732A90-Lab4-Report

Zhixuan Duan(zhidu838) Fengjuan Chen(fench417)

2/21/2020

Question 1 Computations with Metropolis-Hastings

1. Metropolis-Hastings using proposal distribution log-normal

The steps of Metropolis-Hastings algorithm are as follows.

Step1. Initialize chain to X_0, $t = 0$

Step2. While $t < t_{max}$ do

(a) Generate a candidate point $Y \sim q(\cdot|X_t)$, where $q(\cdot|X_t)$ is the proposal distribution.

(b) Generate $U \sim Unif(0,1)$

(c) if $U < \alpha(X_t, Y)$ then $X_{t+1} = Y$

(d) else $X_{t+1} = X_t$

(e) $t = t + 1$

The $\alpha(X_t, Y) = min\{1, \frac{\pi(Y)q(X_t|Y)}{\pi(X_t)q(Y|X_t)}\}$. The function $\pi(\cdot)$ is the target distribution we want to sample from. The function $q(\cdot|X_t)$ is the proposal distribution.

Here, the $\pi(\cdot) = f(x) = x^5 e^{-x}, x > 0$ and proposal distribution is a log-normal $LN(X_t, 1)$.

Since the log-normal distribution function is $\frac{1}{x\delta\sqrt{2\pi}} e^{-\frac{(lnx-u)^2}{2\delta^2}}$, $LN(X_t, 1) = \frac{1}{x\sqrt{2\pi}} e^{-\frac{(lnx-X_t)^2}{2}}$ when $u = X_t$ and $\delta = 1$.

Then we have

$Y \sim q(\cdot|X_t) \Rightarrow Y = LN(mean = X_t, sd = 1)$

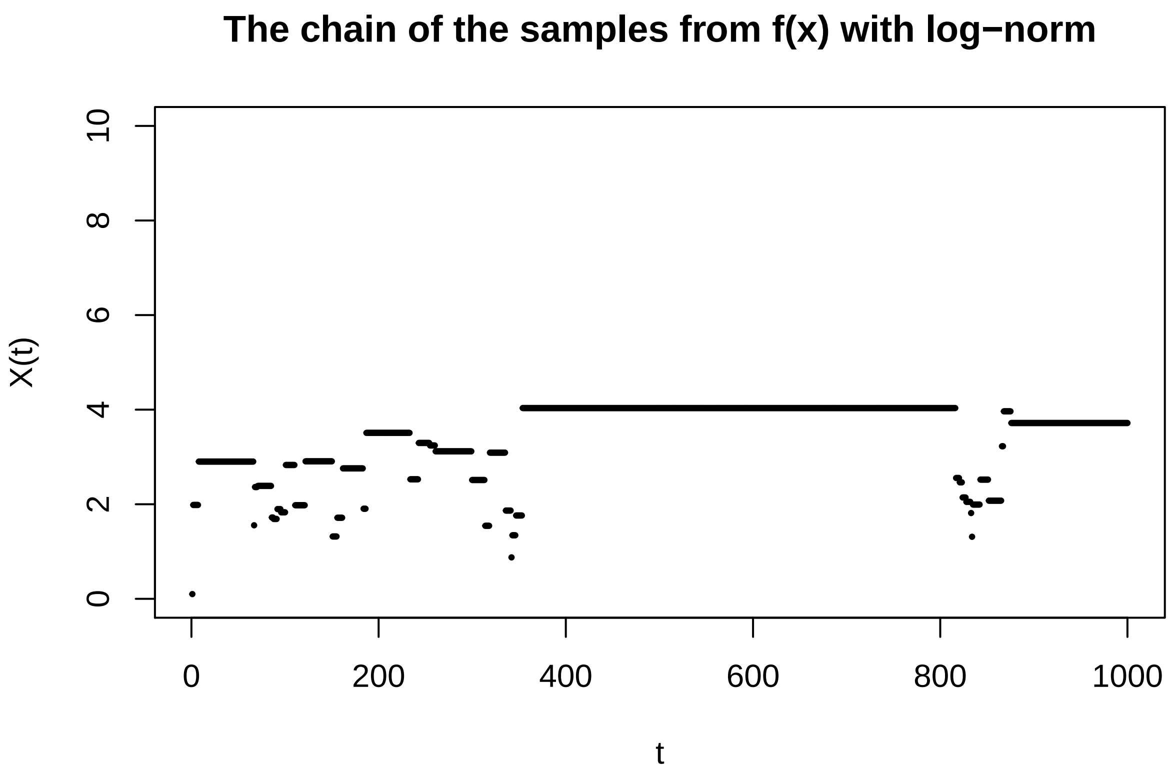
$q(X_t|Y) = LN(X_t, mean = Y, sd = 1)$, $q(Y|X_t) = LN(Y, mean = X_t, sd = 1)$,

