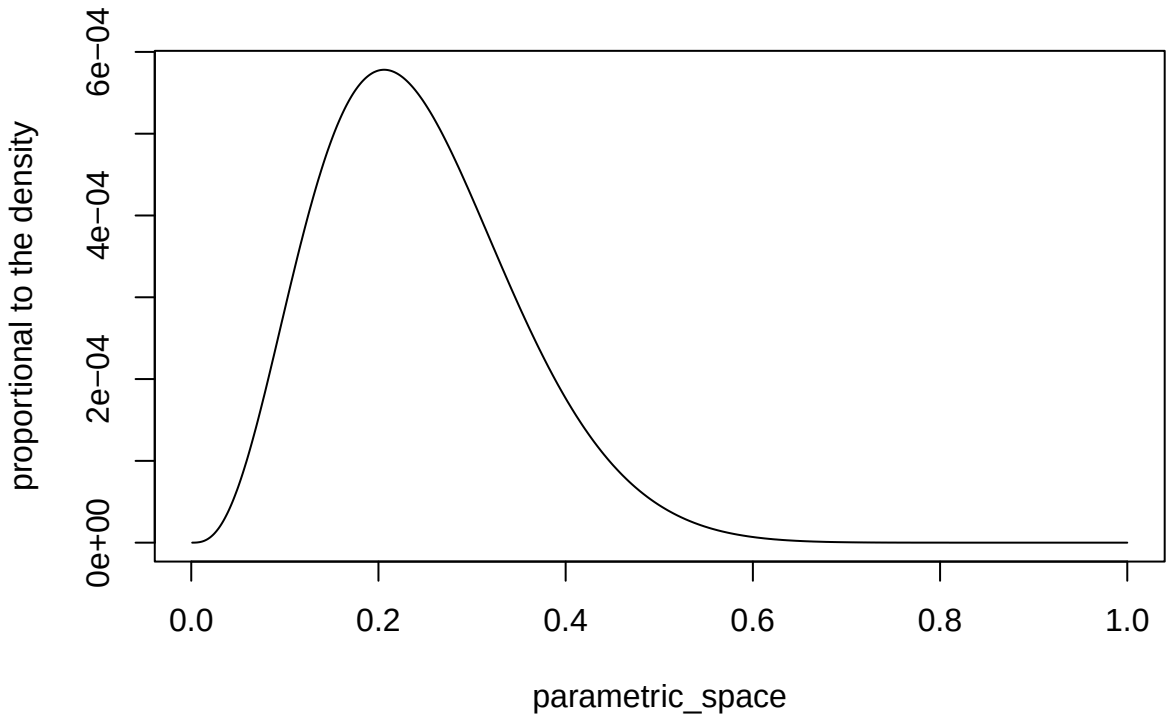
HW1_207

RafaelCatoiaPulgrossi

2023-04-06

Ex 1

```
posterior_prop <- function(theta){
  return(
    (theta^3)*(1-theta)^13+
      (theta^4)*(1-theta)^12+
      (theta^5)*(1-theta)^11
  )
}
parametric_space <- seq(0.001,1,0.001)

plot(parametric_space,posterior_prop(theta=parametric_space),
     type='l',ylab='proportional to the density')
```

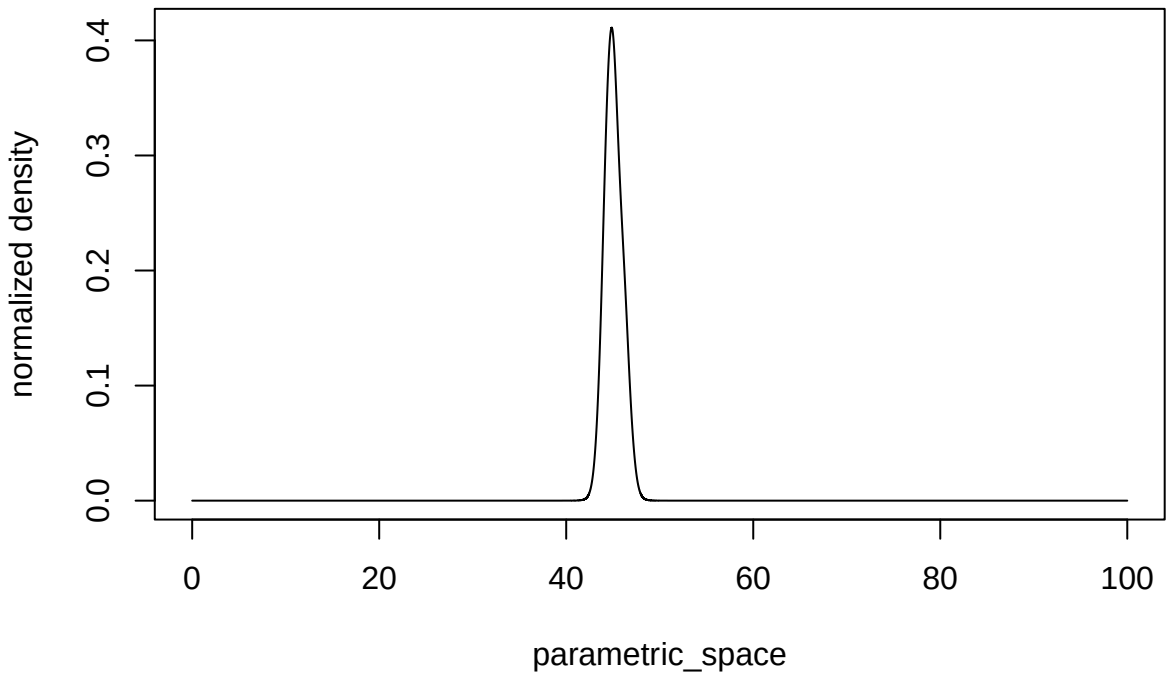


Ex 5

a

Here is the normalized density.

```
y <- c(43,44,45,46.5,47.5)
M=1000

parametric_space <- seq(0,100,1/M)

q_theta_given_y = function(x,theta){
  result <- 1
  for ( i in 1:length(x)){
    result = result * (1/(1+(x[i]-theta)^2))
  }
  return(result/100)
}


## unormalized density
unormalized_density <- q_theta_given_y(y,parametric_space)

## summing the heights of the bins
normalizing_constant <-  sum(unormalized_density/M)

#plotting the normalized density
plot(parametric_space,
     unormalized_density/normalizing_constant,
     type='l',
     ylab = 'normalized density')
```

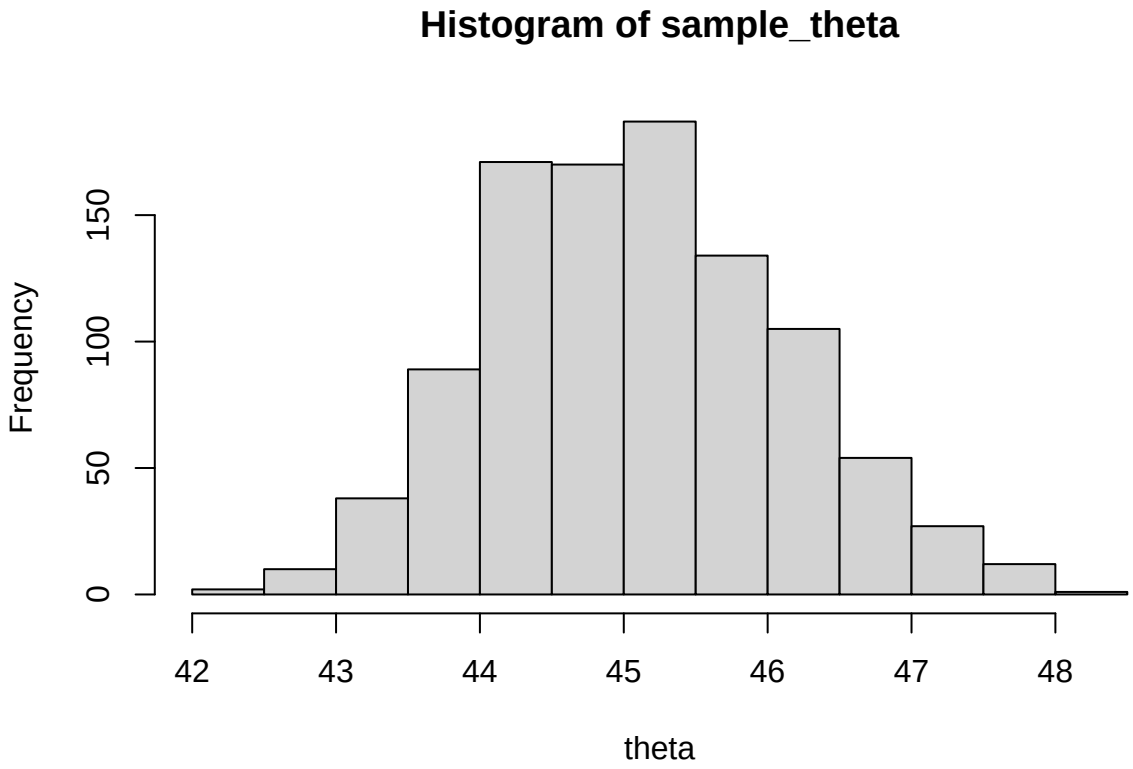



```
normalized_density <-  unormalized_density/normalizing_constant
```

