Lab4,Group15

Naveen Gabriel ,Sridhar Adhikarla

2019-02-21

Contents

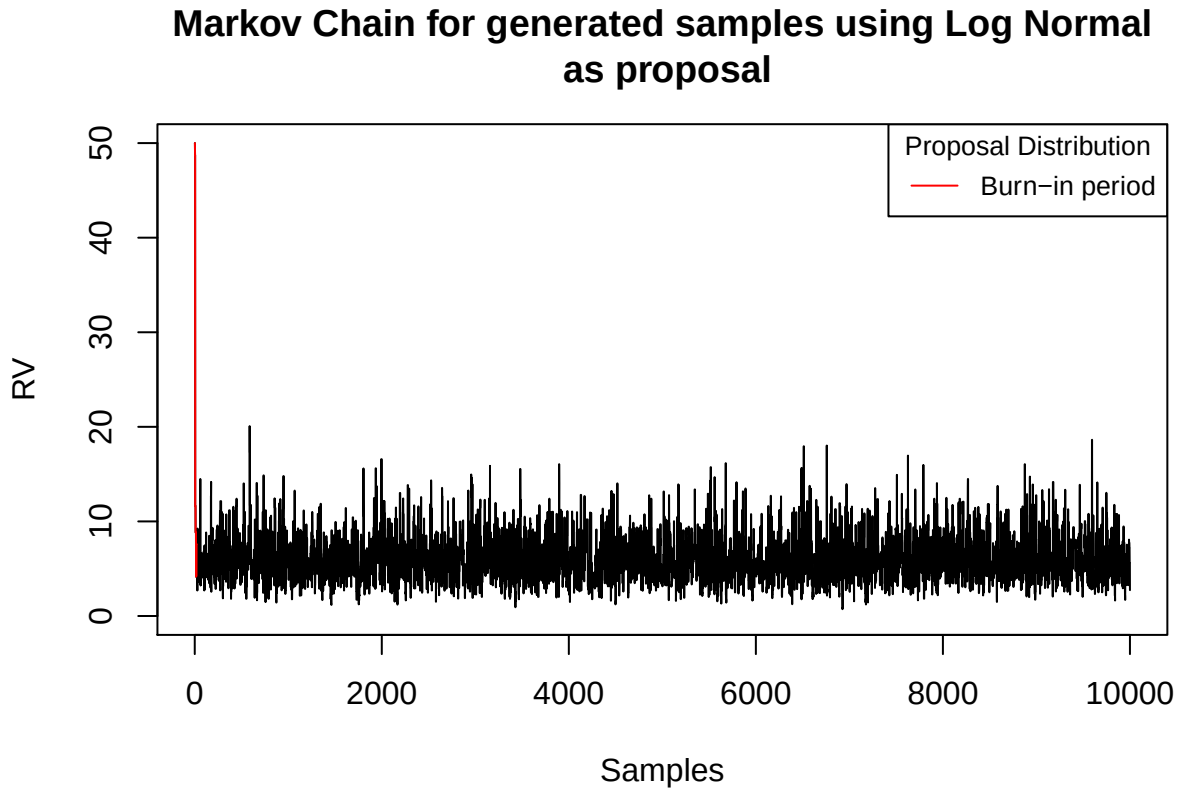
1 Metropolis Hasting Algorithm

1.1 Generate samples from distribution x^5e^{-x} by using proposal distribution as log{normal LN(Xt;1)

We took log normal as our proposal distribution to generate samples for our desired distribution. From the markov chain, the burn in period seems to be first 20 variables which is represented by red line, which represent the non equilibrium state. Post that the markov chain seems to be reaching in equilibrium, moving in the range from 3 to 4.