$\pi(Y) = f(Y) = Y^5 e^{-Y}, Y > 0$ and $\pi(X_t) = f(X_t) = X_t^5 e^{-X_t}, X_t > 0$

The basic function rlnorm() in R can generate a random number from the log-normal distribution while the function dlnorm() can give the density of the log-normal distribution.

The function of Metropolis-Hastings algorithm with proposal distribution log-normal is shown below.

```r
MetroHast <- function(nstep,x0){
  sequen_x=vector(length = nstep)
  sequen_x[1]=x0
  for (i in 2:nstep) {
    x=sequen_x[i-1]
    y=rlnorm(1,meanlog = x,sdlog = 1)
    u=runif(1)
    alpha=(y^5*exp(1)^-y)*(dlnorm(x,meanlog = y,sdlog = 1))/(
            (x^5*exp(1)^-x)*dlnorm(y,meanlog = x,sdlog = 1))
    a=min(1,alpha)
    sequen_x[i]=ifelse(u<=a,y,x)
  }
  return(sequen_x)
}
```



From the plot we can guess that the chain will be converged. However this chain is not very good Markov chain because many adjacent points have almost the same values.

With this starting point $x = 0.1$, it almost has no burn-in period by observing the plot.

2. Metropolis-Hastings using proposal distribution chi-squares

Change the proposal distribution from log-normal distribution to χ^2 distribution with $df = floor(X_t + 1)$.

We have

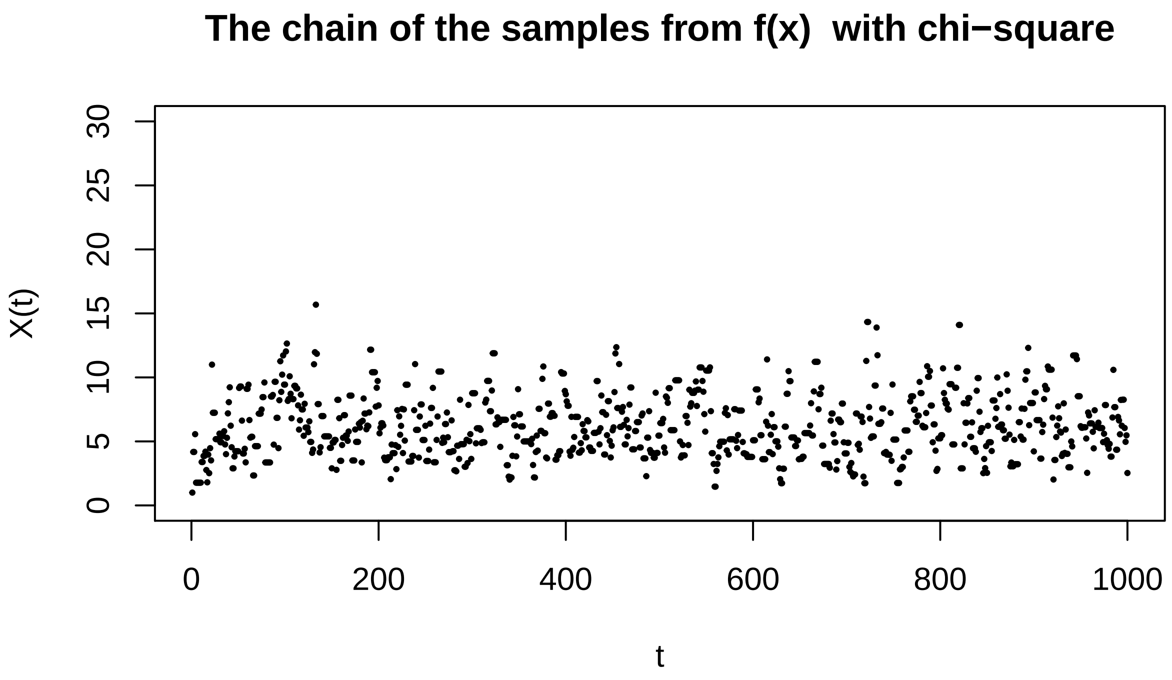
$Y \sim q(\cdot|X_t) \Rightarrow Y = \chi^2(df = floor(X_t + 1))$