b

Sampling from the normalized density

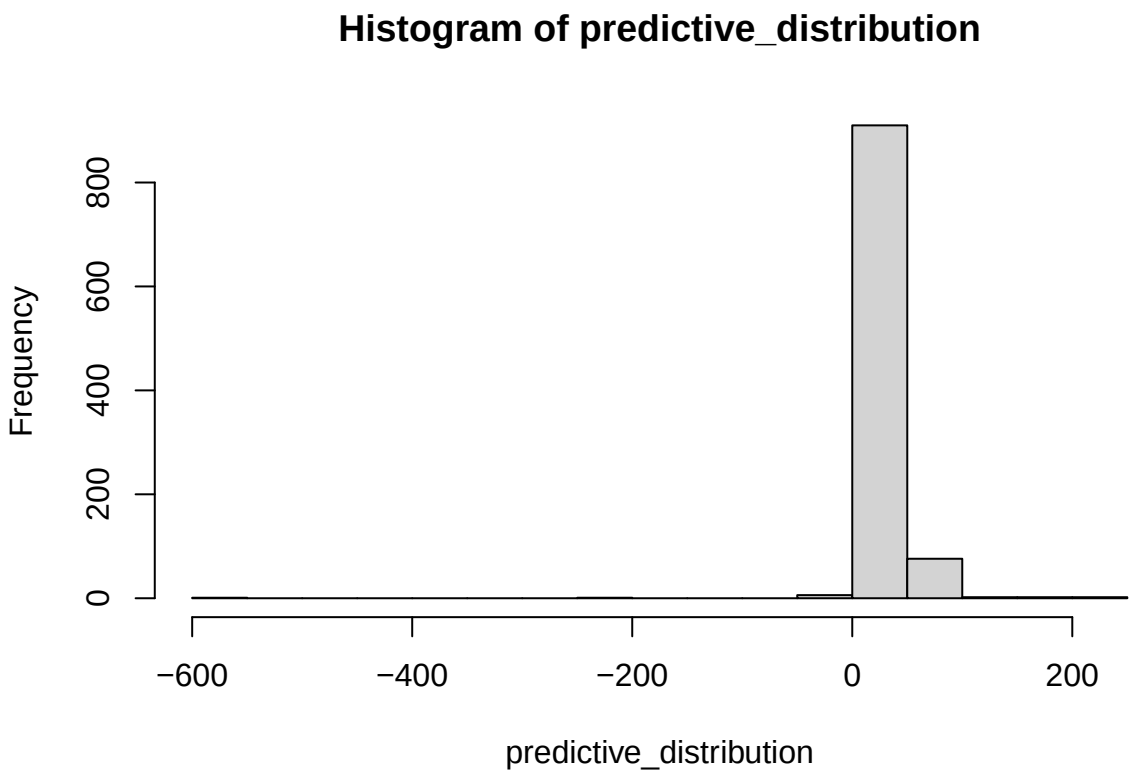
```
sample_theta <- sample(parametric_space,
                       prob = normalized_density,size = 1000)
hist(sample_theta,xlab='theta')
```



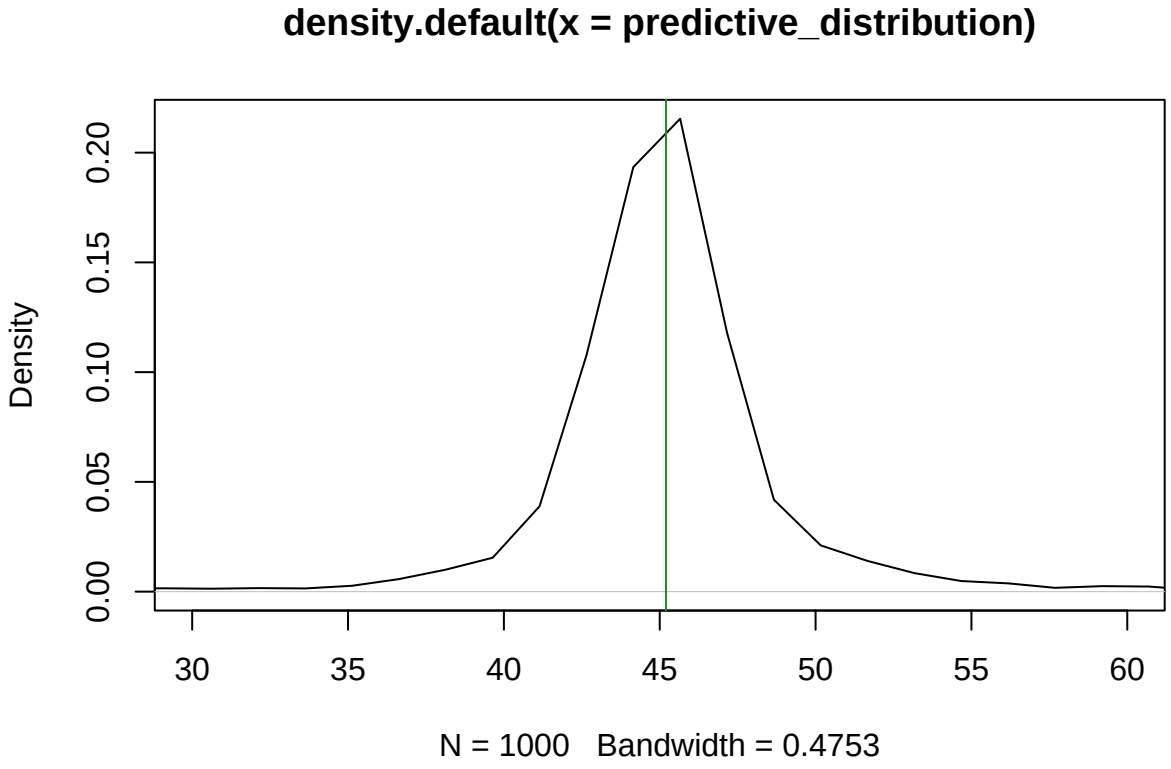
c

Histogram for the predictive distribution.

```
set.seed(57)
predictive_distribution <-
  rcauchy(1000,scale = 1,location = sample_theta)
hist(predictive_distribution)
```




```
#zoom
plot(density(predictive_distribution),xlim=c(30,60))
abline(v=mean(y),col='forestgreen')
```



```
mean(predictive_distribution) ; sd(predictive_distribution)
```

```
## [1] 44.7186
```

```
## [1] 24.22165
```