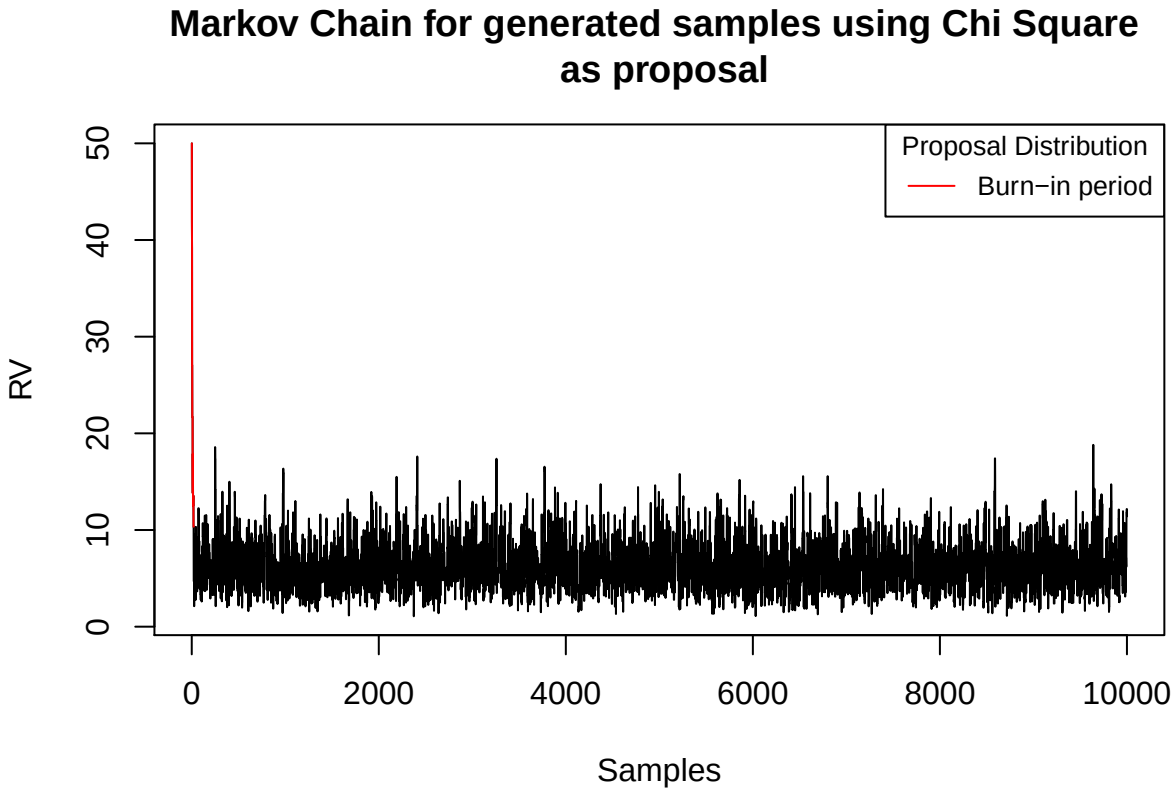
```
target_dist <- function(x){
     if(x<0){
         return(0)}
     else {
         return((x^5)*exp(-x))
  }
}

#Proposal Distribution
proposed_dist  <- function(rv){
    rlnorm(1, meanlog = log(rv), sdlog = 1)  #log normal
}

metropolis_algorithm <- function(samp,rand_no) {
    curr_x <- rand_no                #Current random value
    post_val <- rep(0,times = samp)  #Number of Posterior value which is accepted.
    prop_val <- c()                  #Generate values from the proposal distribution.
    cnt <- 1

#Iterate or generate 2000 points for target distribution using proposal distribution
     while(cnt <= samp){

#Generate a new value from proposal distribution based on current value and sd 1.
#Basically move to the near point. it is kind of markov move which is dependent on
#current value.

#Generate next RV based on the current RV.
        prop_x <- proposed_dist(curr_x)
        U <- runif(1, 0, 1)

#posterior <- likelyhood* prior
        posterior_prop <- target_dist(prop_x)*dlnorm(curr_x,mean = log(prop_x),sd = 1)
        posterior_curr <- target_dist(curr_x)*dlnorm(prop_x,mean = log(curr_x),sd = 1)

        ifelse(U < min(1,(posterior_prop/posterior_curr)),post_val[cnt] <- prop_x,
               post_val[cnt] <- curr_x)

        curr_x <- post_val[cnt]
        prop_val[cnt] <- prop_x

        cnt <- cnt+1

    }
return(list(post_val,prop_val))
```

```
}

val1 <- metropolis_algorithm(10000,50)

{plot(1:10000,y=val1[[1]],type="l",xlab="Samples",ylab="RV",col="black",ylim=c(0,50))
 lines(x=1:20,y=val1[[1]][1:20],col="red")
title(main = "Markov Chain for generated samples using Log Normal\n as proposal")
legend("topright",legend=c("Burn-in period"),
col=c("red"), lty=1:2, cex=0.8,title="Proposal Distribution")
}
```



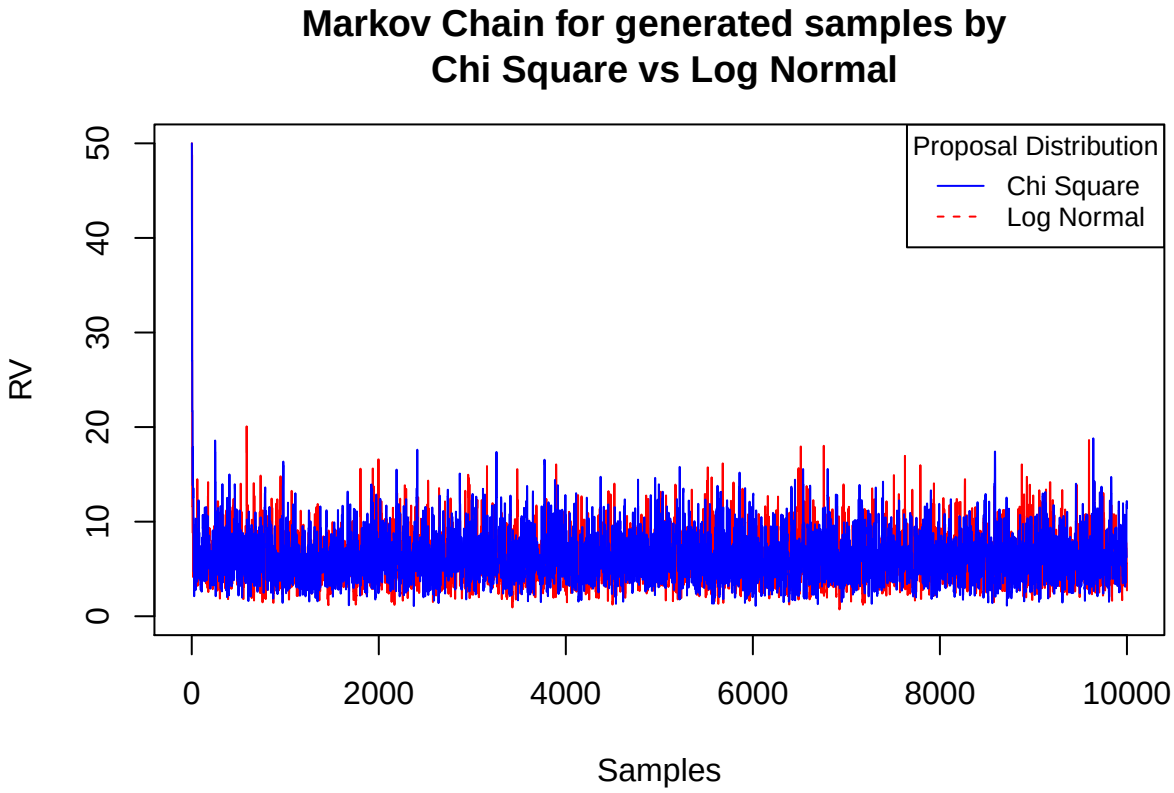
1.2 Chi Square distribution as proposal distribution

