out <- c(out, tmp)    #will copy out each time -- uses more memory.  alternately, you could initiali
  }

  out[seq_len(n)]   #sequence of length n
}

hist(truncNorm(100,1), breaks = 15)
```



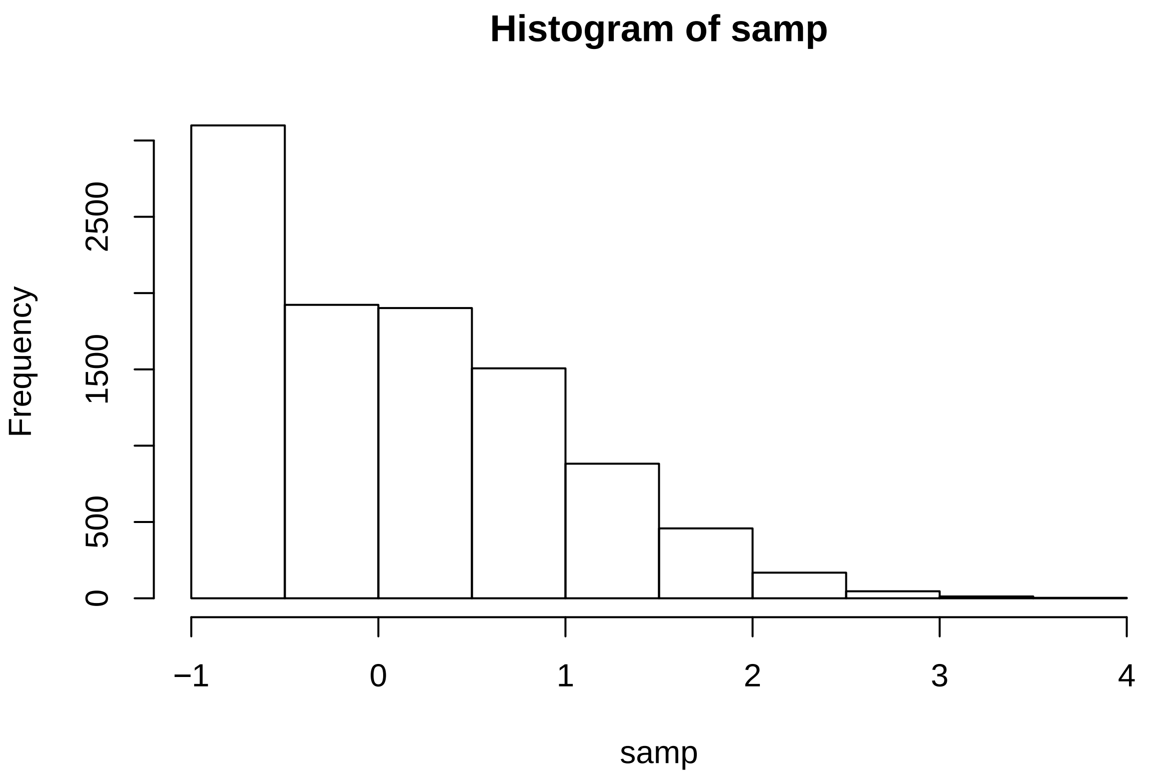
b. Write a function which takes two arguments n and min, and returns n independent random variables from a standard normal distribution truncated below by min. Let min default to 0.

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

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



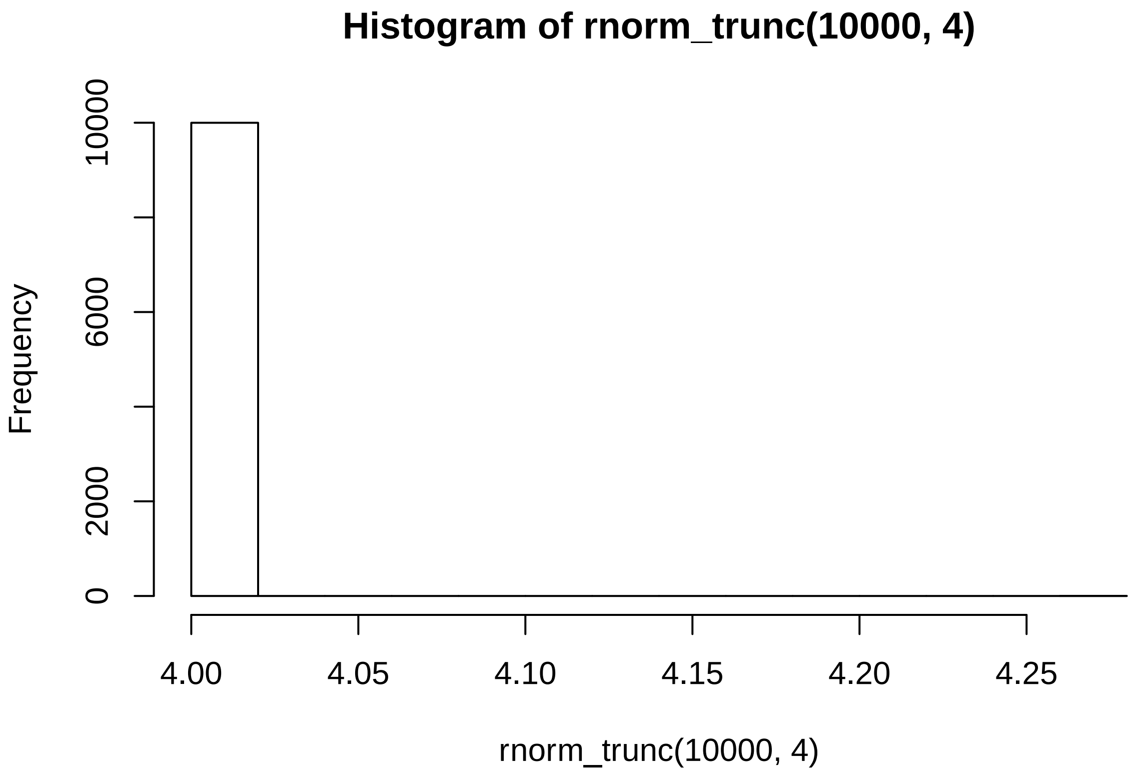
c. Generate 10k truncated normals with min set at -1 and plot as histogram