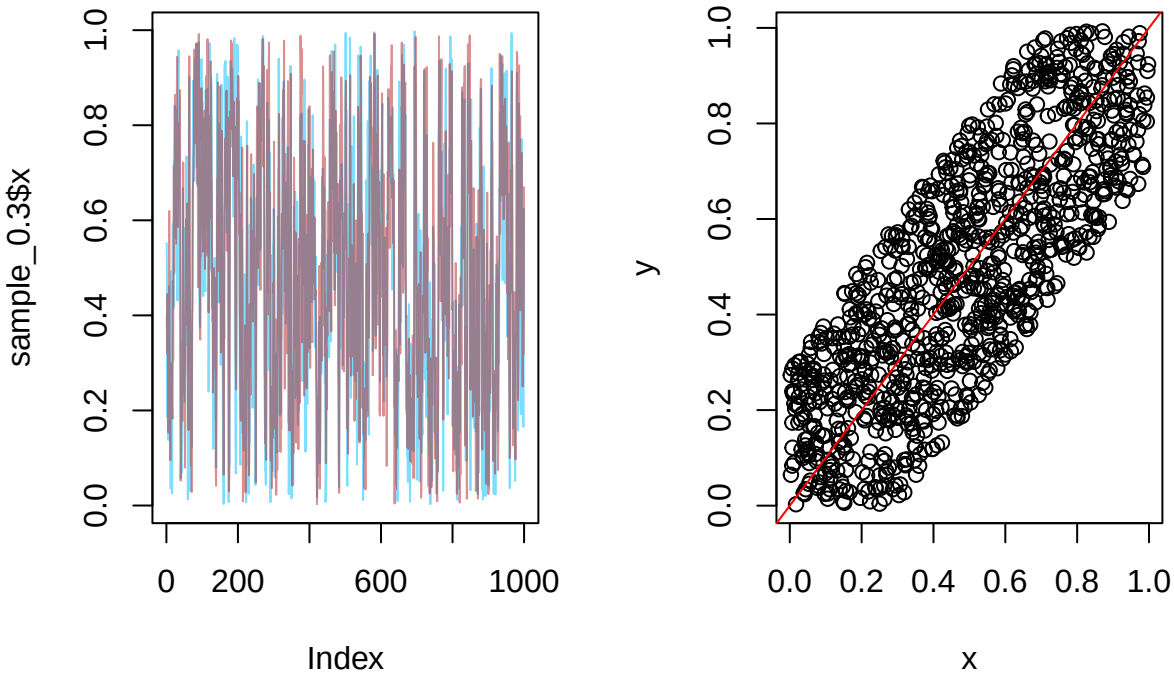
Ex 6

Here we are implementing the Gibbs Sampler.

```
x_given_y <- function(constant,y){
  a <- max(0,y-constant)
  b <- min(y+constant,1)
  return(runif(1,a,b))
}

y_given_x <- function(constant,x){
  a <- max(0,x-constant)
  b <- min(x+constant,1)
  return(runif(1,a,b))
}


gibbs_ex6 <- function(B=1000,constant){
  set.seed(1000)
  samples <- data.frame(x=NA,y=NA)
  new_x <- 0.5 #initial value for x
  for(i in 1:(B)){
    #calculating p(y|x=new_x)
    new_y <- y_given_x(x = new_x,constant = constant)
    #calculating p(x|y=new_y)
    new_x <- x_given_y(y = new_y,constant = constant)
    samples <-rbind(samples, c(new_x,new_y))
  }
  return(samples[-1,])
}

sample_0.3 <- gibbs_ex6(B = 1000,0.3)
sample_0.05 <- gibbs_ex6(B = 1000,0.05)
sample_0.01 <- gibbs_ex6(B = 1000,0.01)
```

For C=0.3

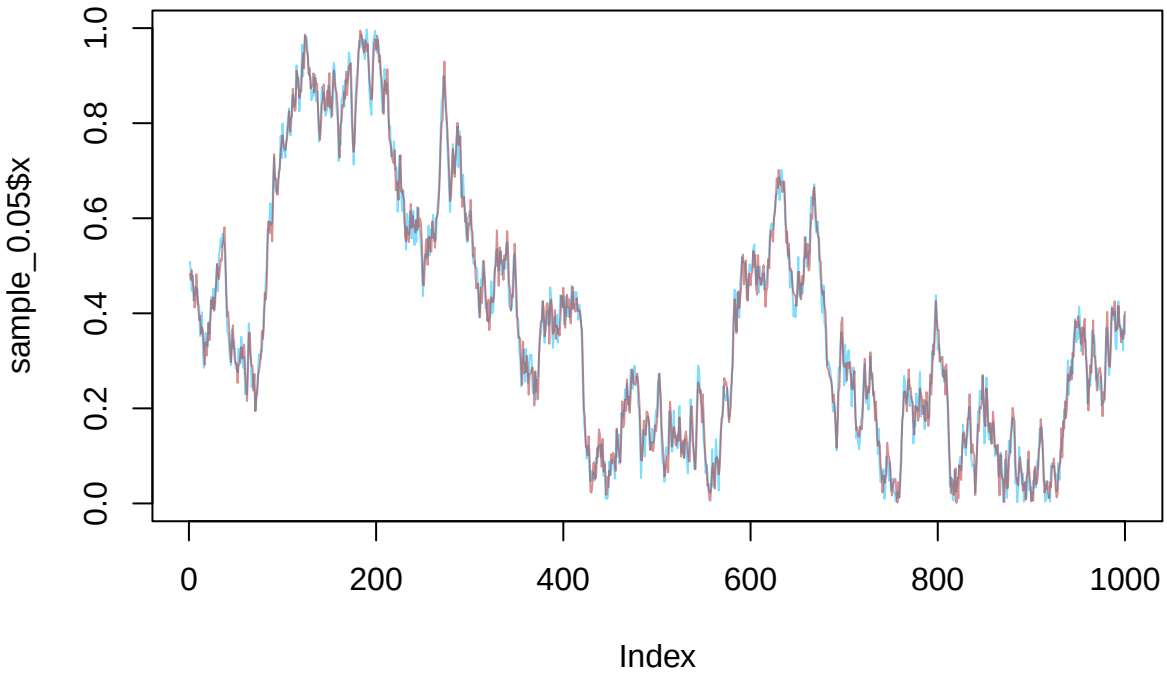
```
par(mfrow=c(1,2))
constant<-0.3
plot(sample_0.3$x,type='l',col=scales::alpha('deepskyblue',0.5))
lines(sample_0.3$y,type='l',col=scales::alpha('firebrick',0.5))
plot(sample_0.3)
abline(0,1,col='red')
```



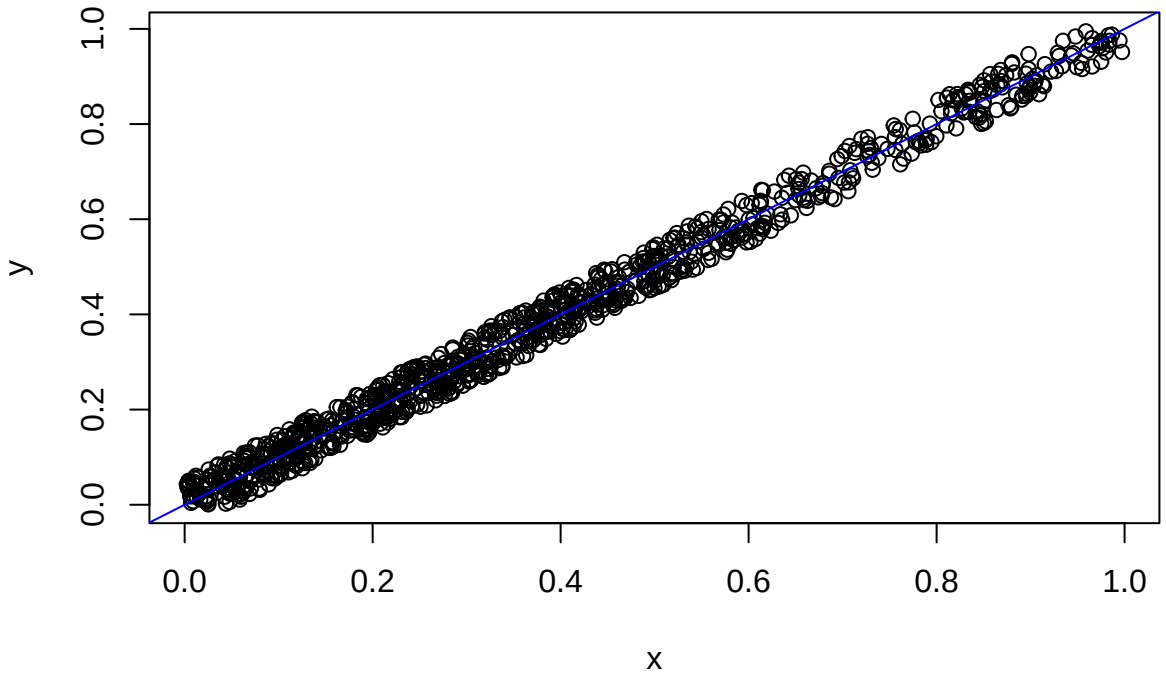

```
#acf(sample_0.3$x)
#acf(sample_0.3$y)
```

For C=0.05

```
constant<-0.05
plot(sample_0.05$x,type='l',col=scales::alpha('deepskyblue',0.5))
lines(sample_0.05$y,type='l',col=scales::alpha('firebrick',0.5))
```