We took Chi Square as our proposal distribution to generate samples for our desired distribution. From the markov chain, the burn in period seems to be first 15 variables which is represented by red line, which represent the non equilibrium state. Post that the markov chain seems to be reaching in equilibrium, moving in the range from 4 to 6.

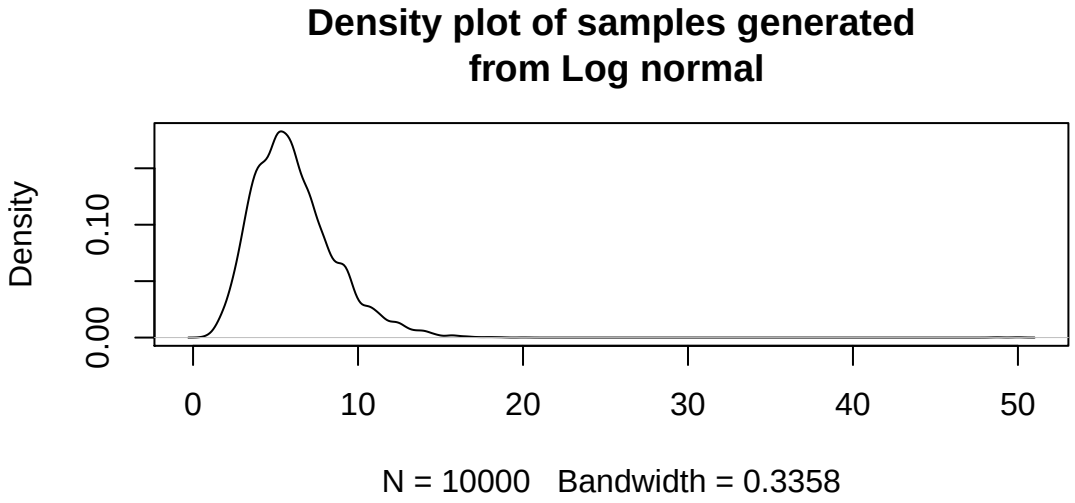


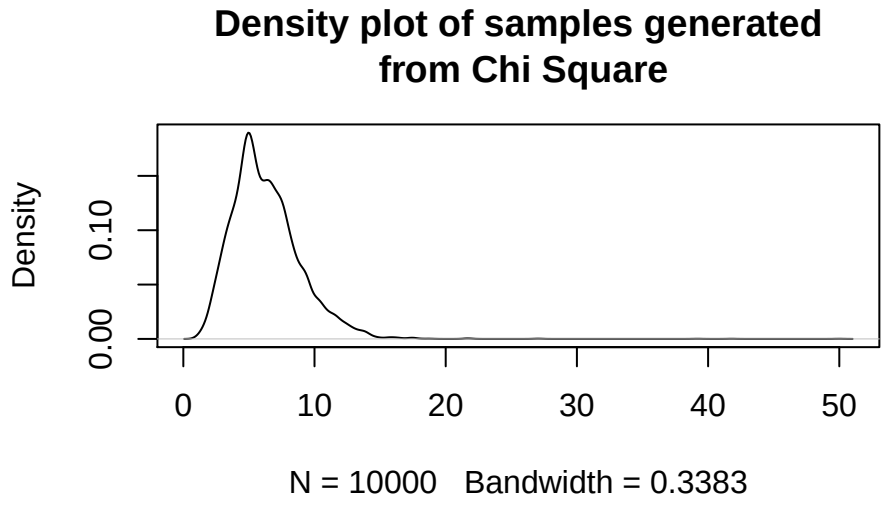
1.3 Comparison

From the comparison of 2 markov chain, the samples generated by Chi Square seems to be converging better than Log normal.



