

# Computer Lab 4 Computational Statistics

*Phillip Hölscher & Zijie Feng*

*8-2-2019*

## Contents

<b>Question 1: Computations with Metropolis-Hastings</b>	<b>2</b>
1. Use Metropolis-Hastings algorithm . . . . .	2
2. Perform Step 1 by using the chi-square distribution . . . . .	4
3. Compare the results of Steps 1 and 2 and make conclusions. . . . .	5
4. Generate 10 MCMC sequences . . . . .	6
5. Estimate . . . . .	6
6. In fact . . . . .	7
<b>Question 2: Gibbs sampling</b>	<b>7</b>
1. Import the data to R and plot the dependence of Y on X. . . . .	8
2. Present the formulae showing the likelihood and the prior: . . . . .	8
<b>Appendix</b>	<b>10</b>

## Question 1: Computations with Metropolis-Hastings

Consider the following probability density function:

$$f(x) \propto x^5 e^{-x}, \quad x > 0$$

You can see that the distribution is known up to some constant of proportionality. If you are interested (NOT part of the Lab) this constant can be found by applying integration by parts multiple times and equals 120.

### 1. Use Metropolis–Hastings algorithm

to generate samples from this distribution by using proposal distribution as log-normal  $\text{LN}(X_t, 1)$ , 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?

```
rm(list=ls())
set.seed(123456)
#pdf - probability density function
pdf_target = function(x){
  if(x<= 0){
    stop("x needs to be bigger than 0")
  }
  # stopifnot(x>0) # check - x >0
  return(x^5 * exp(-x))
}
```

Below you can see the code for the Metropolis-Hastings algorithm. We decided to initialize the starting point ( $X_0$ ) with 1 and use 10000 for  $t$ , as max iterations.

```
# Metropolis-Hastings Sampler

# Initilize chain to  $X_0$ ,  $t=0$ 
t_max = 10000
x_0 = 1 # starting point

# algorithm from the slide
MCMC <- function(t_max, x_0){
  rej = 0
  # browser()
  x_t = rep(x_0, t_max) # vector to save y or  $x_t$ 
  for (t in 2: t_max){
    X = x_t[t-1]
    Y = rlnorm(n = 1, meanlog = log(X), sdlog = 1)
    u = runif(1,0,1)
    num = pdf_target(Y) * dlnorm(X, meanlog = log(Y), sdlog = 1)
    den = pdf_target(X) * dlnorm(Y, meanlog = log(X), sdlog = 1)
    alpha = min(1, num/den)
    if(u < alpha){
      x_t[t] = Y
    } else{
      # reject the Y when is FALSE
      # then we keep  $x_{t-1}$  as  $x_t$ 
      x_t[t] = X
      rej = rej+1
    }
  }
}
```

```

    }
  }
  cat("Reject rate: ", rej/t_max, "\n")
  return(data.frame(vN=1:t_max, vX=x_t))
}

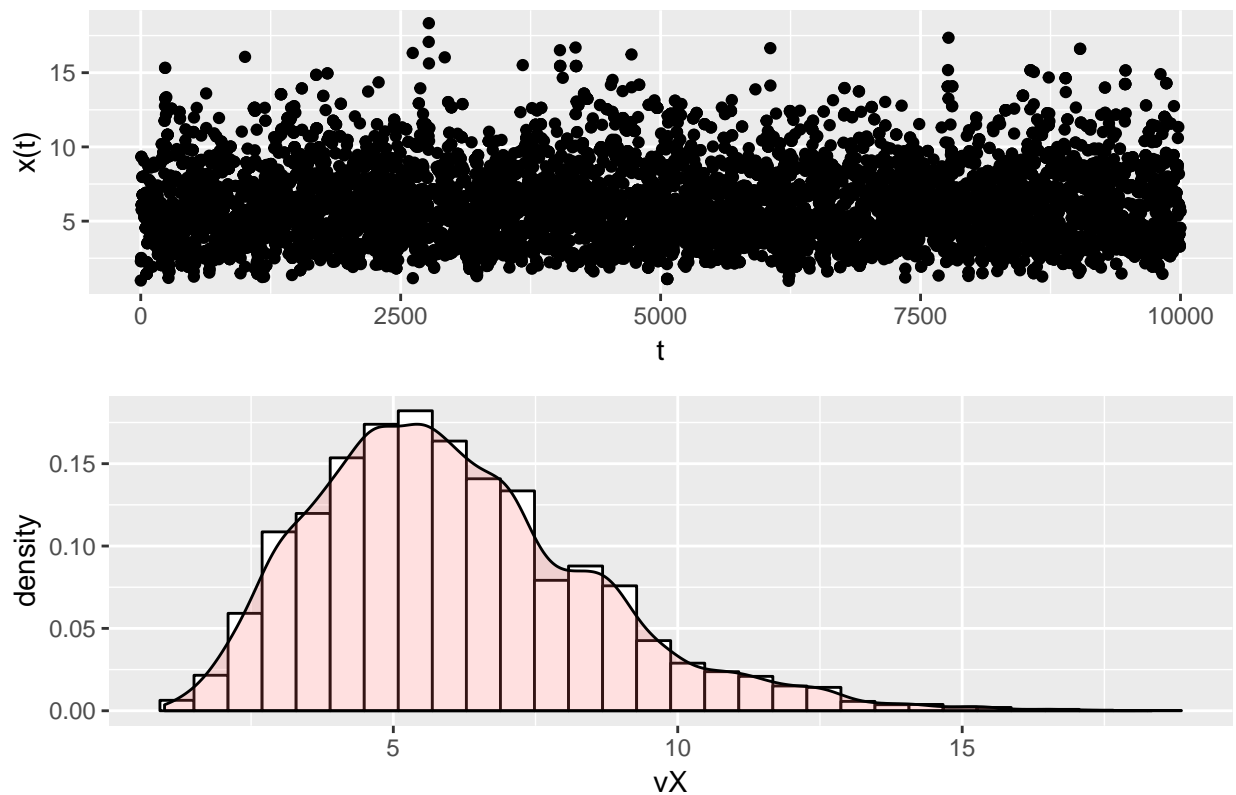
```