$q(X_t|Y) = \chi^2(X_t, df = floor(Y + 1))$, $q(Y|X_t) = \chi^2(Y, df = floor(X_t + 1))$,

$\pi(Y) = f(Y) = Y^5 e^{-Y}, Y > 0$ and $\pi(X_t) = f(X_t) = X_t^5 e^{-X_t}, X_t > 0$

The function of Metropolis-Hastings algorithm with proposal distribution $\chi^2(df = \lfloor X_t + 1 \rfloor)$ is shown below.

```
MetroHast2 <- function(nstep,x0){
  sequen_x=vector(length = nstep)
  sequen_x[1]=x0
  for (i in 2:nstep) {
    x=sequen_x[i-1]
    y=rchisq(1,df=floor(x+1))
    u=runif(1)
    alpha=(y^5*exp(1)^-y)*dchisq(x,df= floor(y+1))/((x^5*exp(1)^-x)*dchisq(y,df = floor(x+1)))
    a=min(1,alpha)
    sequen_x[i]=ifelse(u<=a,y,x)
  }
  return(sequen_x)
}
```



3. Compare Metropolis-Hastings algorithms with different proposals

In Step 1, we didn't obtain a good Markov chain even when we chose different starting points. The points in the chain do not show a perfect fluctuation between some boundaries. In Step 2, we obtained a good Markov chain which shows a better shape.

From these two results, we can concude that to generate samples from the given distribution by using Metropolis-Hastings algorithm the $\chi^2(\lfloor X_t+1 \rfloor)$ is a better proposal distribution than log-normal$LN(X_t, 1)$.

4. Gelman-Rubin method to analyze convergence

Generating k sequences of length n with different starting points and then computing the between- and within sequence variances can give the information about revergence. The Gelman-Rubin factor R can show whether the chain is lack of convergence or not.

In R package coda, the function gelman.diag() can implement the Gelman and Rubin's convergence diagnostic. The 'potential scale reduction factor' is calculated together with the upper confidence limit. The potential scale reduction factor substantially above 1 indicates lack of convergence.

```
f3=mcmc.list()
for (i in 1:10) {
  f3[[i]]=as.mcmc(MetroHast2(1000,i))
}
print(gelman.diag(f3))
```

```
## Potential scale reduction factors:
##
##      Point est. Upper C.I.
## [1,]          1       1.01
```

Good

In the result the label 'Point est.' is the value of the potential scale reduction factor while the label 'Upper C.I.' is the the upper confidence limits of the potential scale reduction factor.

The 'Point est.' and 'Upper C.I.' are both close to 1. This means that the chain is converged.

5. Estimate a definite integral

Estimation of a definite integral $\theta = \int_D f(x)dx$.

