Computational Statistics (732A90) Lab4

Anubhav Dikshit(anudi287) and Thijs Quast(thiqu264)

07 Feburary 2019

Contents

Contributions:

Hector Plata(hecpl268) helped us a lot in the second part of the assignment.

Question 1: Computations with Metropolis-Hastings

Consider the pdf:

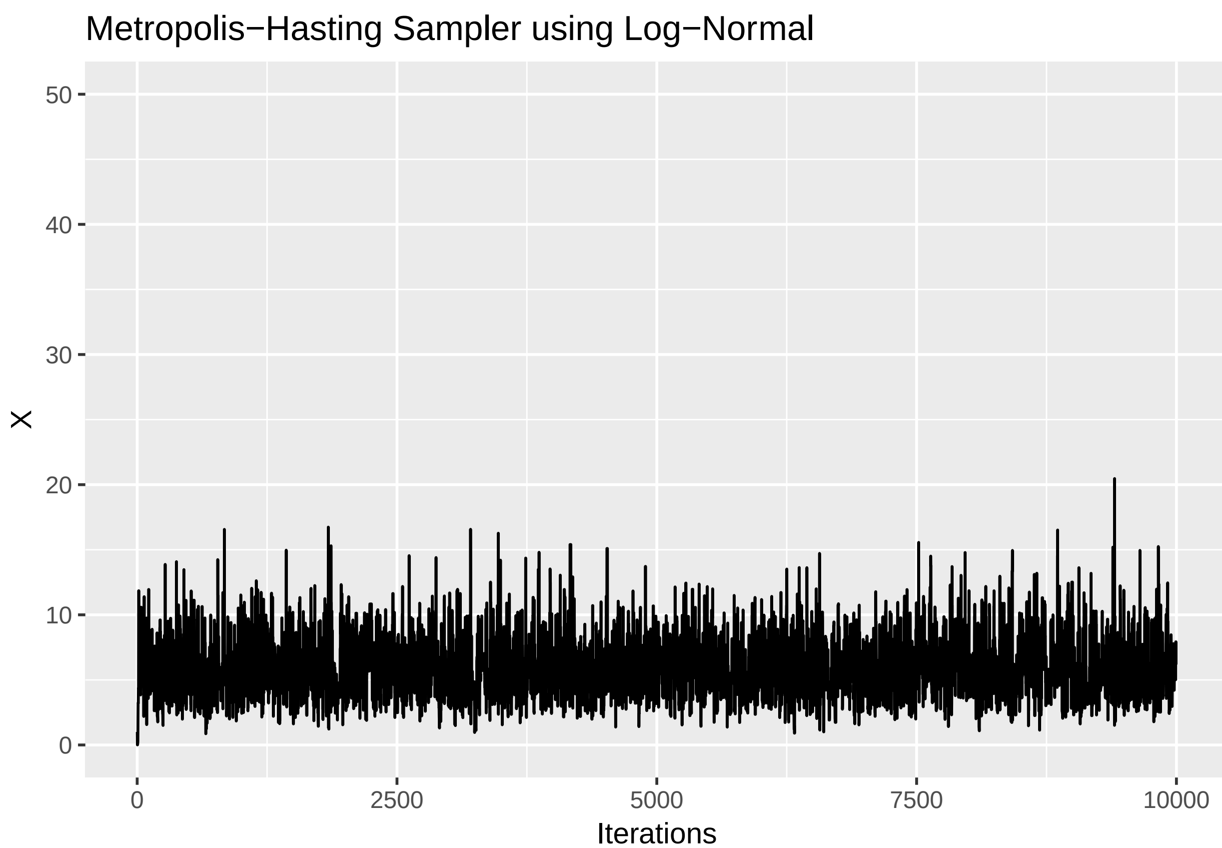
$$f(X) \sim x^5 e^{-x}, x > 0$$

1. Use Metropolis-Hastings algorithm to generate samples from this distribution by using proposal distribution as log-normal LN(Xt; 1), take some starting point. Plot the chain you obtained as a time series plot. What can you guess about the convergence of the chain? If there is a burn-in period, what can be the size of this period?

```
target<- function(x){
 result <- (x^5 * exp(-x))
  return(result)
}

proposed <- function(x,log_mean){
 result <- dlnorm(x,meanlog=log(log_mean), sd=1)
 return(result)
}

N <- 10^4
final <- rep(1,0,N) #Vx
x0 <-  0.004;

for(i in 2:N){
xprime <- rlnorm(1, log(x0), sdlog = 1) # proposed starting value
ratio <- min(c(1,((target(xprime)*proposed(x0, xprime))/(target(x0)*proposed(xprime, x0)))))
accept <- (runif(1) <= ratio)
final[i] <- ifelse(accept,xprime,x0)
x0 <- final[i]
}

step_1 <- final
mean(final)