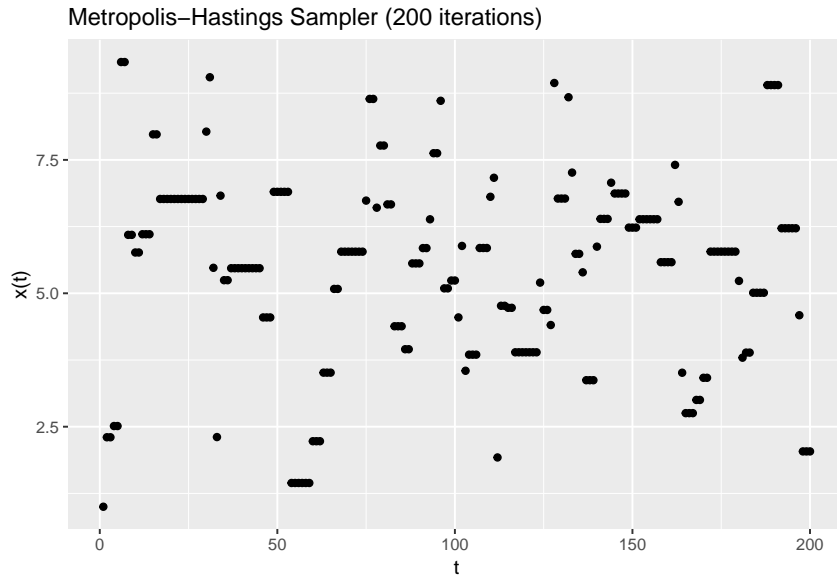
```
data_1_1 = MCMC(t_max, x_0)
```

```
## Reject rate: 0.5611
```

Plot of the chain obtained as a time series plot:

Metropolis–Hastings Sampler 10000 iterations





The sample points seem to be convergent around 2.5 and 5. We also draw another plot with 200 iterations to find the burn-in period, but it seems that burn-in period is so short (from 0 to 10).

## 2. Perform Step 1 by using the chi-square distribution

$\chi^2 (\lfloor X_t + 1 \rfloor)$  as a proposal distribution, where  $\lfloor x \rfloor$  is the floor function, meaning the integer part of  $x$  for positive  $x$ , i.e.  $\lfloor 2.95 \rfloor = 2$