Below is the two density plot samples generated by log normal and chisquare



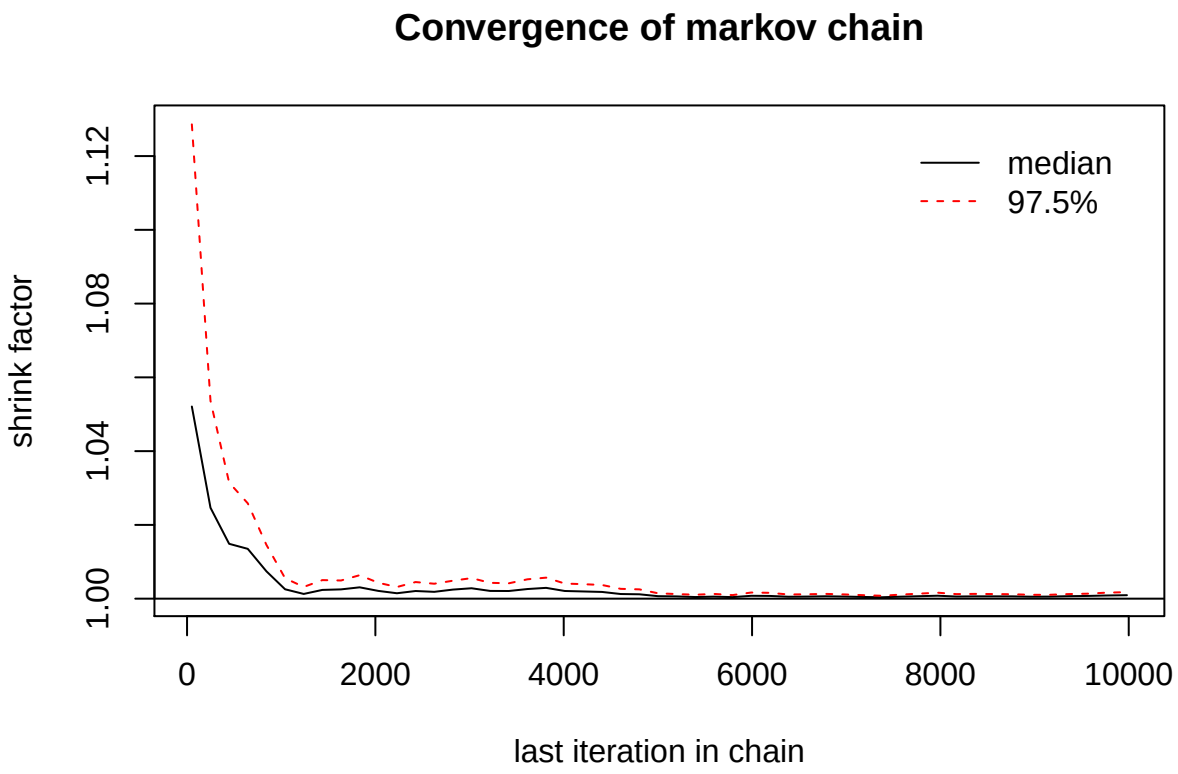


From the above two plot, it seems the chi square is more accurately representing our function than the log normal.

1.4 Generate 10 MCMC sequences using Chi Square as proposal function and use Gelman-Rubin method to analyze convergence.

Using Gelman-Rubin as convergence method , we get 'potential scale reduction factor'. Approximate convergence is diagnosed when the upper limit is close to 1. In our case it is 1. Hence we can assume that abour markov chain does converges.

```
Potential scale reduction factors:

     Point est. Upper C.I.
[1,]          1          1
```



1.5 Estimate $\int_0^\infty x.f(x)$

We can estimate the mean as:

$$E[x] = \int xf(x)dx \approx \frac{1}{j-i}\sum_{t=i+1}^{j} x_t$$

where j-i is the total number of iterations. Basically we are calculating mean.

```
Mean of samples by Log normal: 6.110792
```

```
Mean of samples by Chi Sqaure: 6.300467
```

On manually doing the integral of $\int_0^\infty x.f(x)$, we the the value as 6. The value of mean by log normal and chisquare is approximately closer to the generated value.