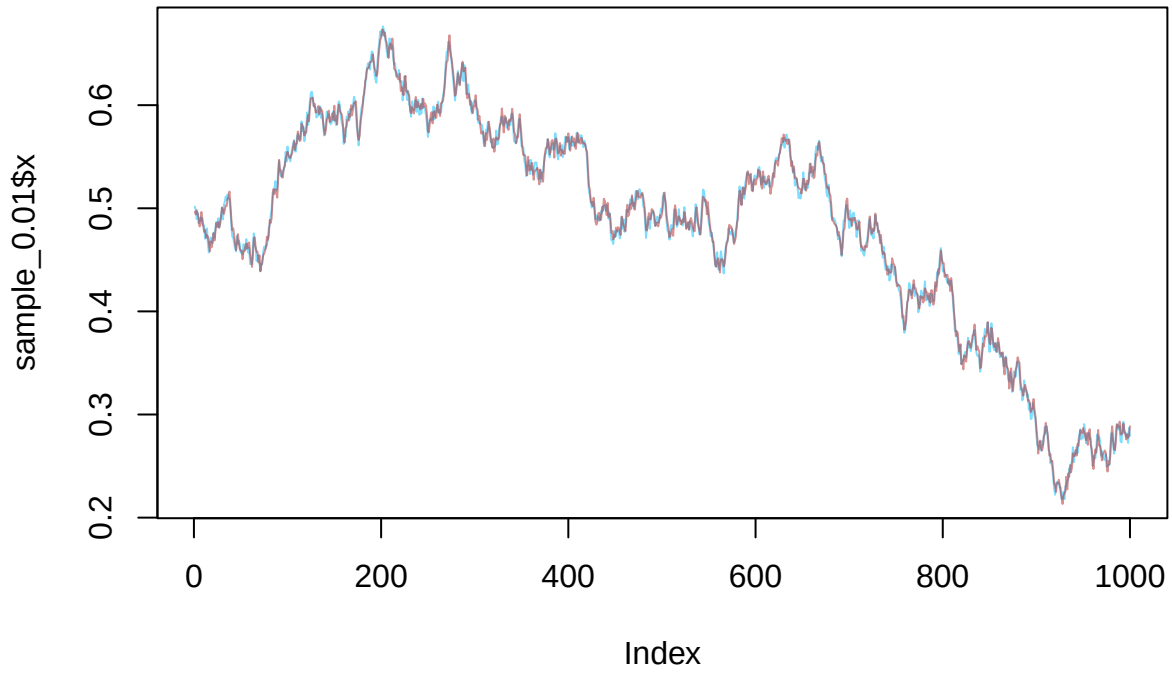

```
plot(sample_0.05)
abline(0,1,col='blue')
```



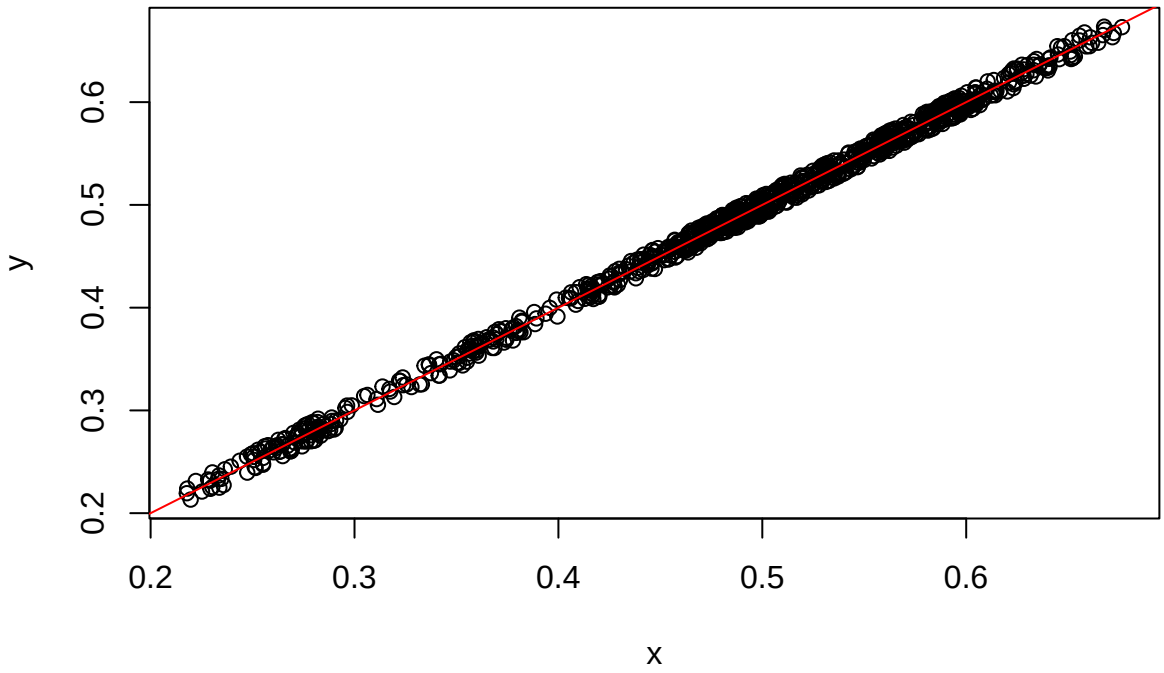

```
#acf(sample_0.05$x)
#acf(sample_0.05$y)
```

For C=0.01

```
constant<-0.01
plot(sample_0.01$x,type='l',col=scales::alpha('deepskyblue',0.5))
lines(sample_0.01$y,type='l',col=scales::alpha('firebrick',0.5))
```




```
plot(sample_0.01)
abline(0,1,col='red')
```




```
#acf(sample_0.01$x)
#acf(sample_0.01$y)
par(mfrow=c(1,1))
```

When C is small, the chain is very dependent on the past observations and as we can see on the last plot, it hasn't visited the entire parametric space yet.

Ex 8 - 3.4

```
nsim <- 10000
posterior_simulation <- data.frame(
  p0 = rbeta(nsim,40,636),
  p1 = rbeta(nsim,23,569)
)
```

Summary for the posterior sample of p_0 and p_1

```
posterior_simulation =
  posterior_simulation %>%
  mutate(odds_p0=p0/(1-p0),
         odds_p1=p1/(1-p1),
         odds_ratio = odds_p1/odds_p0)
```

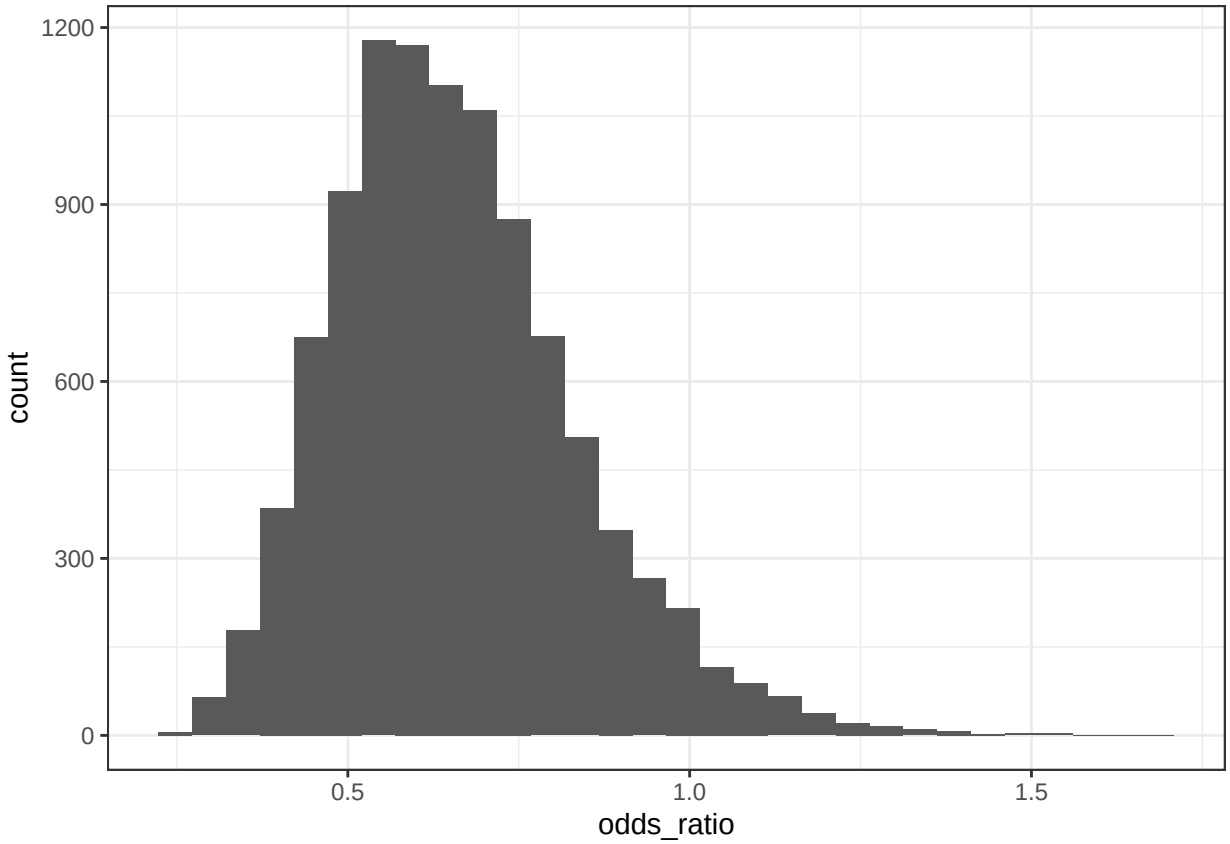
Summary for the posterior sample of odds ratio

```
posterior_simulation %>%
  summarise(
    Min = min(odds_ratio),
    Mean = mean(odds_ratio),
    Median = median(odds_ratio),
    SD = sd(odds_ratio),
    Max = max(odds_ratio)) %>%
  knitr::kable() %>% kableExtra::kable_styling()
```