```

```
## [1] 5.908561
```

```
ggplot() +
geom_line(aes(x=1:N, y=final)) +
labs(x="Iterations", y="X") +
ylim(0, 50) +
ggtitle("Metropolis-Hasting Sampler using Log-Normal")
```



Analysis: The series convergence as evident from the graph, there is a small burn-in period of about 2 iterations.

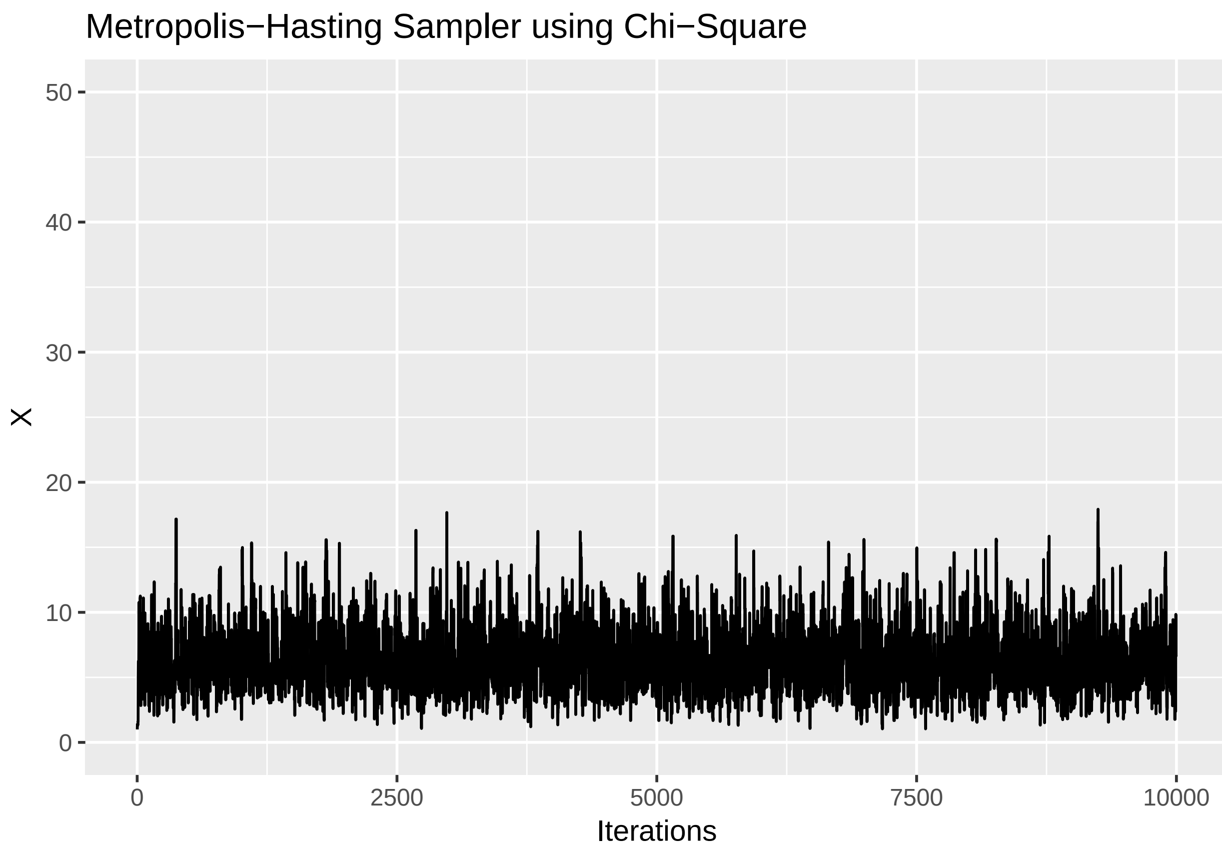
2. Perform Step 1 by using the chi-square distribution chi^2(floor(X(t)+1)) as a proposal distribution, where [x] is the floor function, meaning the integer part of x for positive x, i.e. [2.95]=2.

```
proposed_chi <- function(x){
 result <- rchisq(1, df=floor(x+1))
 return(result)
}

N <- 10^4
x0 <- 0.001 # there is a small burn in
#x0 <- 40 #starting point no burn in
final <- rep(NA,0,N)

for(i in 1:N){
xprime <- proposed_chi(x0) # proposed starting value
ratio <- min(c(1,((target(xprime)*proposed(x0, xprime))/(target(x0)*proposed(xprime, x0)))))
accept <- (runif(1) < ratio)
final[i] <- ifelse(accept,xprime,x0)
x0 <- final[i]
}
```

```
step_2 <- final
mean(final)
```

```
## [1] 6.234919
```

```
ggplot() +
geom_line(aes(x=1:N, y=final)) +
ylim(0, 50) +
labs(x="Iterations", y="X") + ggtitle("Metropolis-Hasting Sampler using Chi-Square")
```