```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

Important point here: R is bad a 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
```

```r
    #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
```

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

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

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

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

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

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

```r
fib_2 = function(n){
  fib_seq = c(1,2)

  if(n < 0) return(0)
  if(n <= 1) return(1)
  if(n == 2) return(2)

  for(i in 3:n){
       fib_seq[i] <- fib_seq[i-1] + fib_seq[i-2]
  }

  return(fib_seq[n])
}
fib_2(1)
```

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

```r
fib_2(2)
```

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

```r
fib_2(3)
```

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

```r
fib_2(4)
```

```
## [1] 5
```

```r
fib_2(5)
```

```
## [1] 8
```

```r
fib_2(100)
```

```
## [1] 5.731478e+20
```

4. MCMC

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

```
#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] -7.771917
```

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

```
## [1] -90.33592
```

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

```
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)
```

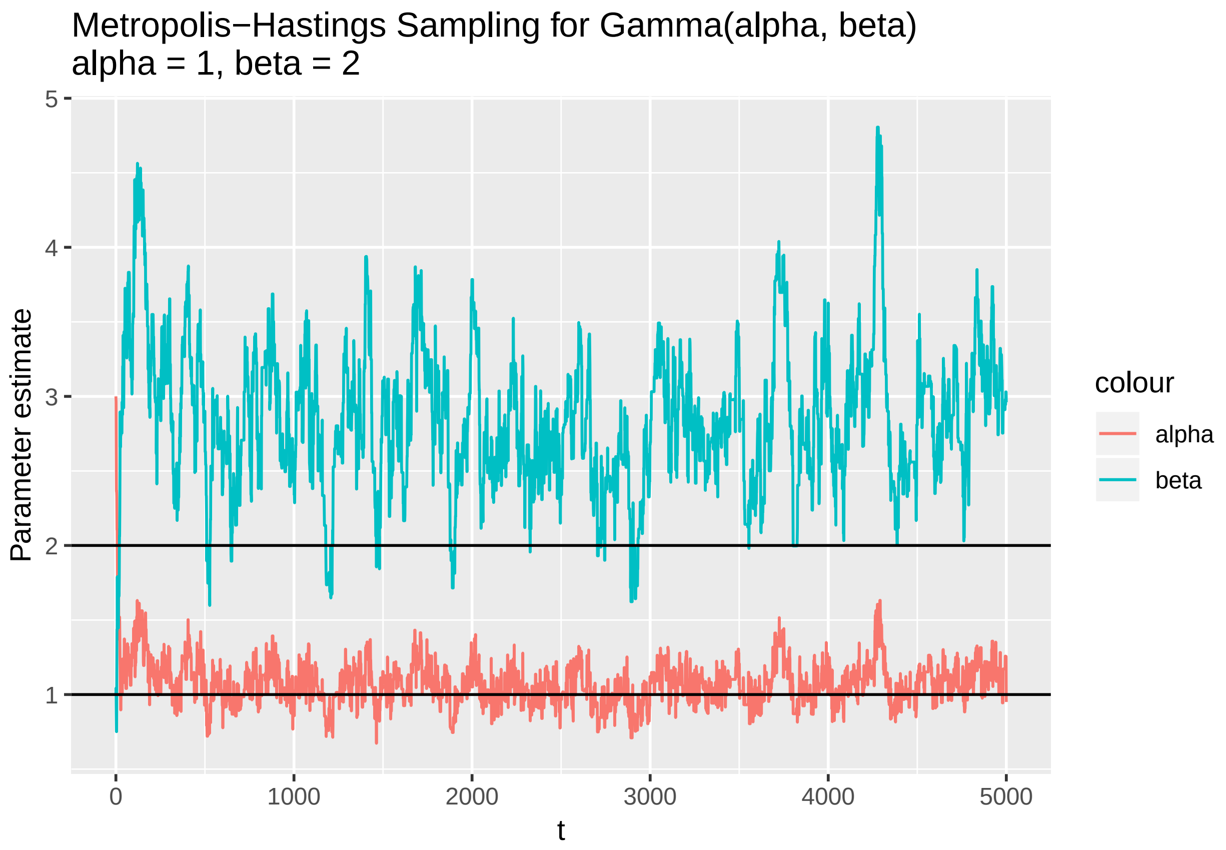
```
##     alpha     beta
## 1 1.09244 2.305277
```

```
run_MH = function(N, x, alpha, beta, sigma){
  chains = data.frame(t = 0, alpha = alpha, beta = beta)
```