Decompose $f(x)$ into $f(x) = g(x)p(x)$ where $\int_D p(x)dx = 1$. Then the estimate of θ is

$\hat{\theta} = \frac{1}{n}\sum_{i=1}^{n} g(x_i)$, $\forall i, x_i \sim p(\cdot)$.

Estimating $\int_0^\infty xf(x)dx$, $g(x)$ above is equalt to x. Thus, the estimation of the $\int_0^\infty xf(x)dx$ is $\hat{\theta} = \frac{1}{n}\sum_{i=1}^{n} x_i$ $\forall i, x_i \sim x^5e^{-x}$.

So, the estimate of the integral is only equalt ot the average of the sample points from Step1 or Step2.

Using the samples from step1, we obtain the estimation is

```
## [1] 3.4436
```

Using the samples from step2, we obtain the estimation is

```
## [1] 6.171925
```

Good

6. The actual value of the integral

The gamma distribution is $f(x) = \frac{1}{\Gamma(\alpha)\beta^\alpha}x^{\alpha-1}e^{-x/\beta}$.

Because x^5e^{-x} is a gamma distribution, we have

$$\begin{cases} \alpha - 1 = 5 \\ -1/\beta = -1 \end{cases}$$

Then we have

$$\begin{cases} \alpha = 6 \\ \beta = 1 \end{cases}$$

The gamma distribution $f(x) = \frac{1}{\Gamma(\alpha)\beta^\alpha}x^{\alpha-1}e^{-x/\beta} = \frac{1}{\Gamma(6)1^6}x^5e^{-x} = \frac{1}{\Gamma(6)}x^5e^{-x}$, where $\Gamma(6) = 5! = 120$.

We want to calculate $\int_0^\infty xf(x)dx = \int_0^\infty x\frac{1}{\Gamma(6)}x^5e^{-x}dx = \frac{1}{\Gamma(6)}\int_0^\infty x^6e^{-x}dx$.

Since

$\int_0^\infty x^6 e^{-x} dx = -\int_0^\infty x^6 de^{-x} = -x^6 e^{-x} + 6\int_0^\infty x^5 e^{-x} dx$

$= [-x^6e^{-x} - 6x^5e^{-x} - 6*5x^4e^{-x} - 6*5*4x^3e^{-x} - 6*5*4*3*x^2e^{-x} - 6*5*4*3*2xe^{-x} - 6*5*4*3*2e^{-x}]_0^\infty$