3. Compare the results of Steps 1 and 2 and make conclusions.

Convergence is quicker in step1 when compared to step2. Also the mean is greather when using chi-square compared to Log-normal distribution.

4. Generate 10 MCMC sequences using the generator from Step 2 and starting points 1,2,3.... or 10. Use the Gelman-Rubin method to analyze convergence of these sequences.

```
target<- function(x){
 result <- (x^5 * exp(-x))
  return(result)
}
```

```r
proposed_chi <- function(x){
 result <- rchisq(1, df=floor(x+1))
 return(result)
}

function_chi_fit <- function(x0){

N <- 10^4
x0 <- x0 # there is a small burn in
final <- rep(NA,0,N)

for(i in 1:N){
xprime <- proposed_chi(x0) # proposed starting value
ratio <- min(c(1,((target(xprime)*proposed(x0, xprime))/(target(x0)*proposed(xprime, x0)))))
accept <- (runif(1) < ratio)
final[i] <- ifelse(accept,xprime,x0)
x0 <- final[i]
}
return(final)
}

all_series <- NULL
for(i in 1:10) {
temp <- function_chi_fit(x0=i)
temp <- coda::as.mcmc(temp)
all_series[[i]] <- temp
}

#convergence analysis
coda::gelman.diag(all_series)
```

```
## Potential scale reduction factors:
##
##      Point est. Upper C.I.
## [1,]          1          1
```

Analysis: As we know the Upper confidence interval value should be as close to 1 as possible, this indicates that our series has converged. Our Upper C.I is 1, thus we can conclude that we series is indeed converged.

5. Estimate Integral of xf(x), with limits from 0 to infinity using step 1 and 2.

When that our series convergence using Log-normal and Chi-square distribution we can use the following identity.

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

```r
mean(step_1)
```

```
## [1] 5.908561
```

```r
mean(step_2)
```

```
## [1] 6.234919
```

6. The distribution generated is in fact a gamma distribution. Look in the literature and define the actual value of the integral. Compare it with the one you obtained.

The pdf of gamma distribution is as follows:

$$f(x|k,\theta) = \frac{1}{\Gamma(k)\theta^k} x^{k-1} e^{\frac{-x}{\theta}}$$

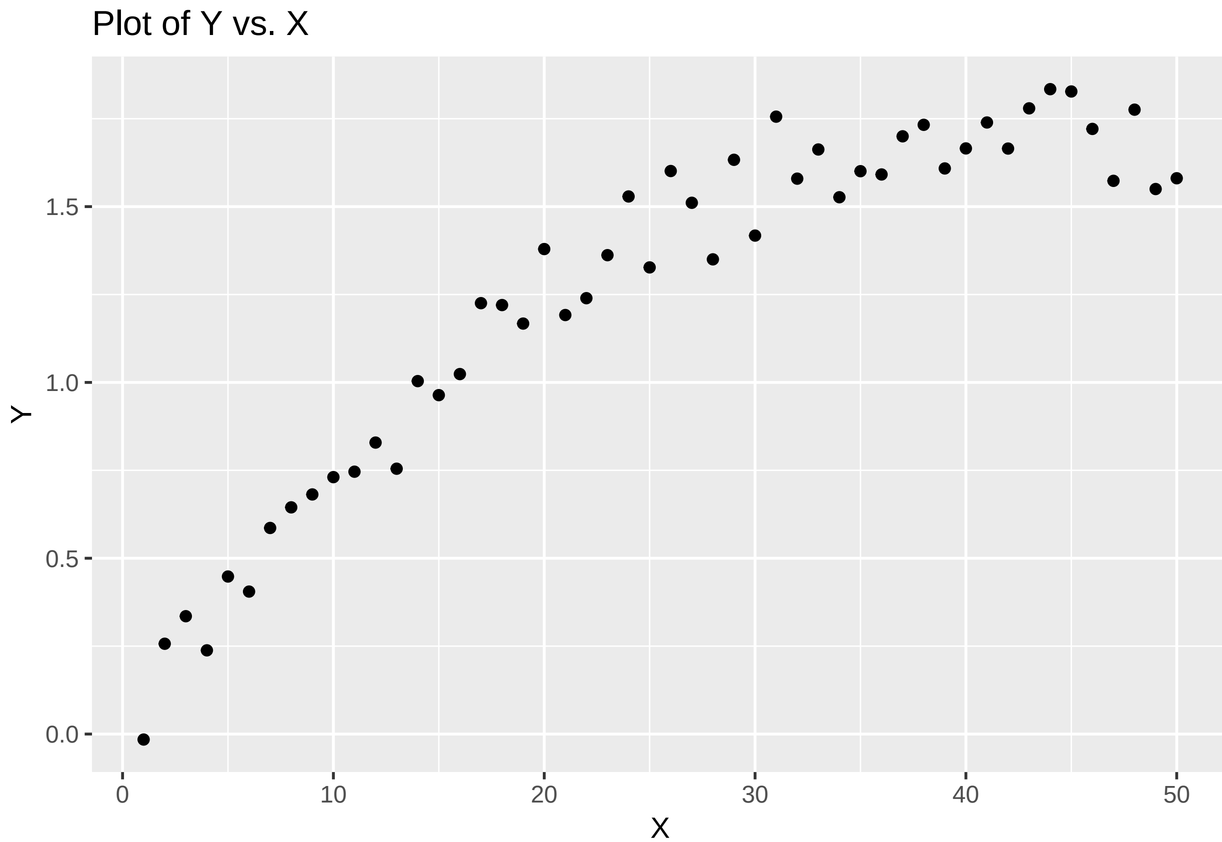
Analysis: Given at the begining that gamma value contast is 120, we get k=6 and $\theta = 1$, thus E[x]=k$\theta = 6$. We find that chi-square function gave a much closer values and this is due to the fact that chi-square is a special case of gamma distribution.

Question 2: Gibbs sampling

A concentration of a certain chemical was measured in a water sample, and the result was stored in the data chemical.RData having the following variables: X: day of the measurement Y: measured concentration of the chemical. The instrument used to measure the concentration had certain accuracy; this is why the measurements can be treated as noisy. Your purpose is to restore the expected concentration values.

1. Import the data to R and plot the dependence of Y on X. What kind of model is reasonable to use here?

```r
load(file= "chemical.Rdata")

data <- as.data.frame(cbind(X,Y))

ggplot(data=data,aes(x=X,y=Y)) + geom_point() + ggtitle("Plot of Y vs. X")
```



A linear fit may be good but a 2nd order polynominal or logarithm/expontial curve might be the best fit.

2. A researcher has decided to use the following (random-walk) Bayesian model (n=number of observations), mu = mu1,mu2,... are unknown parameters:

$$Y \approx N(\mu_i, varience = 0.2), i = 1, 2, 3...n$$

where the prior is

$$p(\mu) = 1$$

$$p(\mu_{i+1}|\mu_i) = N(\mu_i, varience = 0.2), i = 1, 2, 3...n$$

Present the formula showing the likelihood p(Y|mu) and the prior p(mu).