Min	Mean	Median	SD	Max
0.252106	0.6583732	0.6367122	0.1774619	1.688666

```
posterior_simulation %>%
  ggplot(aes(x=odds_ratio))+
  geom_histogram() +
  theme_bw()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Calculating some posterior probabilities

```
y_bar_control = 39/674
y_bar_treatment = 22/680

#Posterior probability of p0 > p_control_MLE
sum(posterior_simulation$p0 > y_bar_control)/nsim
```

```
## [1] 0.5367
```

```
#Posterior probability of p0 > p_treat_MLE
sum(posterior_simulation$p1 > y_bar_treatment)/nsim
```

```
## [1] 0.7863
```

```
#Posterior probability of the odds ratio being smaller than 1
sum(posterior_simulation$odds_ratio < 1)/nsim
```

```
## [1] 0.9583
```

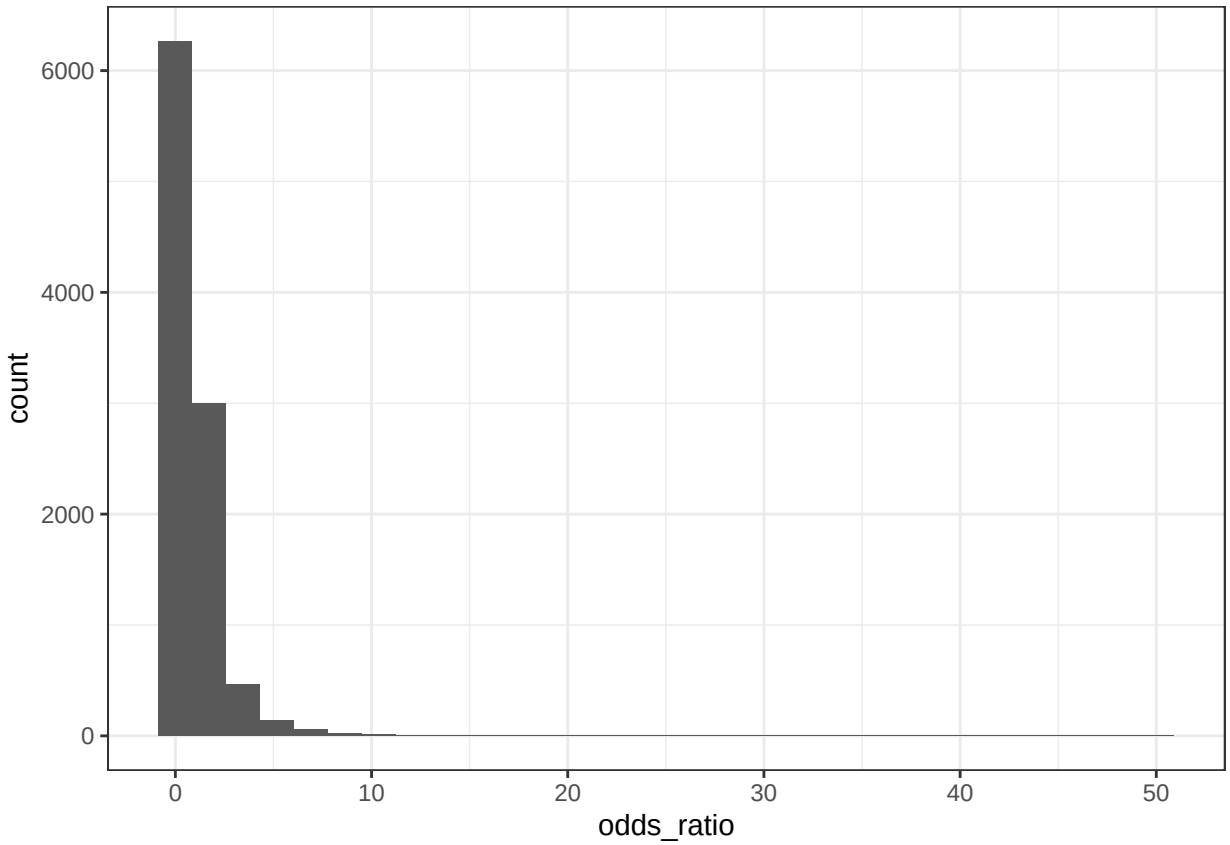
Min	Mean	Median	SD	Max
0.0015135	1.022259	0.6217633	1.451283	50.0575

Now changing the sample size

```
nsim <- 10000
posterior_simulation <- data.frame(
  #34 patients in each group
  p0 = rbeta(nsim,3,35),
  p1 = rbeta(nsim,2,35)
)


posterior_simulation =
  posterior_simulation %>%
  mutate(odds_p0=p0/(1-p0),
         odds_p1=p1/(1-p1),
         odds_ratio = odds_p1/odds_p0)

posterior_simulation %>%
  summarise(
    Min = min(odds_ratio),
    Mean = mean(odds_ratio),
    Median = median(odds_ratio),
    SD = sd(odds_ratio),
    Max = max(odds_ratio)) %>%
  knitr::kable() %>% kableExtra::kable_styling()
```

```
posterior_simulation %>%
  ggplot(aes(x=odds_ratio))+
  geom_histogram() +
  theme_bw()
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
y_bar_control = 2/34
y_bar_treatment = 1/34

#Posterior probability
sum(posterior_simulation$p0 > y_bar_control)/nsim
```

```
## [1] 0.6288
```

```
#Posterior probability of p0 > p_treat_MLE
sum(posterior_simulation$p1 > y_bar_treatment)/nsim
```

```
## [1] 0.7147
```

```
#Posterior probability of the odds ratio being smaller than 1
sum(posterior_simulation$odds_ratio < 1)/nsim
```

```
## [1] 0.6801
```

Ex 9

a)

Lets use as non-informative prior $p(\mu, \sigma) \propto \frac{1}{\sigma^2}$, our data-sampling distribution is given by $p(y|\mu, \sigma^2) \sim N(\mu, \sigma^2)$.

Thus, our posterior distribution is going to be:

$$p(\mu, \sigma^2) = p(\mu|\sigma^2, y)p(\sigma^2|y)$$

being $\mu|\sigma^2, y \sim N(\bar{y}, \frac{\sigma^2}{n})$ and $\sigma^2|y \sim Inv\chi^2(n-1, s^2)$

```
y = c(10,10,12,11,9)
ybar = mean(y)
n = length(y)
S2 = var(y)


set.seed(1234)
generate_posterior_untruncated <- function(y){
  ybar = mean(y)
  n = length(y)
  S2 = var(y)
  sigma2 <- (n-1)*S2/rchisq(n = 1,df = (n-1))
  mu <- rnorm(1,mean=ybar,sd=sqrt(sigma2)/sqrt(n))
  return(data.frame(mu,sigma2))
}
```

```
#Calculating the posterior density for the untruncated likelihood

dchisqncp <- function(x, df, ncp){
  (df/2)^(df/2)/gamma(df/2) * sqrt(ncp)^df * x^(-df/2 - 1) * exp(-df*ncp/(2*x))
}

density_posterior_untruncated <- function(mu,sigma2,y){
  ybar = mean(y)
  S2 = var(y)
  n=length(y)
  return(
    dnorm(mu,mean = ybar,sd=sqrt(sigma2)/sqrt(n))*
      dchisqncp(sigma2,df = n-1,ncp = S2)
  )
}
```

b)

Now our likelihood is different, so does he posterior which is given by: $p(\mu, \sigma^2|y) \propto \frac{1}{\sigma^2} \prod_{i=1}^{n} \left[\Phi(\frac{y_i - \mu + 0.5}{\sigma}) - \Phi(\frac{y_i + \mu - 0.5}{\sigma})\right]$

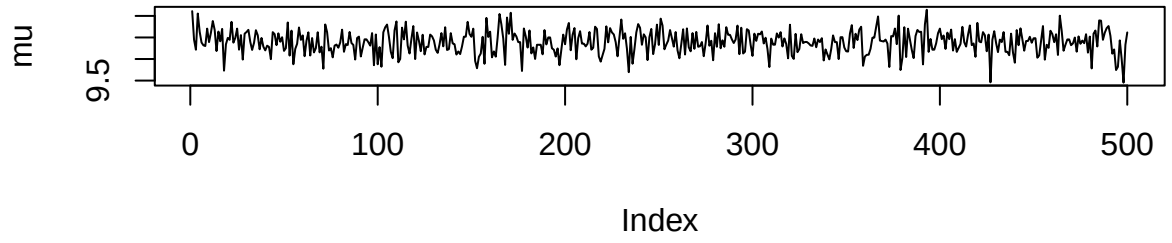
```
#proportional posterior density
prop_density_post_truncated <- function(y,sigma2,mu){
  prop_density <- 0
  for(i in 1:length(y)){
    prop_density =  prop_density +
      log(
        (pnorm((y[i]+0.5-mu)/sqrt(sigma2),mean = 0,sd = 1,)  -
         pnorm((y[i]-0.5-mu)/sqrt(sigma2),mean = 0,sd = 1))
         )
  }
  return(exp(prop_density))
}