$= 0 - (-6*5*4*3*2*e^0) = 6!.$

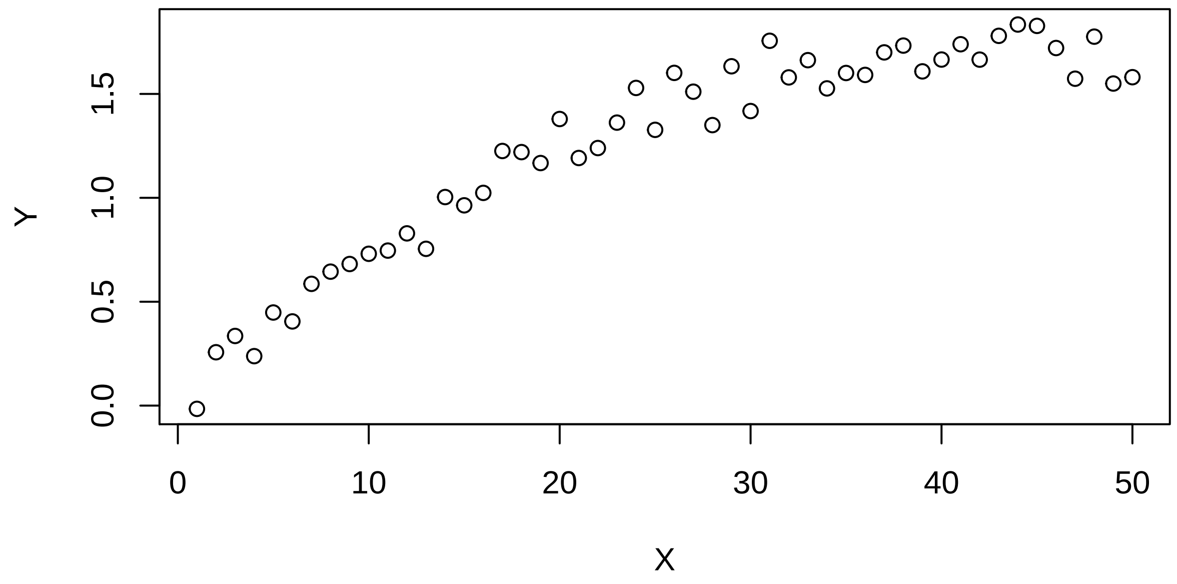
So, we have

$\int_0^\infty xf(x)dx = \frac{1}{\Gamma(6)}\int_0^\infty x^6e^{-x}dx = \frac{1}{\Gamma(6)} * 6! = 6.$ Good

The actual value of the integral is 6 and our estimations of the integral is 3.443 and 6.171925. Our estimation by using χ^2 as proposal is very close to the true value of the integral while the estimation by using log-normal as proposal is far from the true value. That means Metropolis-Hastings with proposal density χ^2 provides a good sample for the given distribution. This is coincident with the result we obtained above.

Question 2 Gibbs sampling

1. Plot the dependence of Y on X



What kind of model is reasonable to use here?

From the plot, the normal distribution seems reasonable for each Y_i.

2.The likelihood and prior

The likelihood

Since $Y_i \sim N(u_i, varaince = 0.2), i = 1, ..., n$, the likelihood of $(\vec{Y}|\vec{u})$ is

$p(\vec{Y}|\vec{u}) = \prod_{i=1}^n p(Y_i|\vec{u}) = \prod_{i=1}^n N(u_i, \delta^2 = 0.2) = \prod_{i=1}^n \frac{1}{\sqrt{2\pi}\delta} e^{-\frac{(y_i-u_i)^2}{2\delta^2}}$

$= (\frac{1}{\sqrt{2\pi}\delta})^n e^{-\frac{1}{2\delta^2}\sum_{i=1}^n (y_i-u_i)^2}$, where $\delta^2 = 0.2$. Correct likelihood. But \sigma, not \delta and \mu, not u.

The prior

The prior $p(\vec{u})$ is

$p(\vec{u}) = p(u_1)p(u_2|u_1)p(u_3|u_2)...p(u_n|u_{n-1})$

$= 1 \cdot \frac{1}{\sqrt{2\pi}\delta} e^{-\frac{(u_2-u_1)^2}{2\delta^2}} \cdot ... \cdot \frac{1}{\sqrt{2\pi}\delta} e^{-\frac{(u_n-u_{n-1})^2}{2\delta^2}}$

$= (\frac{1}{\sqrt{2\pi}\delta})^{n-1} e^{-\frac{1}{2\delta^2}\sum_{i=2}^n (u_i-u_{i-1})^2}$, where $\delta^2 = 0.2$. Correct prior

3. The posterior

The posterior

$p(\vec{u}|\vec{Y}) \propto p(\vec{u}) \cdot p(\vec{Y}|\vec{u})$

$\propto (\frac{1}{\sqrt{2\pi}\delta})^{n-1} e^{-\frac{1}{2\delta^2}\sum_{i=2}^n (u_i-u_{i-1})^2} \cdot (\frac{1}{\sqrt{2\pi}\delta})^n e^{-\frac{1}{2\delta^2}\sum_{i=1}^n (y_i-u_i)^2}$

$\propto e^{-\frac{1}{2\delta^2}(\sum_{i=2}^n (u_i-u_{i-1})^2 + \sum_{i=1}^n (y_i-u_i)^2)}$ Correct