Solution:

From chain rule we can say prior will be

$$p(\vec{\mu}) = \prod_{i=1}^{n-1} N[\mu_i, \sigma^2](\mu_i + 1)$$

Likelihood:

$$p(Y|\mu) = \prod_{i=1}^{n} N[\mu_i, \sigma^2](Y_i)$$

Explaination: Here we have measurements of concentration of a certain chemical over time, however there is noise in the measurements. The idea is to "restore" the trend/mean using Gibbs sampling to recreate most probable distribution of the data negating the noise due to measurements error.

3. Use Bayes' Theorem to get the posterior up to a constant proportionality, and then find out the distributions of (mu_i|mu_-i, Y, where ~mu_is a vector containing all mu values except of mu_i

Applying Bayes theorem we get:

$$p(\mu|Y) \propto p(Y|\mu)p(\mu)$$

$$p(\mu|Y) \propto \exp(\frac{-1}{2\sigma^2}[(\mu_2-\mu_1)^2+(\mu_3-\mu_2)^2+.....+(\mu_n-\mu_{n-1})^2])\times\exp(\frac{-1}{2\sigma^2}[(Y_1-\mu_1)^2+(Y_2-\mu_2)^2+.....+(Y_n-\mu_n)^2])$$

Using property $\exp(\frac{-1}{d}((x-a)^2+(x-b)^2)) \propto \exp(-\frac{[x-\frac{(a+b)}{2}]^2}{\frac{d}{2}}$

Leading us to the following expression for posterior

$$p(\mu|Y) \propto \exp(-\frac{[\mu_n - Y_n]^2}{2\sigma^2} - \sum_{i=1}^{n-1}\frac{[\mu_i - \frac{\mu_{i+1}+Y_i}{2}]^2}{\sigma^2}$$