1.6 Gamma Distribution

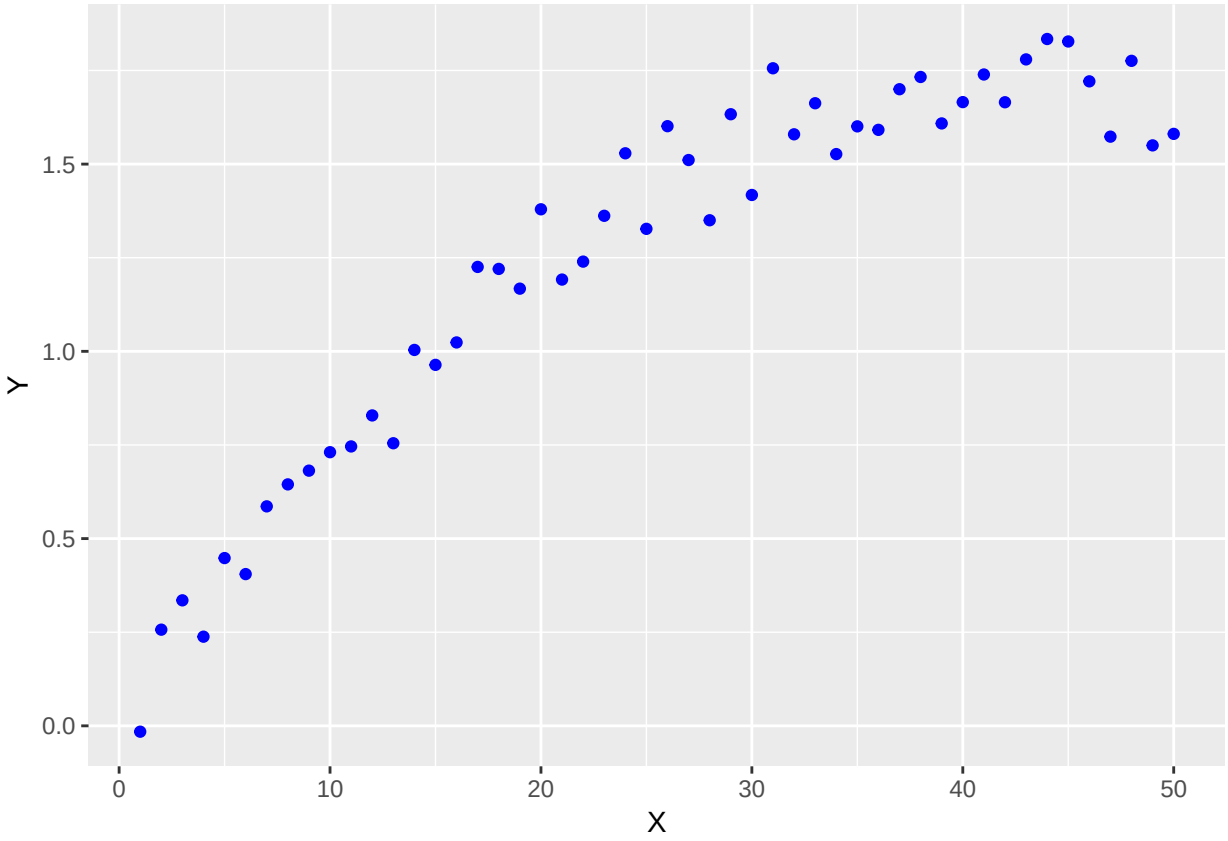
The Gamma distribution equation is :

$$\Gamma(\alpha) = \int_0^\infty x^{\alpha-1} * e^{-x} dx$$

Comparing with our equation $f(x) = \int_0^\infty x^5 * e^{-x}$, we can deduce that $\alpha = 6$. The mean of the equation is α which is 6. The mean generated using M-H algorithm is close to our mean.

2 Gibbs Sampling

2.1 Read Data and Plot X vs Y



We think a polynomial regression would be a good fit to the data. A curve would perfecly catch the dependence between X and Y.

2.2 Calculate Likelyhood and prior

Given :

$$Y_i \sim \mathcal{N}(\mu_i, \sigma^2 = 0.2)$$
$$p(\mu_1) = 1$$
$$p(\mu_{i+1}|\mu_i) \sim \mathcal{N}(\mu_i, 0.2),\ i = 1, 2, 3..., n-1$$

Where prior is $p(\mu_1)$ and $p(\mu_{i+1}|\mu_i)$. We can use chain rule to calculate the prior as shown below.

$$p(\mu) = p(\mu_1).p(\mu_2|\mu_1).p(\mu_3|\mu_2)...p(\mu_n|\mu_{n-1})$$
$$p(\mu) = k.e^{-\frac{1}{2.\sigma^2}.\sum_{i=1}^{n-1}(\mu_{i+1}-\mu i)^2}$$

Each of Y_i is dependent on μ_i hence we can calculate likelyhood $P(Y|\mu)$ as :

$$p(Y|\vec{\mu}) = p(Y_1|\vec{\mu_1}).p(Y_2|\vec{\mu_2}).p(Y_3|\vec{\mu_3})...p(Y_n|\vec{\mu_n})$$
$$p(Y|\vec{\mu}) = \prod_{i=1}^{n} p(Y_i|\vec{\mu_i})$$
$$p(Y|\vec{\mu}) = k.e^{-\frac{1}{2.\sigma^2}\cdot\sum_{i=1}^{n}(Y_i-\mu i)^2}$$

2.3 Calculating posterior and conditional distributions.

We know the posterior formula as:

$$P(\mu|Y) \propto P(Y|\mu).P(\mu)$$

Using the derived likelyhood, prior as calculated before, we can use it in this equation to get posterior as :

$$P(\mu|Y) \propto k.e^{-\frac{1}{2.\sigma^2}\cdot\sum_{i=1}^{n}(Y_i-\mu i)^2}.e^{-\frac{1}{2.\sigma^2}\cdot\sum_{i=1}^{n-1}(\mu_{i+1}-\mu i)^2}$$
$$P(\mu|Y) \propto e^{-\frac{1}{2.\sigma^2}(\sum_{i=1}^{n}(Y_i-\mu i)^2+\sum_{i=1}^{n-1}(\mu_{i+1}-\mu i)^2)}$$

But for each μ we need to get a conditional distribution where μ_i is dependent on μ_{-i}. Using Hint A, B and C we can get conditional distribution as:

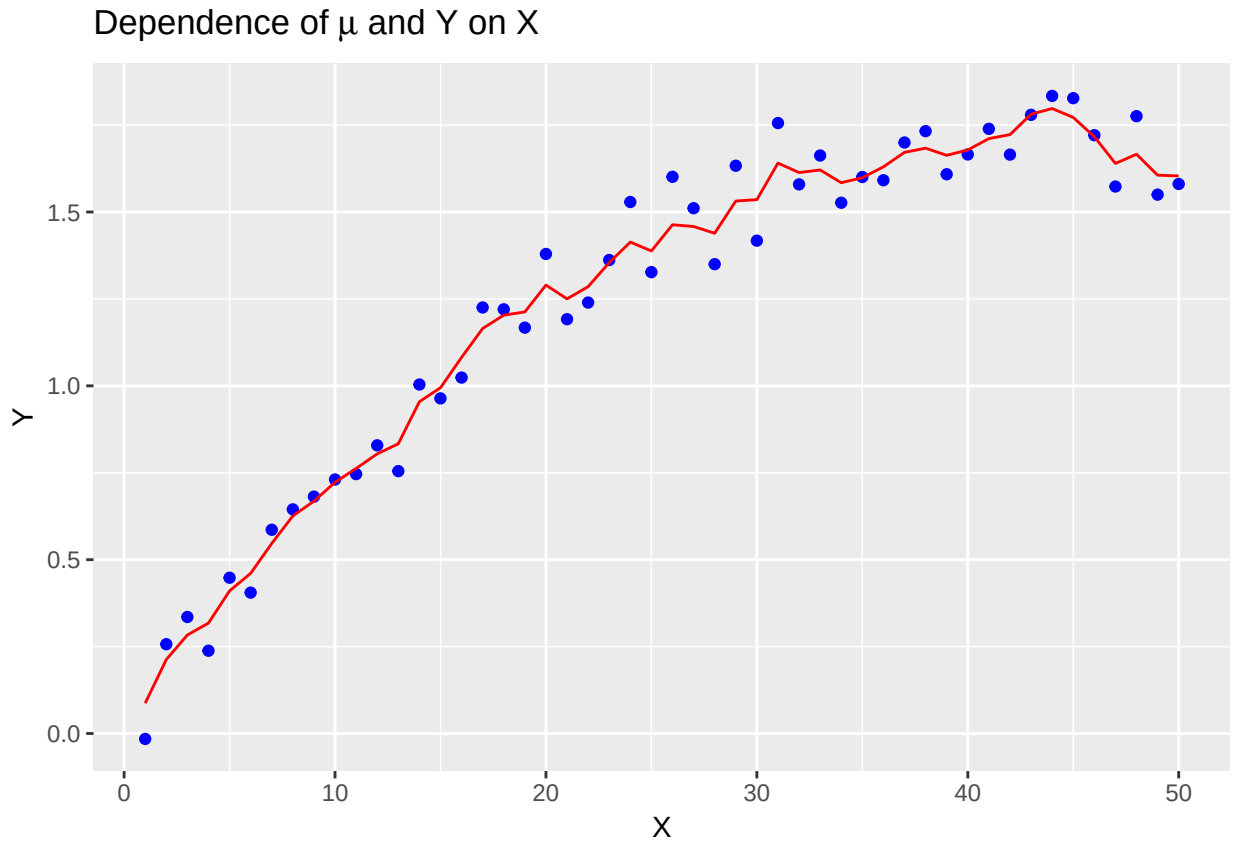
$$P(\mu_1|\mu_{-1}, Y) = e^{-\frac{1}{2.\sigma^2}(\mu_1-(\frac{\mu_2+y_1}{2}))^2}$$
$$P(\mu_i|\mu_{-i}, Y)) = e^{-\frac{1}{2.\sigma^2}(\mu_i-(\frac{\mu_{i-1}+\mu_{i+1}+y_n}{3}))^2}$$
$$P(\mu_n|\mu_{-n}, Y)) = e^{-\frac{1}{2.\sigma^2}(\mu_n-(\frac{\mu_{n-1}+y_n}{2}))^2}$$

Each of the above distribution is normally distributed with below properties :

$$\mu_1 \sim \mathcal{N}(\frac{\mu_2+Y_1}{2}, \sigma^2 = 0.2)$$
$$\mu_i \sim \mathcal{N}(\frac{\mu_{i-1}+\mu i+1+Y_i}{3}, \sigma^2 = 0.2)$$
$$\mu_n \sim \mathcal{N}(\frac{\mu_{n-1}+Y_n}{2}, \sigma^2 = 0.2)$$