```r
  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)
}
```

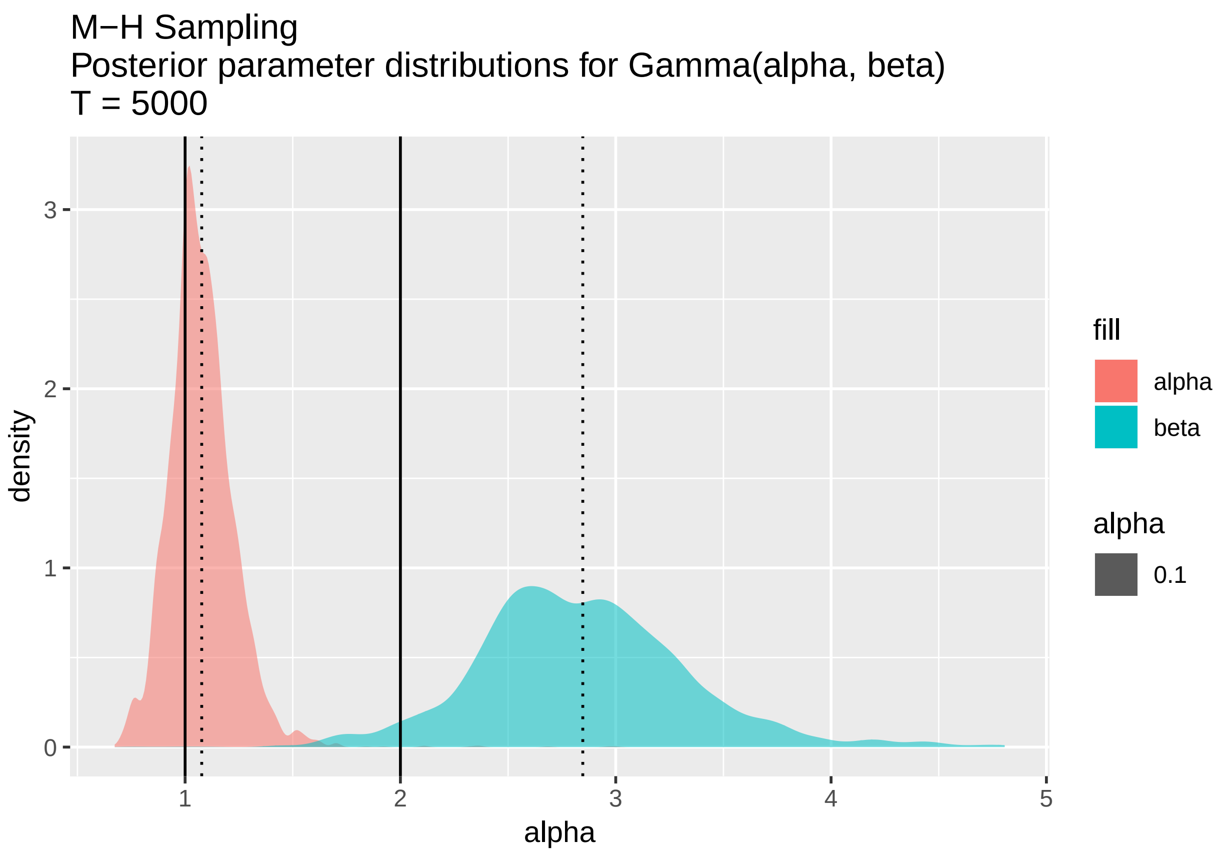
Now, run the full M-H algorithm and plot the resulting chains

```r
chains = run_MH(5000, x = x, alpha = 3, beta = 1, sigma = 2/sqrt(length(x)))

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")
```



Plot the posterior distributions of the parameters

```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

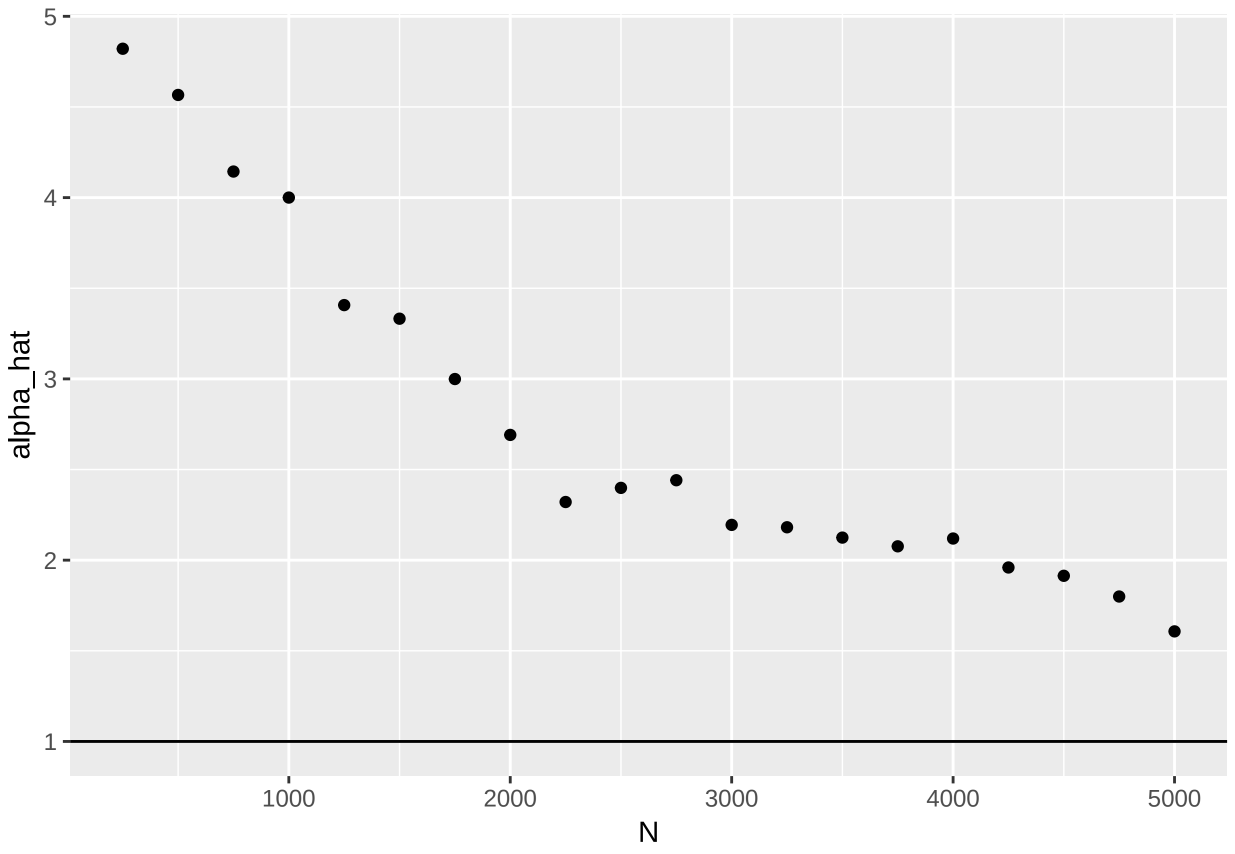
```
Ts = seq(250, 5000, by = 250)

all_chains = lapply(Ts, FUN = run_MH, x = x, alpha = 5, beta = 0.3, sigma = 0.01)

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 4.821162  251
##  2:  500 4.566323  501
##  3:  750 4.143756  751
##  4: 1000 4.000220 1001
##  5: 1250 3.407072 1251
##  6: 1500 3.332075 1501
##  7: 1750 2.999205 1751
##  8: 2000 2.691003 2001
##  9: 2250 2.320869 2251
## 10: 2500 2.398723 2501
## 11: 2750 2.441058 2751
## 12: 3000 2.194616 3001
## 13: 3250 2.181507 3251
## 14: 3500 2.124507 3501
## 15: 3750 2.076403 3751
## 16: 4000 2.119446 4001
## 17: 4250 1.959706 4251
## 18: 4500 1.913846 4501
## 19: 4750 1.799522 4751
## 20: 5000 1.607151 5001
```