Marginal distribution for mu and sigma

mu=1

$$m_1 = 2\mu_{i-1} - Y_{i-1}$$
$$\sigma_1 = 2\sigma^2$$

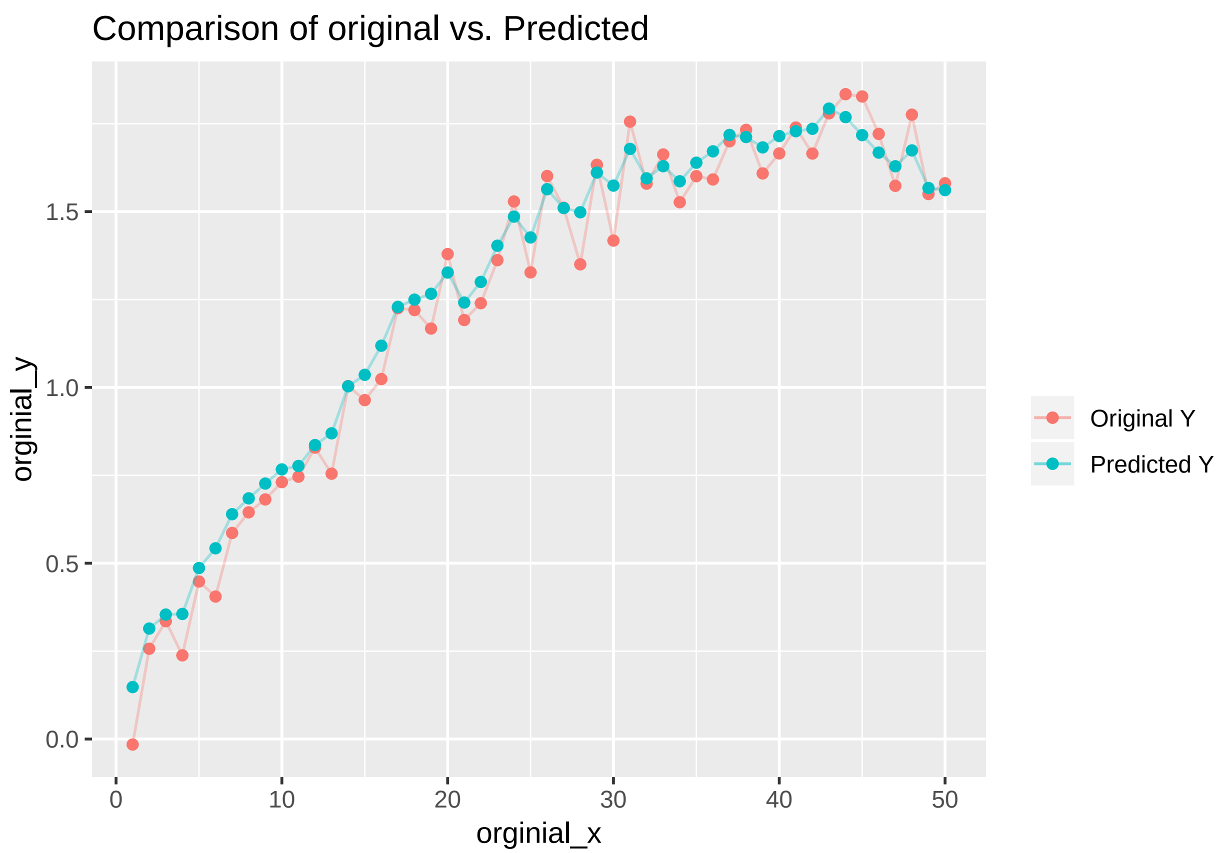
mu=2

$$m_2 = \frac{\mu_{i+1} + Y_i}{2}$$
$$\sigma_2 = \sigma^2/2$$

$$p(\mu i, \mu_{i-1}|Y) = N(\frac{1}{5}(2Y_i + 2\mu_{i-1} + 2\mu_{i+1} - Y_{i-1}), \frac{2\sigma^2}{5})(\mu_i)$$

4. Use the distributions derived in Step 3 to implement a Gibbs sampler that uses mu = (0,. . . ,0) as a starting point. Run the Gibbs sampler to obtain 1000 values of mu and then compute the expected value of mu by using a Monte Carlo approach. Plot the expected value of mu versus X and Y versus X in the same graph. Does it seem that you have managed to remove the noise? Does it seem that the expected value of mu can catch the true underlying dependence between Y and X?