Then

$p(\vec{u_1}|\vec{u}_{-1}, \vec{Y}) \propto e^{-\frac{1}{2\delta^2/2}(u_1 - \frac{u_2+y_1}{2})^2}$

$p(\vec{u_2}|\vec{u}_{-2}, \vec{Y}) \propto e^{-\frac{1}{2\delta^2/3}(u_2 - \frac{u_1+u_3+y_2}{3})^2}$

$p(\vec{u_3}|\vec{u}_{-3}, \vec{Y}) \propto e^{-\frac{1}{2\delta^2/3}(u_3 - \frac{u_2+u_4+y_3}{3})^2}$

$p(\vec{u}_{n-1}|\vec{u}_{-(n-1)}, \vec{Y}) \propto e^{-\frac{1}{2\delta^2/3}(u_{n-1} - \frac{u_{n-2}+u_n+y_{n-1}}{3})^2}$

$p(\vec{u}_n|\vec{u}_{-n}, \vec{Y}) \propto e^{-\frac{1}{2\delta^2/2}(u_n - \frac{u_{n-1}+y_n}{2})^2}$

To summarize, we have

$$p(\vec{u_i}|\vec{u}_{-i}, \vec{Y}) \sim \begin{cases} N(\frac{u_2+y_1}{2}, \frac{1}{2}\delta^2) & i = 1 \\ N(\frac{u_{i-1}+u_{i+1}+y_i}{3}, \frac{1}{3}\delta^2) & 1 < i < n \\ N(\frac{u_{n-1}+y_n}{2}, \frac{1}{2}\delta^2) & i = n \end{cases}$$