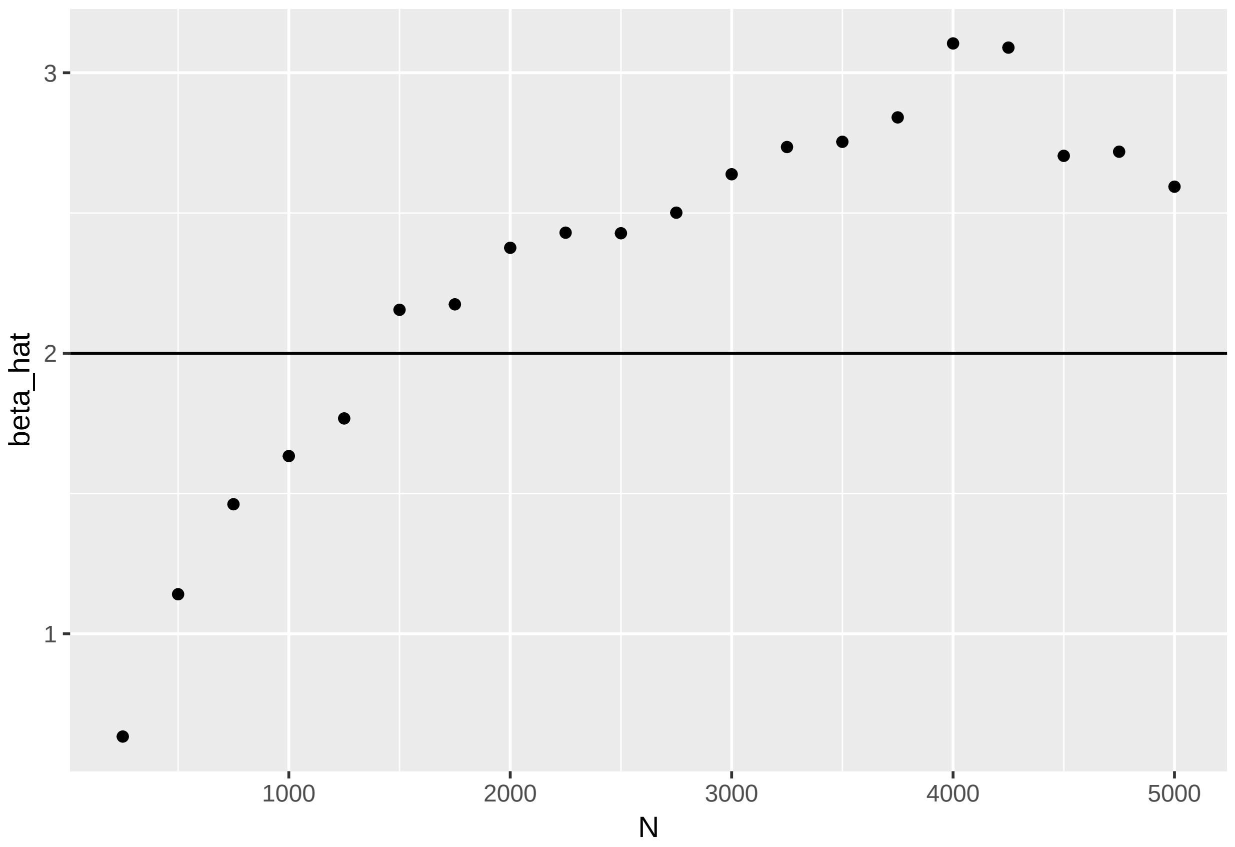
```
all_chains[, .(mean(beta), .N), N]
```

```
##        N        V1    N
##  1:  250 0.6337948  251
##  2:  500 1.1410063  501
##  3:  750 1.4618715  751
##  4: 1000 1.6336097 1001
##  5: 1250 1.7677558 1251
##  6: 1500 2.1549477 1501
##  7: 1750 2.1744548 1751
##  8: 2000 2.3760333 2001
##  9: 2250 2.4298656 2251
## 10: 2500 2.4278791 2501
## 11: 2750 2.5011539 2751
## 12: 3000 2.6381690 3001
## 13: 3250 2.7352140 3251
## 14: 3500 2.7537821 3501
## 15: 3750 2.8408260 3751
## 16: 4000 3.1044815 4001
## 17: 4250 3.0896072 4251
## 18: 4500 2.7037493 4501
## 19: 4750 2.7185277 4751
## 20: 5000 2.5938450 5001
```