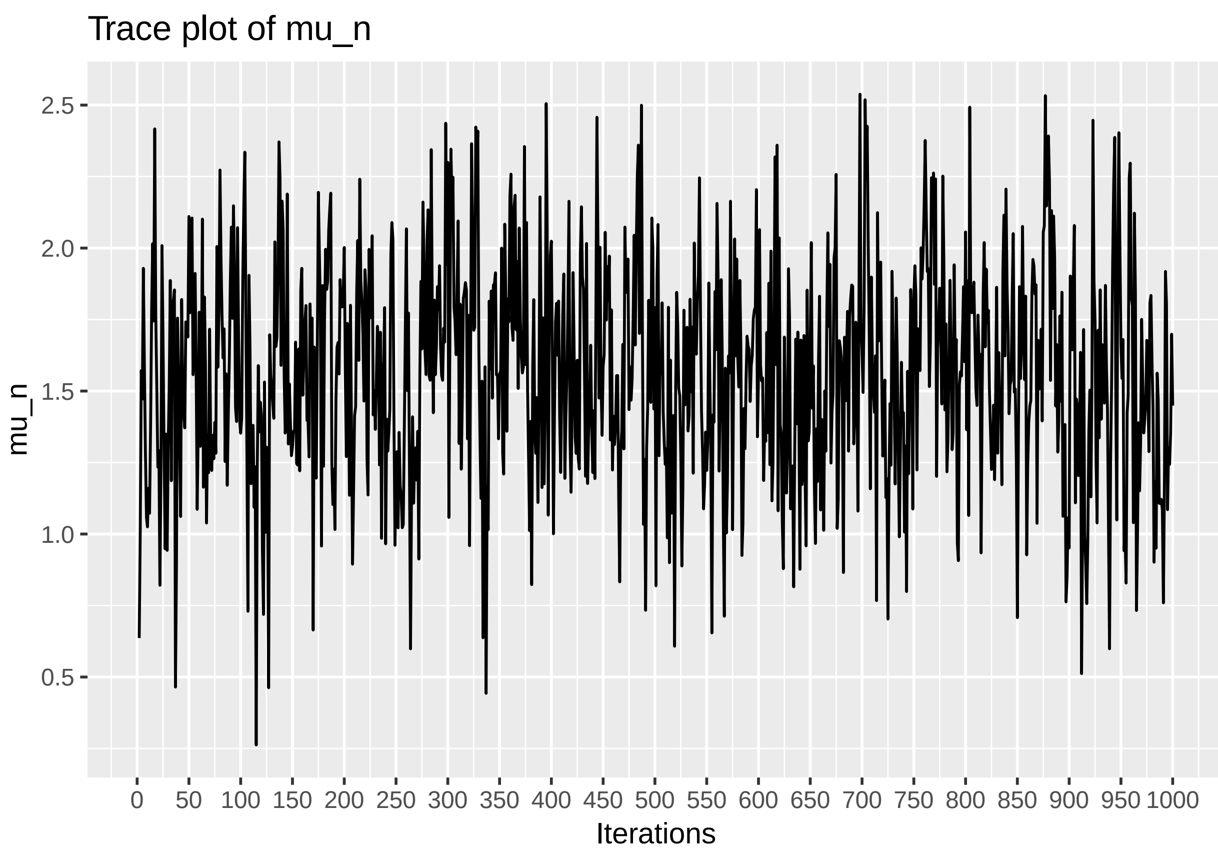
```r
gen_mu = function(i, mu, Y){

n = length(mu)
mean_mu <- rep(0L, n)
var_mu <- rep(0L, n)

mean_mu <- ifelse(i == 1, (Y[1] + mu[2])*0.5, ifelse(i == n, (2*Y[n] - Y[n-1] + 2*mu[n-1])*0.3333, (2*mu
var_mu <- ifelse(i == 1, 0.1, ifelse(i == n,2*0.2/5,0.08))

# Generate mu_i using a normally distributed marginal distribution
rnorm(1, mean_mu, sqrt(var_mu))
}


# Function to run Gibbs sampling
mcmc_gib = function(Y, num_iter){
n = length(Y)
mu0 = rep(0,length(Y))
mu = matrix(0, nrow = num_iter, ncol = n)
mu[1, ] = mu0
for (i in 2:num_iter){
latest_mu = mu[i-1, ]
for(j in 1:n){
latest_mu[j] = gen_mu(j, latest_mu, Y)
}
mu[i, ] = latest_mu
}
return(mu)
}

set.seed(12345)
result = mcmc_gib(Y, num_iter=1000)
exp_mu = colMeans(result)

ggplot(data.frame(orginial_x = X, exp_mu = exp_mu, orginial_y = Y)) +
geom_point(aes(x = orginial_x, y = orginial_y, color = "Original Y")) +
geom_line(aes(x = orginial_x, y = orginial_y, color = "Original Y"), alpha = 0.3) +
geom_point(aes(x = orginial_x, y = exp_mu, color = "Predicted Y")) +
geom_line(aes(x = orginial_x, y = exp_mu, color = "Predicted Y"), alpha = 0.3) +
labs(color="") + ggtitle("Comparison of original vs. Predicted")
```



Analysis: As evident fromt the plot, our predicted value is close to the actual value.

5. Make a trace plot for mu_n and comment on the burn-in period and convergence.

```
ggplot(data.frame(x = 2:nrow(result),
y = result[2:nrow(result), ncol(result)])) +
geom_line(aes(x = x, y = y)) +
xlab("Iterations") + ylab("mu_n") +
ggtitle("Trace plot of mu_n") +
scale_x_continuous(breaks = seq(0,1000,50))
```




```
head(result[,50],10)
```

```
##  [1] 0.0000000 0.6363762 1.0056683 1.5716017 1.4716282 1.9292479 1.5919426
##  [8] 1.2858624 1.0570105 1.0250274
```

Analysis: As seen from values the burn-in period is about 3 iterations and convergence is indeed evident from the plot.

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
options(scipen=999)
library(dplyr)
library(ggplot2)
library(coda)

target<- function(x){
 result <- (x^5 * exp(-x))
  return(result)
}

proposed <- function(x,log_mean){
 result <- dlnorm(x,meanlog=log(log_mean), sd=1)
 return(result)
}
```

```r
N <- 10^4
final <- rep(1,0,N) #Vx
x0 <-  0.004;

for(i in 2:N){
xprime <- rlnorm(1, log(x0), sdlog = 1) # proposed starting value
ratio <- min(c(1,((target(xprime)*proposed(x0, xprime))/(target(x0)*proposed(xprime, x0)))))
accept <- (runif(1) <= ratio)
final[i] <- ifelse(accept,xprime,x0)
x0 <- final[i]
}

step_1 <- final
mean(final)

ggplot() +
geom_line(aes(x=1:N, y=final)) +
labs(x="Iterations", y="X") +
ylim(0, 50) +
ggtitle("Metropolis-Hasting Sampler using Log-Normal")


proposed_chi <- function(x){
 result <- rchisq(1, df=floor(x+1))
 return(result)
}