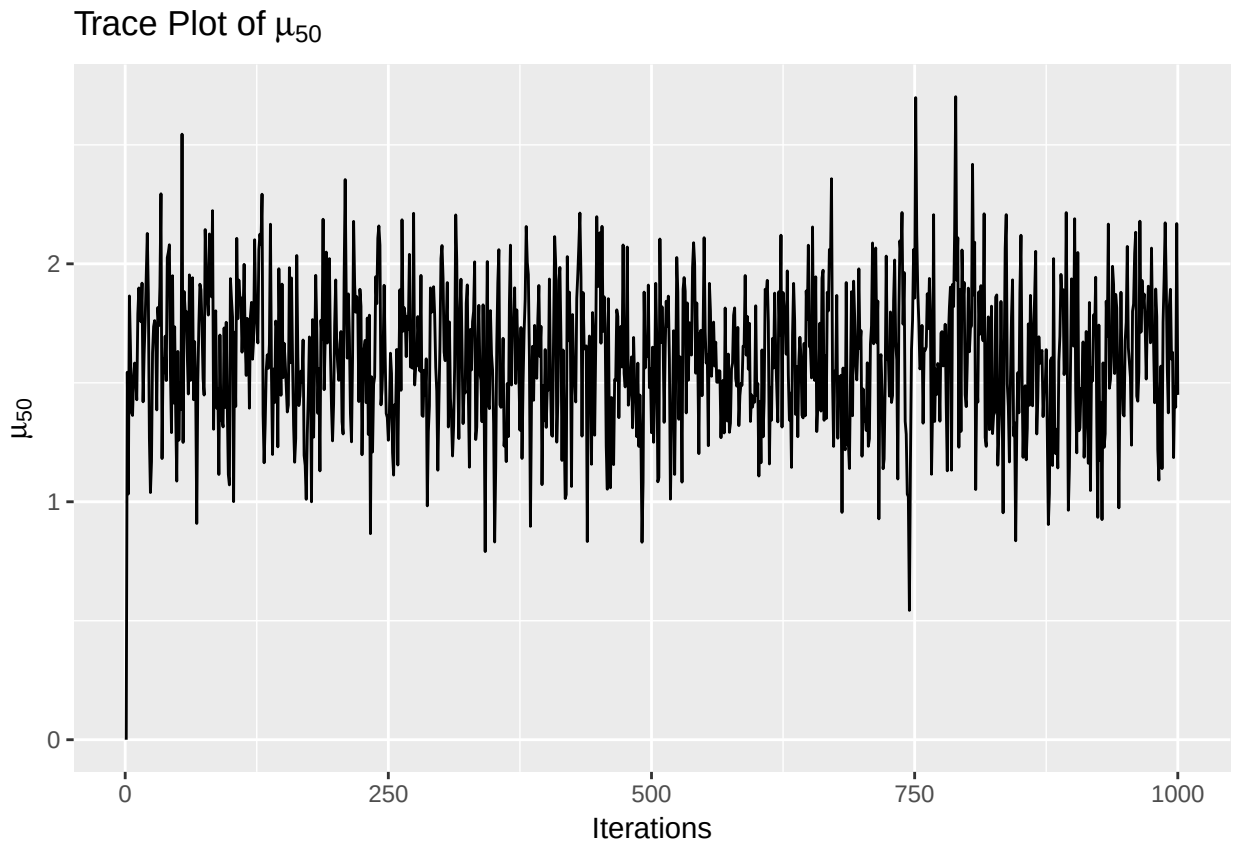
The μ are found out by running multiple iterations.

2.4 Gibbs Sampler



The red line shows the trace of expected μ value and blue line is the dependence of Y on X. From the graph,yes it looks like the line fits the data quite well removing much of the noise. I think the expected value of mu catches the underlying dependence between X and Y.

2.5 Trace Plot



The mu value gets close to the actual distribution of Y after the first iteration. For μ_n the update in the first iteration ends up being the average of Y_n and μ_{n-1}(which is close to the average of the previous two points of Y), so the μ_n for the first iteration ends up being close to the average of the points Y_i and Y_{i-1}. So the burn in period would be just one iteration.

The value of mu keeps fluctuating. Since we are generating a random number using rnorm with a small standard deviation in each iteration, the value of μ stays close to the average of the points Y_n and Y_{n-1}.