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

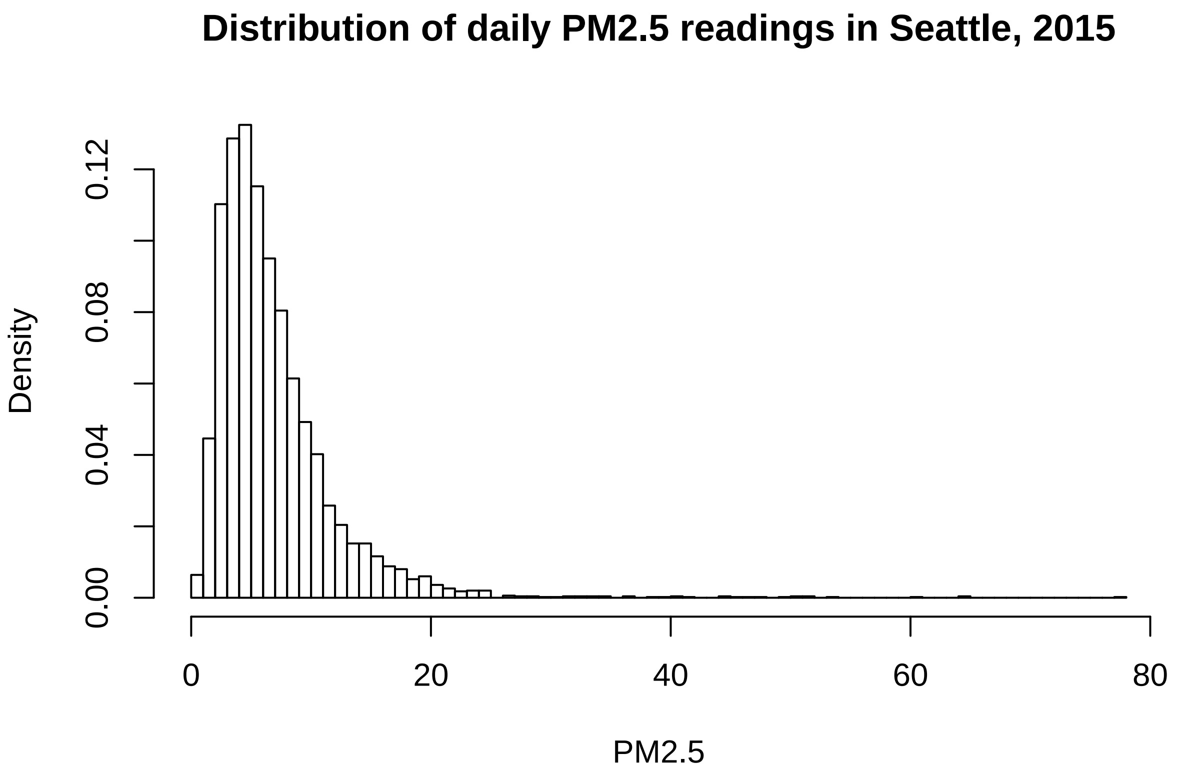


```
ggplot(all_chains[, .(beta_hat = mean(beta)), N]) + geom_point(aes(x = N, y = beta_hat)) + geom_hline(yi
```



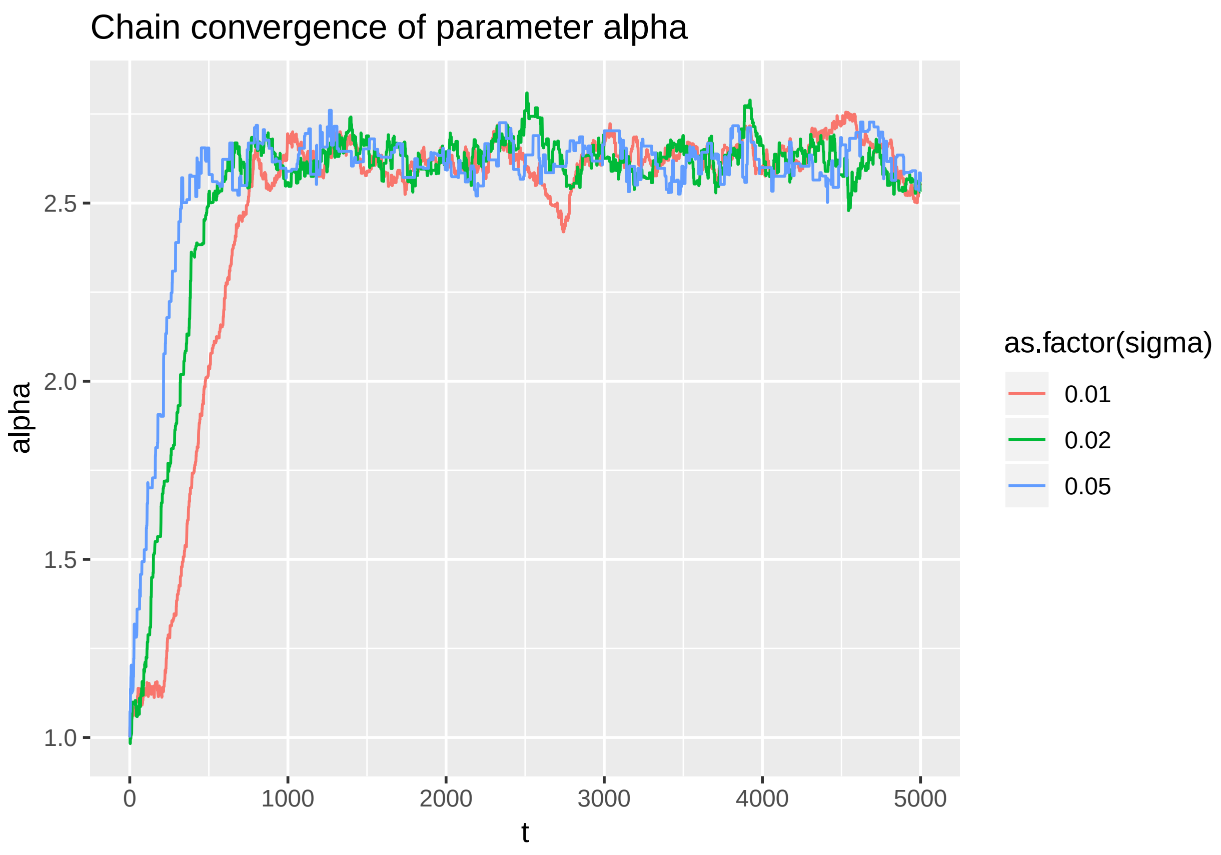
d. Air pollution data.

```
x <- scan("/Users/valeriebradley/Documents/Oxford/Module 1/R review/airpol.txt")
hist(x, breaks = 100, freq = FALSE, main = "Distribution of daily PM2.5 readings in Seattle, 2015", xlab
```