N <- 10^4
x0 <- 0.001 # there is a small burn in
#x0 <- 40 #starting point no burn in
final <- rep(NA,0,N)

for(i in 1:N){
xprime <- proposed_chi(x0) # proposed starting value
ratio <- min(c(1,((target(xprime)*proposed(x0, xprime))/(target(x0)*proposed(xprime, x0)))))
accept <- (runif(1) < ratio)
final[i] <- ifelse(accept,xprime,x0)
x0 <- final[i]
}

step_2 <- final
mean(final)

ggplot() +
geom_line(aes(x=1:N, y=final)) +
ylim(0, 50) +
labs(x="Iterations", y="X") + ggtitle("Metropolis-Hasting Sampler using Chi-Square")


target<- function(x){
 result <- (x^5 * exp(-x))
  return(result)
}
```

```
proposed_chi <- function(x){
 result <- rchisq(1, df=floor(x+1))
 return(result)
}

function_chi_fit <- function(x0){

N <- 10^4
x0 <- x0 # there is a small burn in
final <- rep(NA,0,N)

for(i in 1:N){
xprime <- proposed_chi(x0) # proposed starting value
ratio <- min(c(1,((target(xprime)*proposed(x0, xprime))/(target(x0)*proposed(xprime, x0)))))
accept <- (runif(1) < ratio)
final[i] <- ifelse(accept,xprime,x0)
x0 <- final[i]
}
return(final)
}

all_series <- NULL
for(i in 1:10) {
temp <- function_chi_fit(x0=i)
temp <- coda::as.mcmc(temp)
all_series[[i]] <- temp
}

#convergence analysis
coda::gelman.diag(all_series)


mean(step_1)
mean(step_2)

load(file= "chemical.Rdata")

data <- as.data.frame(cbind(X,Y))

ggplot(data=data,aes(x=X,y=Y)) + geom_point() + ggtitle("Plot of Y vs. X")


gen_mu = function(i, mu, Y){

n = length(mu)
mean_mu <- rep(0L, n)
var_mu <- rep(0L, n)

mean_mu <- ifelse(i == 1, (Y[1] + mu[2])*0.5, ifelse(i == n, (2*Y[n] - Y[n-1] + 2*mu[n-1])*0.3333, (2*mu
var_mu <- ifelse(i == 1, 0.1, ifelse(i == n,2*0.2/5,0.08))

# Generate mu_i using a normally distributed marginal distribution
rnorm(1, mean_mu, sqrt(var_mu))
```

```
}


# Function to run Gibbs sampling
mcmc_gib = function(Y, num_iter){
n = length(Y)
mu0 = rep(0,length(Y))
mu = matrix(0, nrow = num_iter, ncol = n)
mu[1, ] = mu0
for (i in 2:num_iter){
latest_mu = mu[i-1, ]
for(j in 1:n){
latest_mu[j] = gen_mu(j, latest_mu, Y)
}
mu[i, ] = latest_mu
}
return(mu)
}

set.seed(12345)
result = mcmc_gib(Y, num_iter=1000)
exp_mu = colMeans(result)

ggplot(data.frame(orginial_x = X, exp_mu = exp_mu, orginial_y = Y)) +
geom_point(aes(x = orginial_x, y = orginial_y, color = "Original Y")) +
geom_line(aes(x = orginial_x, y = orginial_y, color = "Original Y"), alpha = 0.3) +
geom_point(aes(x = orginial_x, y = exp_mu, color = "Predicted Y")) +
geom_line(aes(x = orginial_x, y = exp_mu, color = "Predicted Y"), alpha = 0.3) +
labs(color="") + ggtitle("Comparison of original vs. Predicted")

ggplot(data.frame(x = 2:nrow(result),
y = result[2:nrow(result), ncol(result)])) +
geom_line(aes(x = x, y = y)) +
xlab("Iterations") + ylab("mu_n") +
ggtitle("Trace plot of mu_n") +
scale_x_continuous(breaks = seq(0,1000,50))


head(result[,50],10)
```