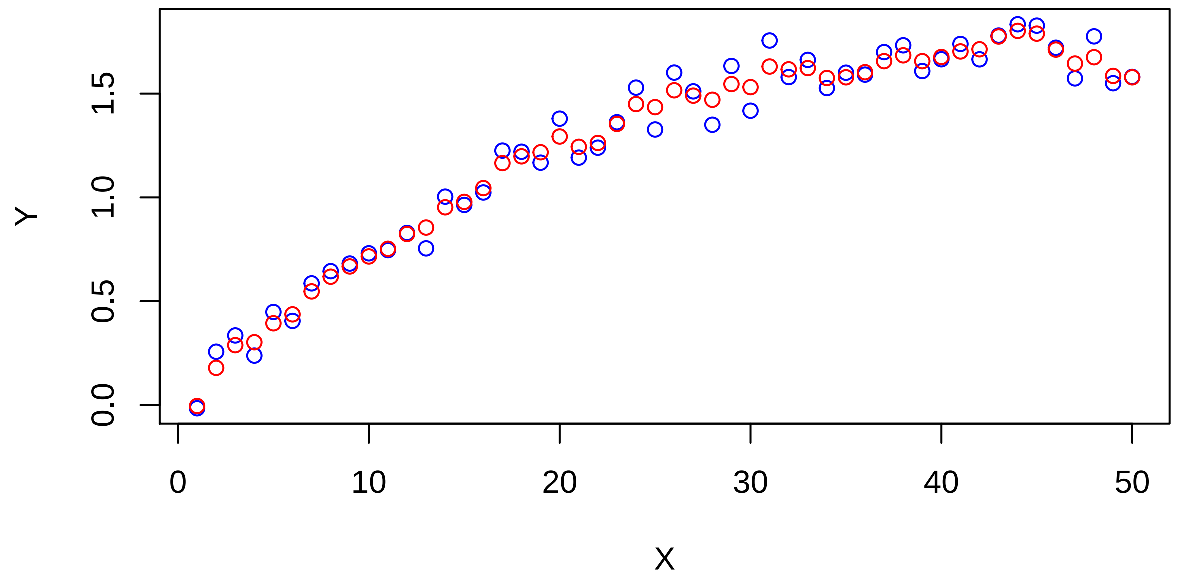
Correct

4. A Gibbs sampler

Using the ditributions derived in Step 3, we implement the Gibbs sampler that uses $\vec{u}^0 = (0, ..., 0)$ as a starting point.

```r
Gibbs_sampler <- function(tmax=10,data=Y){
  d=length(data)
  u_sample=matrix(0,nrow = tmax,ncol = d)
  for (i in 2:tmax) {
    u_old=u_sample[i-1,]
    u_new=rep(0,d)
    u_new[1]=rnorm(1,mean=(u_new[2]+data[1])/2,sd=sqrt(0.1))
    for (j in 2:(d-1)) {
      u_new[j]=rnorm(1,mean = (u_new[j-1]+u_old[j+1]+data[j])/3,sd=sqrt(0.2/3))
    }
    u_new[d]=rnorm(1,mean = (u_new[d-1]+data[d])/2,sd=sqrt(0.1))
    u_sample[i,]=u_new
  }
  return(u_sample)
}
sample_u=Gibbs_sampler(tmax = 1000)
```

We run the Gibbs sampler to obtain 1000 values of $\vec{u}$ and then compute the expected value of $\vec{u}$ by using a Monte Carlo approach, the average of 1000 samples. Then we plot the expected value of $\vec{u}$ and data Y in the same graph, red points for expected values while blue points for data Y.

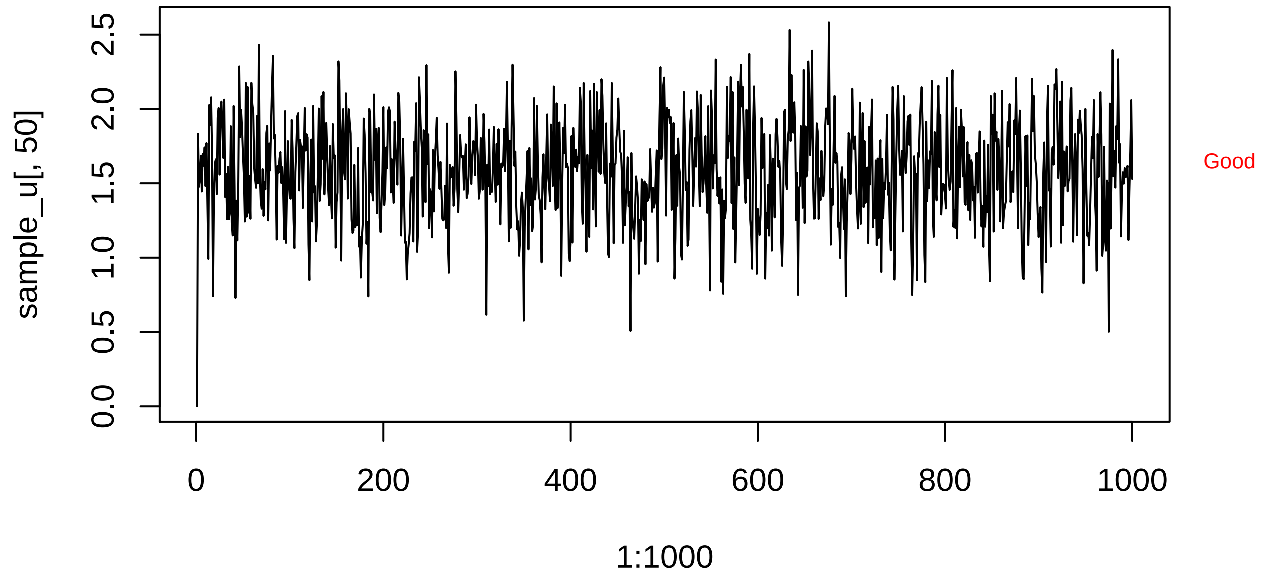


Good

The red points in the graph are more smoothing than the blue points and red points are more closer to each other than the blue points are. It seems that we have managed to remove the noise and the expected value of u catches the true underlying dependence between Y and X.

5. A trace plot for u_n

We illustrate the trace plot for u_n below.