#### MCMC ----------------------------------------------------
## Metropolis Hastings ---------------------------------------
```

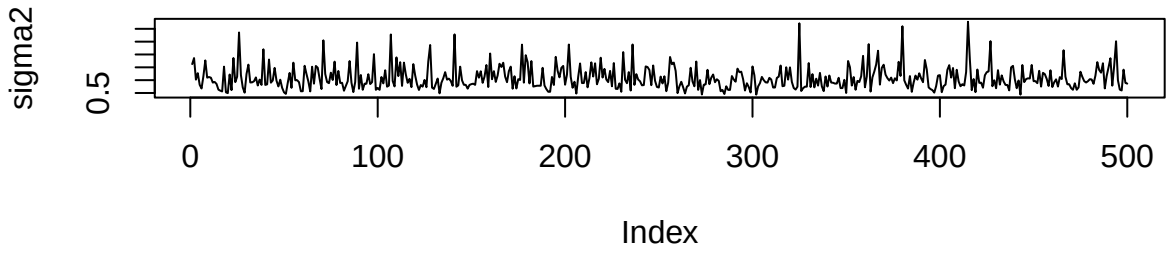
```r
# candidate will be the posterior without truncation

n_it <- 500 #number of samples to be generated
step <- 10
burn_in = 500
total_samples = burn_in + n_it*step
init_value <- generate_posterior_untruncated(y=y) # initial value of theta
accepted=0
post_samp_truncated <-
  data.frame(mu=rep(NA,n_it),
             sigma2=rep(NA,n_it))

## Let's use as candidate the posterior
## pretending unrounded measurements
post_samp_truncated[1,]<-init_value
for (i in 2:total_samples){
  candidate <- generate_posterior_untruncated(y=y)

  ## Calculating r
  #numerators
  num1 = prop_density_post_truncated(
    y = y,
    sigma2 = candidate$sigma2,
    mu = candidate$mu)

  num2 = density_posterior_untruncated(
    y = y,
    mu=candidate$mu,
    sigma2 = candidate$sigma2
  )

  #denominators
  den1 = prop_density_post_truncated(
    y = y,
    sigma2 = post_samp_truncated[i-1,]$sigma2,
    mu = post_samp_truncated[i-1,]$mu)

  den2 = density_posterior_untruncated(
    y = y,
    mu=post_samp_truncated[i-1,]$mu,
    sigma2 = post_samp_truncated[i-1,]$sigma2
  )

  ## r
  ## using the log since the densities can be very small
  r=min(1,exp(log(num1)+log(num2)-(log(den1)+log(den2))))
  #r=min(1,exp(log(num1)-log(den1)))

  if(runif(1)>r){
    # rejected! we will start the new iteration with the same point
    post_samp_truncated[i,] <- post_samp_truncated[i-1,]
  } else {
    # accepted! we will move our chain to the candidate point and start the
```

```
    # next iteration from this point
    post_samp_truncated[i,] <- candidate
    accepted<-accepted+1
  }
}

## acceptance rate =
accepted/total_samples
```

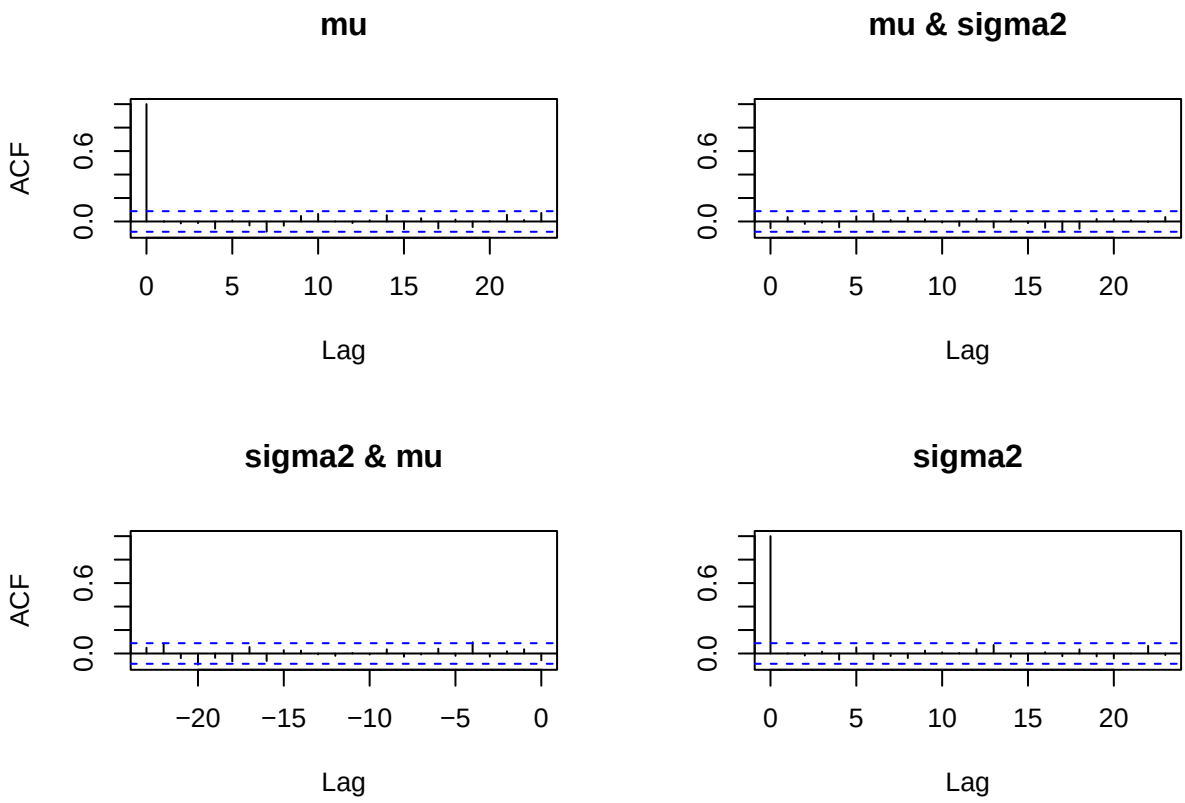
```
## [1] 0.3885455
```

```
## brushing the sample
post_sample_clean <- post_samp_truncated[-c(1:burn_in),][seq(1,total_samples-burn_in,by=step),]

par(mfrow=c(2,1))
plot(post_sample_clean$mu,type='l',ylab='mu')
plot(post_sample_clean$sigma2,type='l',ylab='sigma2')
```






```
par(mfrow=c(1,1))
acf(post_sample_clean)
```



c)

```
sample_posterior_untruncated <- data.frame(
  mu=rep(NA,n_it),
  sigma2=rep(NA,n_it)
)

for(i in 1:n_it){
  sample_posterior_untruncated[i,] <- generate_posterior_untruncated(y=y)
}

posterior_samples <- data.frame(
  mu_untruncated = sample_posterior_untruncated$mu,
  mu_truncated = post_sample_clean$mu,
  sigma2_untruncated = sample_posterior_untruncated$sigma2,
  sigma2_truncated = post_sample_clean$sigma2
)

data.frame(
  Mean =  posterior_samples %>%
    summarise(Untruncated = mean(mu_untruncated),
              Truncated = mean(mu_truncated)) %>% t(),
  SD =posterior_samples %>%
    summarise(Untruncated = sd(mu_untruncated),
              Truncated = sd(mu_truncated)) %>% t(),
  Percentile_0.5 =  posterior_samples %>%
    summarise(Untruncated = quantile(mu_untruncated,0.05),
              Truncated = quantile(mu_truncated,0.1)) %>% t(),
```

Table 1: For mu

	Mean	SD	Percentile_0.5	Percentile_1	Percentile_25	Percentile_50
Untruncated	10.39151	0.7224377	9.28141	9.598535	9.984416	10.41365
Truncated	10.39554	0.2725610	10.04506	9.939021	10.218478	10.39331

Table 2: For sigma

	Mean	SD	Percentile_0.5	Percentile_1	Percentile_25	Percentile_50
Untruncated	2.549833	3.4040734	0.4983951	0.6439316	0.9929236	1.5914511
Truncated	1.085370	0.4549925	0.6252114	0.5610709	0.7750116	0.9722241