3 Appendix

```
knitr::opts_chunk$set(
    echo = TRUE,
    eval=TRUE,
    message = FALSE,
    warning = FALSE,
    comment = NA
)

#libraries
library(coda) #To do Gelman-Rubin test
target_dist <- function(x){
     if(x<0){
         return(0)}
     else {
         return((x^5)*exp(-x))
  }
}

#Proposal Distribution
proposed_dist  <- function(rv){
    rlnorm(1, meanlog = log(rv), sdlog = 1)  #log normal
}

metropolis_algorithm <- function(samp,rand_no) {
    curr_x <- rand_no                #Current random value
    post_val <- rep(0,times = samp)  #Number of Posterior value which is accepted.
    prop_val <- c()                  #Generate values from the proposal distribution.
    cnt <- 1

#Iterate or generate 2000 points for target distribution using proposal distribution
     while(cnt <= samp){

#Generate a new value from proposal distribution based on current value and sd 1.
#Basically move to the near point. it is kind of markov move which is dependent on
#current value.

#Generate next RV based on the current RV.
        prop_x <- proposed_dist(curr_x)
        U <- runif(1, 0, 1)

#posterior <- likelyhood* prior
        posterior_prop <- target_dist(prop_x)*dlnorm(curr_x,mean = log(prop_x),sd = 1)
        posterior_curr <- target_dist(curr_x)*dlnorm(prop_x,mean = log(curr_x),sd = 1)

        ifelse(U < min(1,(posterior_prop/posterior_curr)),post_val[cnt] <- prop_x,
               post_val[cnt] <- curr_x)

        curr_x <- post_val[cnt]
        prop_val[cnt] <- prop_x

        cnt <- cnt+1
```

```
    }
return(list(post_val,prop_val))
}

val1 <- metropolis_algorithm(10000,50)

{plot(1:10000,y=val1[[1]],type="l",xlab="Samples",ylab="RV",col="black",ylim=c(0,50))
 lines(x=1:20,y=val1[[1]][1:20],col="red")
title(main = "Markov Chain for generated samples using Log Normal\n as proposal")
legend("topright",legend=c("Burn-in period"),
col=c("red"), lty=1:2, cex=0.8,title="Proposal Distribution")
}
proposed_dist2  <- function(rv){
    rchisq(1,df = 1,ncp = floor(rv+1)) #log normal
}

likelyhood <- function(rv){
    dchisq(rv,df=1,ncp = curr_x)
}

metropolis_algorithm <- function(samp,rand_no) {
    curr_x <- rand_no                #Current random value
    post_val <- rep(0,times = samp)  #Number of Posterior value which is accepted.
    prop_val <- c()                  #Generate values from the proposal distribution.
    cnt <- 1

#Iterate or generate 2000 points for target distribution using proposal distribution
     while(cnt <= samp){

#Generate a new value from proposal distribution based on current value and sd 1.
#Basically move to the near point. it is kind of markov move which is dependent on
#current value.

#Generate next RV based on the current RV.
        prop_x <- proposed_dist2(curr_x)
        U <- runif(1, 0, 1)

#posterior <- likelyhood* prior
        posterior_prop <- target_dist(prop_x) * dchisq(curr_x,df=1,ncp  = prop_x)
        posterior_curr <- target_dist(curr_x) * dchisq(prop_x,df=1,ncp = curr_x)

        ifelse(U < min(1,(posterior_prop/posterior_curr)),post_val[cnt] <- prop_x,
               post_val[cnt] <- curr_x)

        curr_x <- post_val[cnt]
        prop_val[cnt] <- prop_x

        cnt <- cnt+1

    }

return(list(post_val,prop_val))
}
```

```
val2 <- metropolis_algorithm(10000,50)

{plot(1:10000,y=val2[[1]],type="l",xlab="Samples",ylab="RV")
title(main = "Markov Chain for generated samples using Chi Square \n as proposal")
 lines(x=1:20,y=val2[[1]][1:20],col="red")
 legend("topright",legend=c("Burn-in period"),
col=c("red"), lty=1:2, cex=0.8,title="Proposal Distribution")
}

{plot(x=1:10000,y=val1[[1]],type="l",xlab="Samples",ylab="RV",col="red",ylim=c(0,50))
lines(x=1:10000,y=val2[[1]],col="blue")
title("Markov Chain for generated samples by \n Chi Square vs Log Normal")
legend("topright",legend=c("Chi Square","Log Normal"),
col=c("blue","red"), lty=1:2, cex=0.8,title="Proposal Distribution")
}

{plot(density(val1[[1]]),main="Density plot of samples generated\n from Log normal")
}
{plot(density(val2[[1]]),main="Density plot of samples generated\n from Chi Square")
}
#Generate multiple markov chain
multi_mc <- lapply(1:10,function(x) as.mcmc(metropolis_algorithm(10000,x)[[1]][20:10000]))

multi_mc <- as.mcmc.list(multi_mc)
gelman.diag(multi_mc)
gelman.plot(multi_mc,main="Convergence of markov chain")
sum1 = 0
sum2 = 0
for (i in 1:10000){
    sum1 = sum1 + (val1[[1]][i])
    sum2 = sum2 + (val2[[1]][i])
}

sum1 =sum1/10000
sum2 =sum2/10000

cat("Mean of samples by Log normal:",sum1)
cat("\nMean of samples by Chi Sqaure:",sum2)
knitr::opts_chunk$set(
    echo = TRUE,
    eval=TRUE,
    message = FALSE,
    warning = FALSE,
    comment = NA
)

library(ggplot2)
#1
load("chemical.RData")
ggplot() + geom_point(aes(X, Y), col="blue")
#4
f.MCMC.Gibbs<-function(nstep, Y){
  n = length(Y)
```

```
  mu = matrix(0, nrow = nstep, ncol = n)
  mu[1, ] = rep(0,n)
  for (i in 2:nstep){
    last_mu = mu[i-1, ]
    new_mu = mu[i, ]
    for(j in 1:n){
      mu_new = ifelse(j==1, (Y[1] + last_mu[2])*0.5,
                      ifelse(j==n, (Y[n] + new_mu[n-1])*0.5,
                             (Y[j]+new_mu[j-1]+last_mu[j+1])/3 ))
      var_new = ifelse(i==1, 0.2/2,
                       ifelse(i==n, 0.2/2,
                              0.2/3))
      new_mu[j] = rnorm(1, mu_new, sqrt(var_new))
    }
    mu[i, ] = new_mu
  }
  return(mu)
}

mus = f.MCMC.Gibbs(nstep = 1000, Y)
mu_avg = colSums(mus)/1000
ggplot() + geom_point(aes(X, Y), colour="blue") + geom_line(aes(X, mu_avg), colour="red") +ggtitle(expr
#5 Trace plot
Iter = 1:1000
MU_N = mus[,50]
ggplot() + geom_line(aes(Iter, MU_N)) +
    ylab(expression(mu[50])) +
    xlab("Iterations") +
    ggtitle(expression(paste("Trace Plot of ", mu[50])))
```