This figure displays a perfect trace plot. There is almost no burn-in period and convergence is obvious since the chain has reached stationarity.

The chain traverses the posterior space rapidly, and it can jump from a remote region of the posterior to another in relatively few steps. (This comment comes from website: https://astro.uni-bonn.de/~kbasu/ObsCosmo/Slides2012/Lecture3_2012.pdf)

Appendix

```r
set.seed(12345)

# Question 1 Computations with Metropolis-Hastings

## 1. Use Metropolis-Hastings algorithm to generate samples

MetroHast <- function(nstep,x0){
  sequen_x=vector(length = nstep)
  sequen_x[1]=x0
  for (i in 2:nstep) {
    x=sequen_x[i-1]
    y=rlnorm(1,meanlog = x,sdlog = 1)
    u=runif(1)
    alpha=(y^5*exp(1)^-y)*(dlnorm(x,meanlog = y,
                    sdlog = 1))/((x^5*exp(1)^-x)*dlnorm(y,
                                  meanlog = x,sdlog = 1))
    a=min(1,alpha)
    sequen_x[i]=ifelse(u<=a,y,x)
  }
  return(sequen_x)
}

chain1=MetroHast(1000,0.1)
plot(1:1000,chain1,pch=19,cex=0.3,
     col="black",xlab="t", ylab="X(t)",
     ylim = c(0,10),
     main="The chain of the samples from f(x)")

## 2. Use Metropolis-Hastings algorithm to generate samples- 2

MetroHast2 <- function(nstep,x0){
  sequen_x=vector(length = nstep)
  sequen_x[1]=x0
  for (i in 2:nstep) {
    x=sequen_x[i-1]
    y=rchisq(1,df=floor(x+1))
    u=runif(1)
    alpha=(y^5*exp(1)^-y)*dchisq(x,df= floor(y+1))/((x^5*exp(1)^-x)*dchisq(y,
                            df = floor(x+1)))
    a=min(1,alpha)
    sequen_x[i]=ifelse(u<=a,y,x)
  }

  return(sequen_x)
}

chain2=MetroHast2(1000,1)
 plot(1:1000,chain2,pch=19,cex=0.3,
      col="black",xlab="t", ylab="X(t)",
      ylim = c(0,30),
      main="The chain of the samples from f(x)")
```

```r
## 4. Gelman-Rubin method to analyze convergence
library("coda")

f3=mcmc.list()
for (i in 1:10) {
  f3[[i]]=as.mcmc(MetroHast2(1000,i))
}

print(gelman.diag(f3))

## 5. Estimate a definite integral

mean(chain1)
mean(chain2)


# Question 2 Gibbs sampling

## 1. Import the data and plot the dependence of Y on X

load("chemical.RData")
## variable X is the day of measurement
## variable Y is the measurement
plot(X,Y)

## 4.Gibbs sampler

Gibbs_sampler <- function(tmax=10,data=Y){
  d=length(data)
  u_sample=matrix(0,nrow = tmax,ncol = d)
  for (i in 2:tmax) {
    u_old=u_sample[i-1,]
    u_new=rep(0,d)
    u_new[1]=rnorm(1,mean=(u_new[2]+data[1])/2,sd=sqrt(0.1))
    for (j in 2:(d-1)) {
      u_new[j]=rnorm(1,mean = (u_new[j-1]+u_old[j+1]+data[j])/3,sd=sqrt(0.2/3))
    }
    u_new[d]=rnorm(1,mean = (u_new[d-1]+data[d])/2,sd=sqrt(0.1))
    u_sample[i,]=u_new
  }
  return(u_sample)
}

sample_u=Gibbs_sampler(tmax = 1000)

expec_u=colMeans(sample_u)
plot(X,Y,col="blue",type = 'l')
points(X,expec_u,col="red",type = 'l')

## 5. a trace plot for u_n

plot(1:1000, sample_u[,50],type = "l")
```