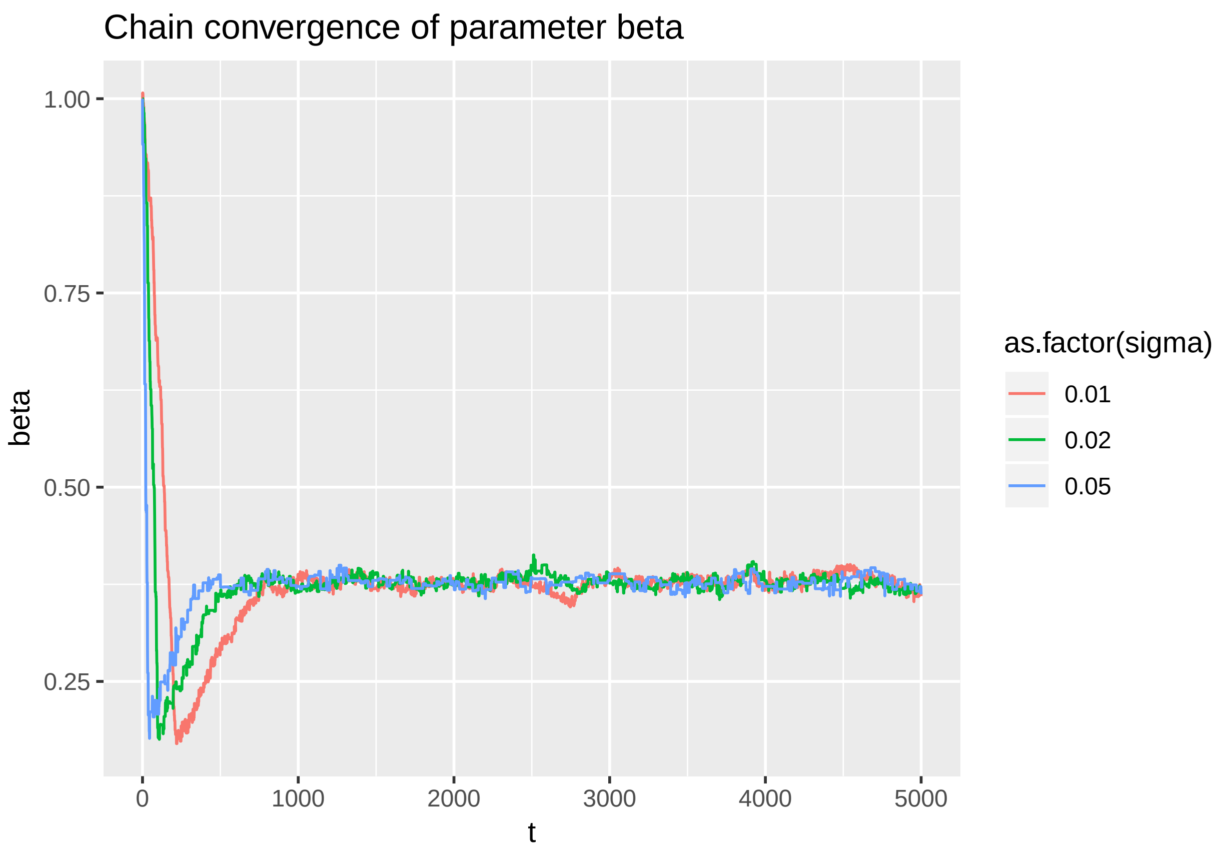


Model data as iid Gamma using Exp(1) priors from above. Run for 5,000 iterations with starting points $\alpha = 1$, $\beta = 1$.

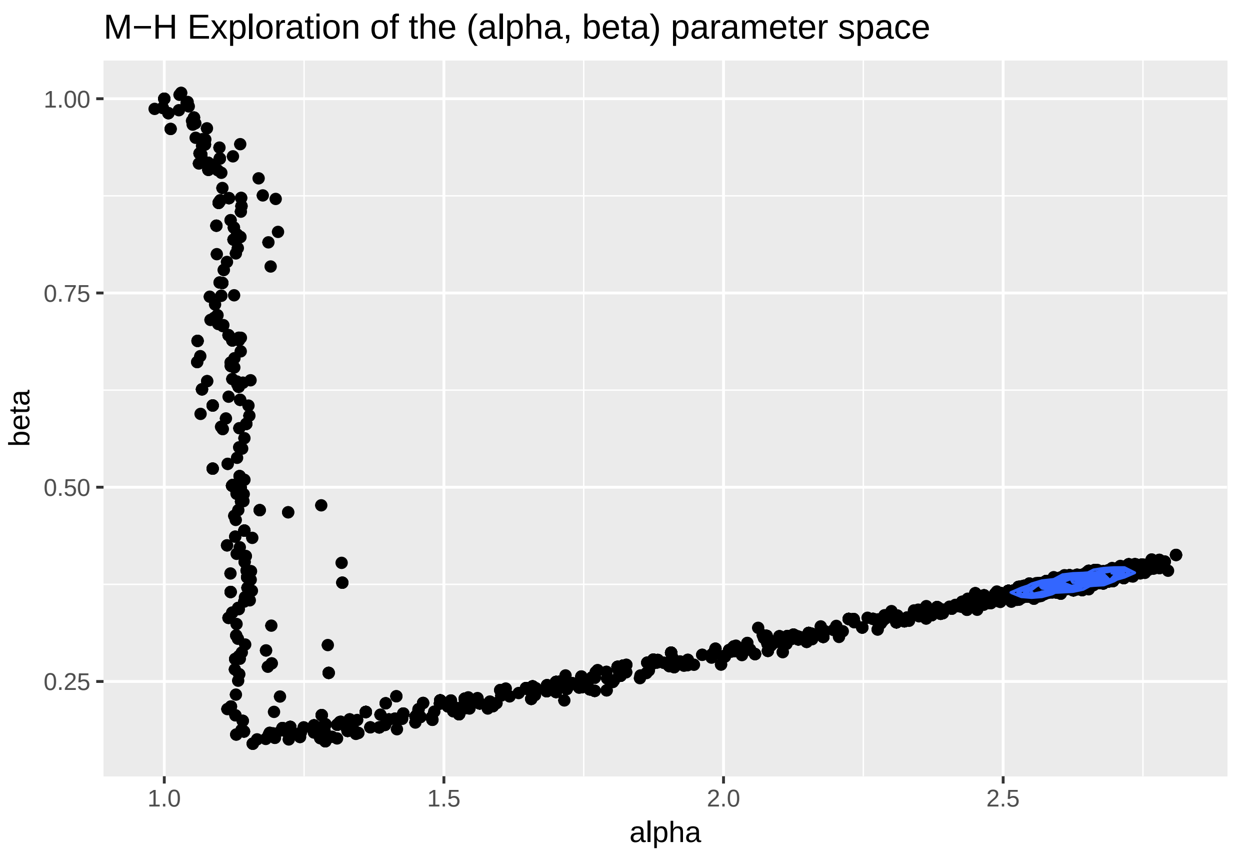
```r
sigmas = c(0.01, 0.02, 0.05)
chains_airpol = lapply(sigmas, FUN = run_MH, N = 5000, x = x, alpha = 1, beta = 1)

chains_airpol = rbindlist(chains_airpol)
chains_airpol = as.data.table(chains_airpol)
chains_airpol[, sigma := sort(rep(sigmas, times = 5000 + 1))]

ggplot(chains_airpol) + geom_line(aes(x = t, y = alpha, group = sigma, color = as.factor(sigma))) +
  ggtitle("Chain convergence of parameter alpha")
```



```r
ggplot(chains_airpol) + geom_line(aes(x = t, y = beta, group = sigma, color = as.factor(sigma))) +
  ggtitle("Chain convergence of parameter beta")
```



```r
ggplot(chains_airpol, aes(x = alpha, y = beta)) + geom_point() +
  ggtitle("M-H Exploration of the (alpha, beta) parameter space") +
  geom_density2d()
```