```
  Percentile_1 = posterior_samples %>%
    summarise(Untruncated = quantile(mu_untruncated,0.1),
              Truncated = quantile(mu_truncated,0.05)) %>% t(),
  Percentile_25 =   posterior_samples %>%
    summarise(Untruncated = quantile(mu_untruncated,0.25),
              Truncated = quantile(mu_truncated,0.25)) %>% t(),
  Percentile_50 = posterior_samples %>%
    summarise(Untruncated = quantile(mu_untruncated,0.5),
              Truncated = quantile(mu_truncated,0.5)) %>% t()
) %>% knitr::kable(caption = 'For mu') %>% kableExtra::kable_styling()
```

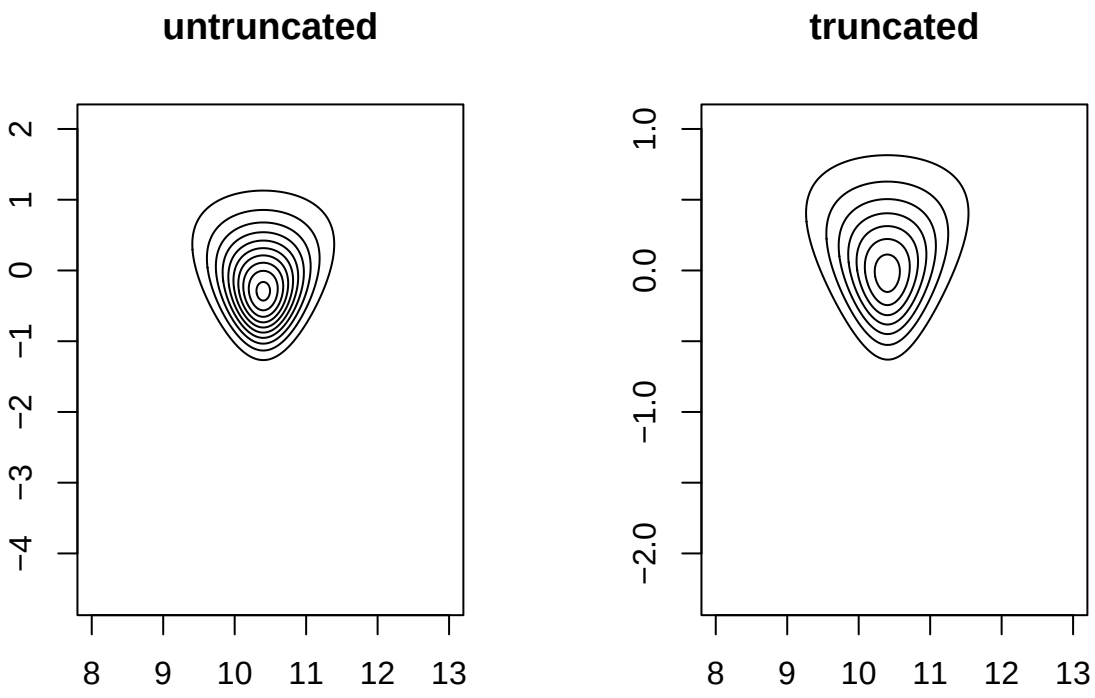
```
data.frame(
  Mean =  posterior_samples %>%
    summarise(Untruncated = mean(sigma2_untruncated),
              Truncated = mean(sigma2_truncated)) %>% t(),
  SD =posterior_samples %>%
    summarise(Untruncated = sd(sigma2_untruncated),
              Truncated = sd(sigma2_truncated)) %>% t(),
  Percentile_0.5 =  posterior_samples %>%
    summarise(Untruncated = quantile(sigma2_untruncated,0.05),
              Truncated = quantile(sigma2_truncated,0.1)) %>% t(),
  Percentile_1 = posterior_samples %>%
    summarise(Untruncated = quantile(sigma2_untruncated,0.1),
              Truncated = quantile(sigma2_truncated,0.05)) %>% t(),
  Percentile_25 =   posterior_samples %>%
    summarise(Untruncated = quantile(sigma2_untruncated,0.25),
              Truncated = quantile(sigma2_truncated,0.25)) %>% t(),
  Percentile_50 = posterior_samples %>%
    summarise(Untruncated = quantile(sigma2_untruncated,0.5),
              Truncated = quantile(sigma2_truncated,0.5)) %>% t()
) %>% knitr::kable(caption = 'For sigma') %>% kableExtra::kable_styling()
```

Now,for the contour plot we are going to use the $log(\sigma^2)$ in order to have a better visualization.

```
#density untruncated
mu_grid <- seq(8,13,0.01)
sigma2_grid <- seq(0.01,8,0.01)
density_values <- matrix(NA,nrow=length(mu_grid),ncol=length(sigma2_grid))

for(i in 1:length(mu_grid)){
  for(j in 1:length(sigma2_grid)){
```

```
  #cat(paste(i,j),'\n')
    density_values[i,j] <- density_posterior_untruncated(
      y = y,
      mu = mu_grid[i],
      sigma2 = sigma2_grid[j]
    )
  }
}
```

```
#density untruncated
density_values_truncated <- matrix(NA,nrow=length(mu_grid),ncol=length(sigma2_grid))

for(i in 1:length(mu_grid)){
  for(j in 1:length(sigma2_grid)){
  #cat(paste(i,j),'\n')
    density_values_truncated[i,j] <- prop_density_post_truncated(
      y = y,
      mu = mu_grid[i],
      sigma2 = sigma2_grid[j]
    )
  }
}

#this one is not normalized! but since we would divide by a constant we can still use the shape to see
par(mfrow=c(1,2))
contour(
  x = mu_grid,
  y = log(sigma2_grid),
  z = density_values,
  main='untruncated',
  drawlabels = F
  )

contour(
  x = mu_grid,
  y = log(sqrt(sigma2_grid)),
  z = density_values_truncated,
  drawlabels = F,main='truncated',
  )
```




```
par(mfrow=c(1,1))
```

d)

For each i, z_i is the the unrounded value of y_i. Therefore, we can generate a sample of z_i conditional on y_i by generating from a truncated normal distribution between $y_i - 5$ and $y_i + 5$ with parameters from our posterior sample.

```
z_samples <- matrix(NA, n_it, 5)
for (i in 1:5) {
  z_samples[, i] <- truncnorm::rtruncnorm(
    n_it, a = y[i] - 0.5, b = y[i] + 0.5,
    mean = sample_posterior_untruncated$mu, sd = sample_posterior_untruncated$sigma2)
}