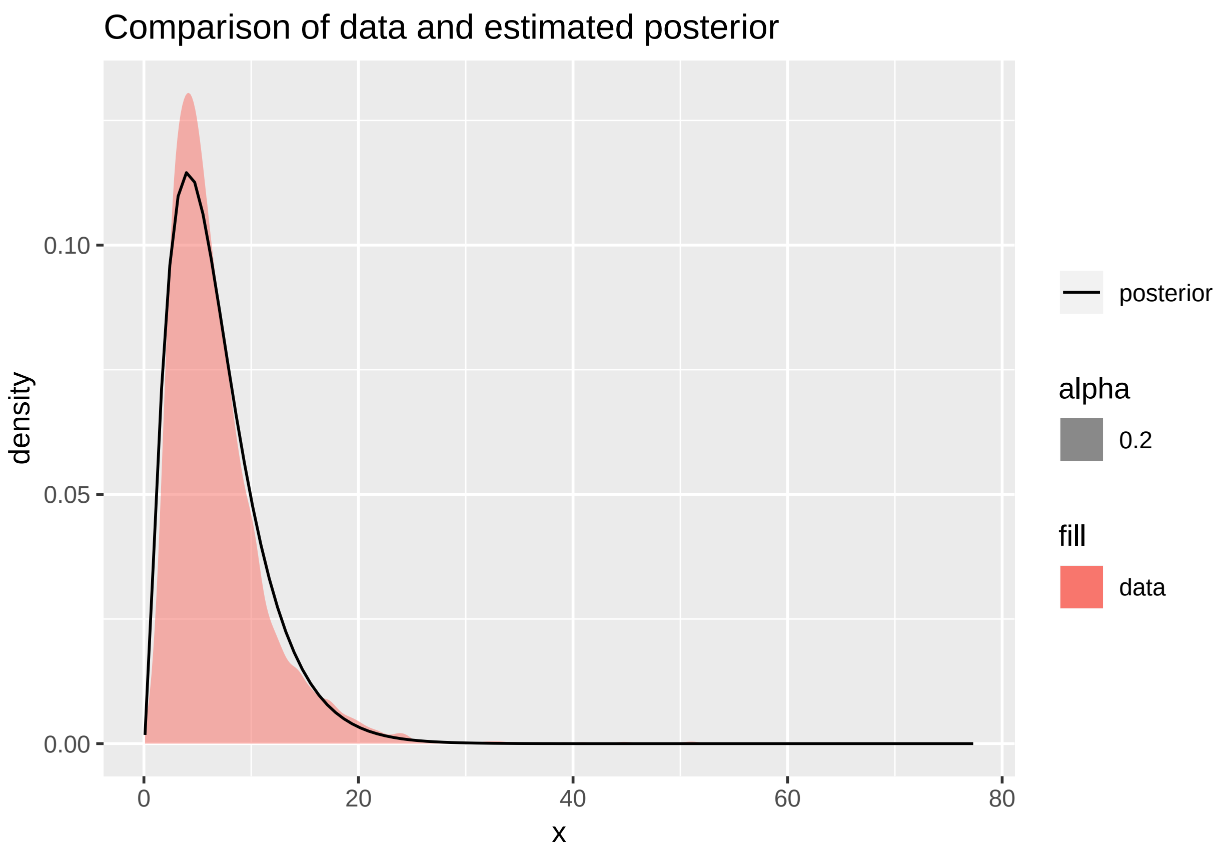


e. Find posterior means of α and β. Plot density of corresponding Gamma distribution over the histogram of the data.

```
chains_airpol[, lapply(.SD, mean), .SDcols = c('alpha', 'beta')]
```

```
##      alpha      beta
## 1: 2.52988 0.3749879
```

```
ggplot(data.frame(x = x)) + stat_density(aes(x = x, fill = 'data', alpha = 0.2)) +
  stat_function(fun = dgamma, args = list(shape = chains_airpol[, mean(alpha)], rate = chains_airpol[, 
  scale_colour_manual("",values = 'black') +
  ggtitle('Comparison of data and estimated posterior')
```



5. Methods

```
setClass('biv', representation(x = "numeric", y = "numeric"))
new_df = new('biv', x = rnorm(n = 20), y = rpois(n = 20, lambda = 5))
new_df
```

```
## An object of class "biv"
## Slot "x":
##  [1]  2.22093222  0.83426764 -1.11433932  0.27749348 -1.43302676
##  [6] -0.79321681  1.06051949 -2.49921643 -1.08725894 -1.22343514
## [11] -1.68157423 -0.28758539  0.45722618  0.04790570  0.33238820
## [16]  0.39156596  0.86217702  0.04231375  1.17919519  0.49887685
##
## Slot "y":
##  [1] 9 6 9 4 4 3 7 6 6 1 3 6 4 5 3 4 6 5 2 4
```