((z_samples[, 2] - z_samples[, 1])^2) %>% mean()
```

```
## [1] 0.1491312
```

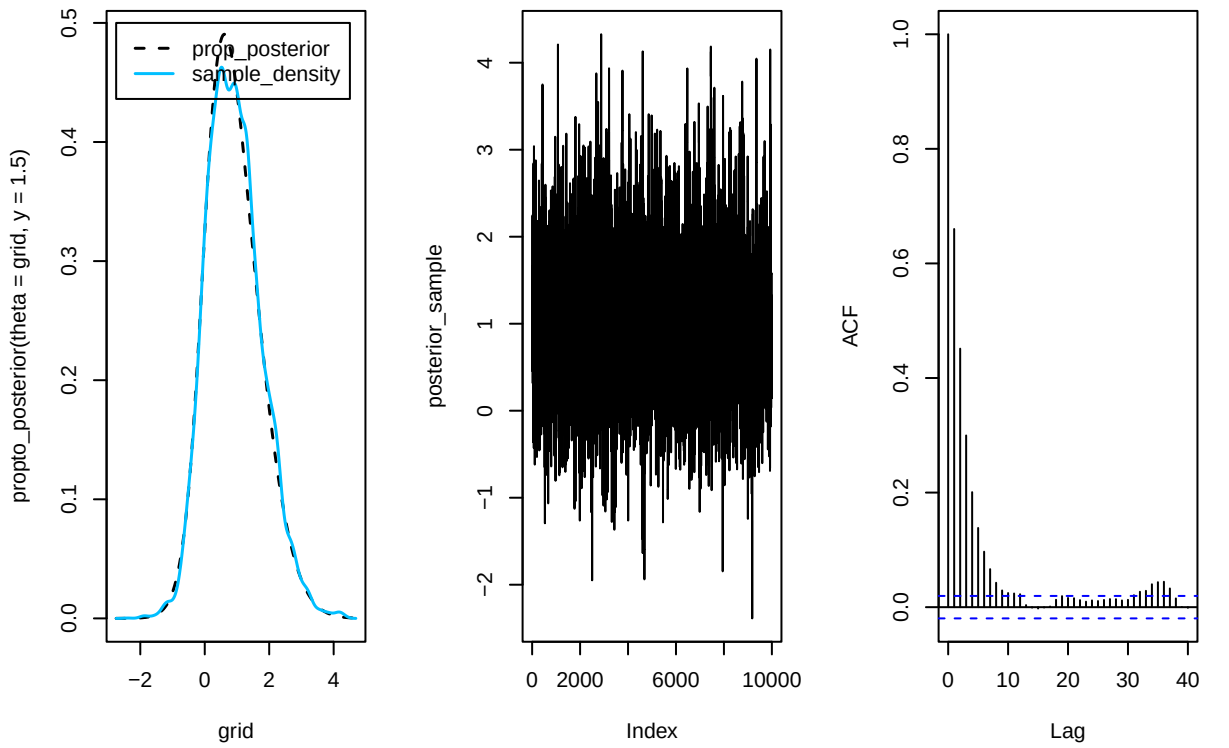
Ex 10

```
# proportional to the posterior distribution that we
# are going to draw samples from
# Witout using log
propto_posterior <- function(y=1.5,theta){
  numerator <- exp(-(1/2)*(y-theta)^2)
  denominator <- (1+theta^2)
  return(numerator/denominator)
}

# Using log
propto_posterior <- function(y=1.5,theta){
```

```
 numerator <- -(1/2)*(y-theta)^2
 denominator <- -log(1+theta^2)
 return(exp(numerator+denominator))
}

randWalk_MH_diagnostics <- function(posterior_sample){
  lindseys_density  <- density(posterior_sample)
  a <- round(min(lindseys_density$x),3) ; b <-round(max(lindseys_density$x),3)
  grid<- seq(a,b,0.001)
  par(mfrow=c(1,3))
  plot(grid,propto_posterior(theta = grid,y=1.5),type='l',lty=2,lwd=1.5)
  lines(density(posterior_sample),type='l',col='deepskyblue',lwd=1.5)
  legend(a,0.5,legend = c('prop_posterior','sample_density'),
         col=c('black','deepskyblue'),lwd = c(1.5,1.5), lty=c(2,1))
  plot(posterior_sample,type = 'l')
  acf(posterior_sample)
  par(mfrow=c(1,3))
}


n_it <- 10000 #number of samples to be generated
init_value <- 0.5 # initial value of theta
proposal_dist <- function(theta){rnorm(1,theta,sd=2)} # we are going to use the normal distribution as
post_sample <- rep(NA,n_it) #creating the vector that will store the samples
post_sample[1]=init_value #the first draw of the posterior is the initial value
accepted=0 # counting the accepted values to see the acceptance rate
# initializing the iteration
for (i in 2:n_it){
  candidate <- proposal_dist(post_sample[i-1])

  #calculating acceptance probability of the current point
  r = min(1,propto_posterior(theta = candidate) /
            propto_posterior(theta = post_sample[i-1]))

  #accepting the candidates with probability r
  if(runif(1)>r){
    # if not accepted, we will start the new iteration with the same point
    post_sample[i] <- post_sample[i-1]
  } else {
    # if accepted, we will move our chain to the candidate point and start the
    # next iteration from this point
    post_sample[i] <- candidate
    accepted<-accepted+1
  }
}

accepted/n_it
```

```
## [1] 0.4294
```

```
randWalk_MH_diagnostics(post_sample)
```




```
summary(post_sample)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -2.3869  0.3080  0.8580  0.9182  1.4414  4.3269
```

```
n_it <- 1000 #number of samples to be generated
init_value <- 0 # initial value of theta
step <- 5
burn_in = 10000
total_samples = burn_in + n_it*step
# we are going to use the normal distribution as proposal
proposal_dist <- function(theta){rnorm(1,theta,sd=2)}
#creating the vector that will store the samples
post_sample <- rep(NA,total_samples)
#the first draw of the posterior is the initial value
post_sample[1]=init_value
accepted=0 # counting the accepted values to see the acceptance rate

# initializing the iteration
for (i in 2:total_samples){
  candidate <- proposal_dist(post_sample[i-1])

  #calculating acceptance probability of the current point
  r = min(1,propto_posterior(theta = candidate) /
            propto_posterior(theta = post_sample[i-1]))

  #accepting the candidates with probability r
  if(runif(1)>r){
    # if not accepted,
```

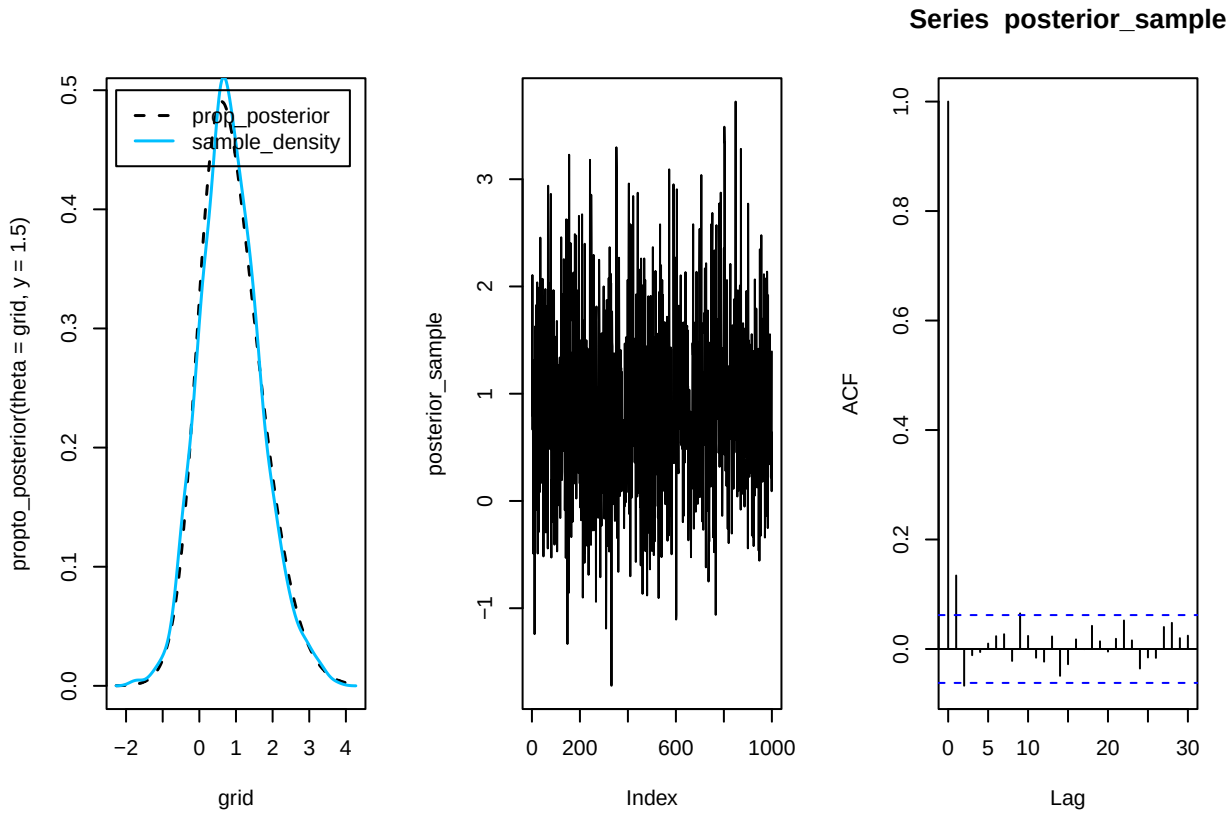
```
    # we will start the new iteration with the same point
    post_sample[i] <- post_sample[i-1]
  } else {
    # if accepted,
    # we will move our chain to the candidate point and start the
    # next iteration from this point
    post_sample[i] <- candidate
    accepted<-accepted+1
  }
}

#acceptance rate
accepted/total_samples
```

```
## [1] 0.4358
```

```
post_sample_clean <- post_sample[-c(1:burn_in)][seq(1,length(post_sample)-burn_in,step)]

#verifying if the length of the draw matches with the desired number
post_sample_clean %>% length()
```

```
## [1] 1000
```

```
randWalk_MH_diagnostics(post_sample_clean)
```




```
post_sample_clean %>% summary()
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -1.7224  0.3063  0.8010  0.8652  1.3884  3.7239
```

```
post_sample_clean %>% sd()
```

```
## [1] 0.8282481
```