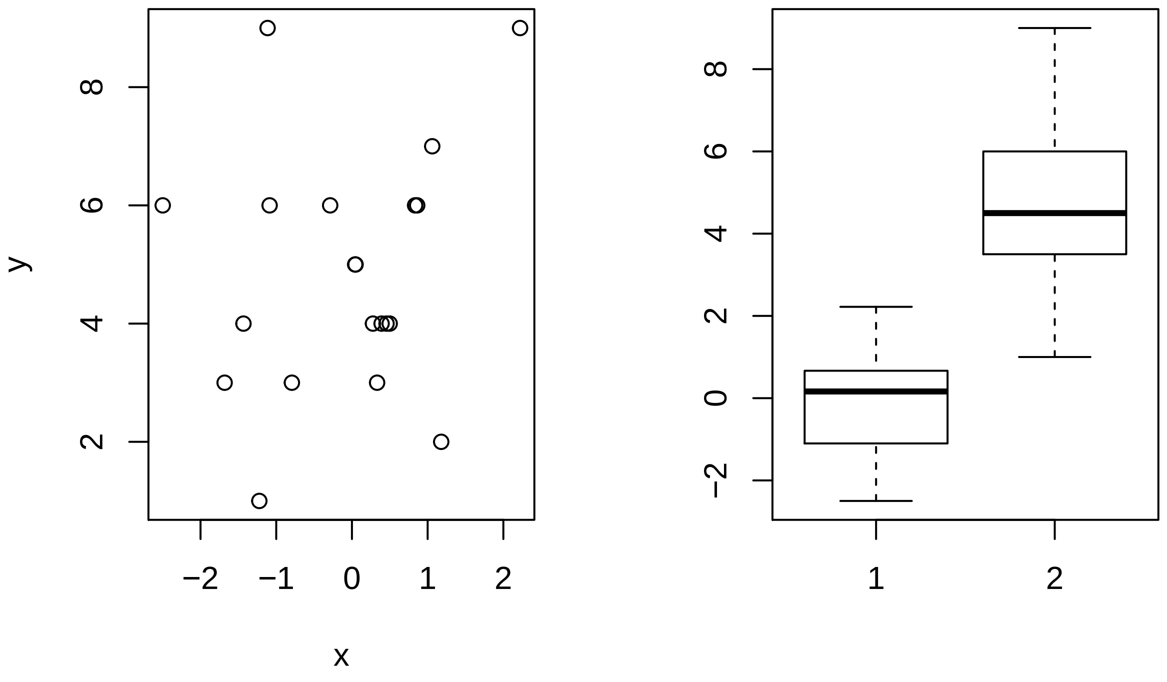
Create a print method (that invisibly returns the object)

```
setMethod('print', 'biv', function(x){
  cat(paste0("Bivariate data, ", length(x@x), " entries\n"))
  cat(paste('x :' , paste(x@x[1:min(length(x@x),6)], collapse = " "), "...\n"))
  cat(paste('y :' , paste(x@y[1:min(length(x@y),6)], collapse = " "), "..."))
  invisible(x)
})
print(new_df)
```

```
## Bivariate data, 20 entries
## x : 2.22093222179919 0.834267639367716 -1.11433931599906 0.277493478109035 -1.43302675842844 -0.7932
## y : 9 6 9 4 4 3 ...
```

Construct a plot method for class `biv`

```r
setMethod('plot', 'biv', function(x){
  layout(matrix(c(1,2), 1, 2, byrow = TRUE))
  plot(x = x@x, y = x@y, xlab = 'x', ylab = 'y')
  boxplot(x@x, x@y)
})
plot(new_df)
```



6. Functions

Create a function to return a model matrix from 2 variables

```r
getSimpleMatrix = function(x,z){
  if(length(x) != length(z)) stop(print("x and z must have the same number of entries"))

  n_obs = length(x)
  intercept = rep(1, n_obs)
  predictors = c(x,z)
  interactions = x * z

  return(matrix(c(intercept, predictors, interactions), ncol = 4, byrow = F))
}

getSimpleMatrix(x = c(1,2,3,4,5), z = c(6,7,8,9,0))
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    1    6    6
## [2,]    1    2    7   14
## [3,]    1    3    8   24
## [4,]    1    4    9   36
## [5,]    1    5    0    0
```

Check with model.matrix

```
model.matrix(as.formula(~1+x+z+x:z), data = data.frame(x = c(1,2,3,4,5), z = c(6,7,8,9,0)))
```

```
##   (Intercept) x z x:z
## 1           1 1 6   6
## 2           1 2 7  14
## 3           1 3 8  24
## 4           1 4 9  36
## 5           1 5 0   0
## attr(,"assign")
## [1] 0 1 2 3
```

Create a function to return a model matrix from n variables

```
getMatrix = function(...){
  inputs = list(...)

  #check input length
  if(length(unique(unlist(lapply(inputs, length)))) > 1) stop("All inputs must have the same number of o

  n_obs = length(x)
  n_vars = length(inputs)

 predictors = matrix(unlist(inputs), ncol = n_vars, byrow = F)
 interactions = apply(combn(x = n_vars, 2), 2, function(x){
    inputs[[x[1]]] * inputs[[x[2]]]
  })

 return(cbind(rep(1, n_obs), predictors, interactions))
}

getMatrix(x = c(1,2,3,4,5), z = c(6,7,8,9,0), y = c(2,4,6,8,10))
```

```
## Warning in cbind(rep(1, n_obs), predictors, interactions): number of rows
## of result is not a multiple of vector length (arg 1)
```

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7]
## [1,]    1    1    6    2    6    2   12
## [2,]    1    2    7    4   14    8   28
## [3,]    1    3    8    6   24   18   48
## [4,]    1    4    9    8   36   32   72
## [5,]    1    5    0   10    0   50    0
```

7. Mixtures

$X^{(i)} = (X_{i1}, ..., X_{ik})$ where each X_{ij} is binary A discrete mixture model assumes that each component of the vector $X^(i)$ is independent, conditional upon an unknown class label $U_i \in \{1, ..., l\}$ a. write down the likelihood for one observation $X^{(1)}$, and then for n observations. What are the parameters to be estimated? $$

L() $$ b.