Metropolis–Hastings algorithm with chi-square as proposal distribution

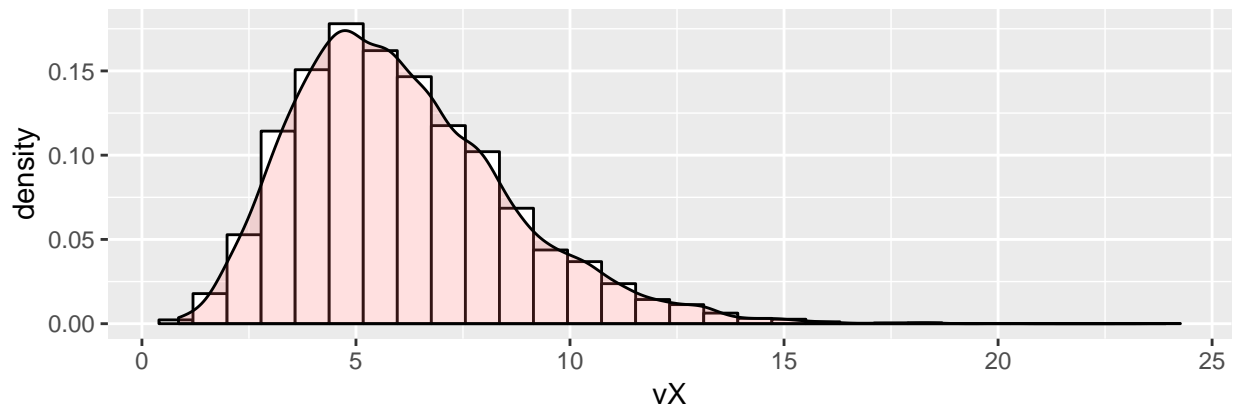
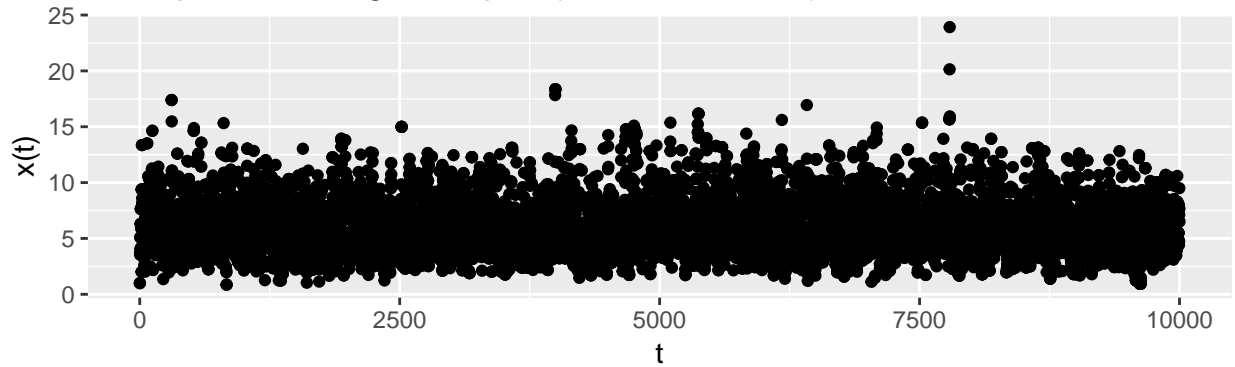
```
MCMC2 <- function(t_max, x_0){
  rej = 0
  x_t = rep(x_0, t_max) # vector to save y or x_t
  for (t in 2: t_max){
    X = x_t[t-1]
    Y = rchisq(1,df=floor(X+1))
    u = runif(1,0,1)
    num = pdf_target(Y)*dchisq(X,floor(Y+1))
    den = pdf_target(X)*dchisq(Y,floor(X+1))

    alpha = min(1, num/den)
    if(u < alpha){
      x_t[t] = Y
    } else{
      # reject the Y when is FALSE
      # then we keep x_{t-1} as x_{t}
      x_t[t] = X
      rej = rej+1
    }
  }
  cat("Reject rate: ",rej/t_max,"\n")
  return(data.frame(vN=1:t_max, vX=x_t))
}

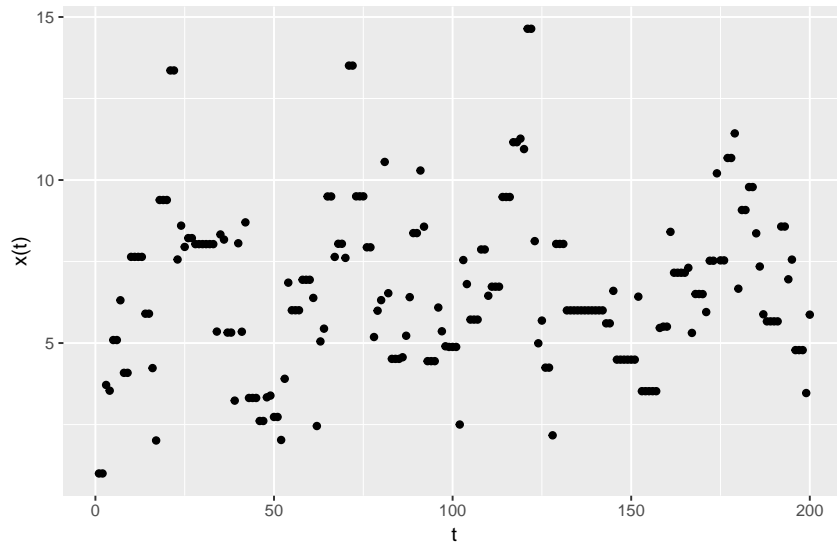
data_1_2 = MCMC2(t_max, x_0)
```

## Reject rate: 0.4005

Metropolis–Hastings Sampler (10000 iterations)



Metropolis–Hastings Sampler (200 iterations)



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

In step 2, the MH method based on chi-square distribution has similar burn-in period (0~10) and convergent interval (2.5~15). However, the rejection rate of the method in step 2 is around 0.4, but in step 1 is more

than 0.5, which means that chi-square distribution might be a better proposal distribution than log-normal distribution to generate the samples of our target distribution.

## 4. Generate 10 MCMC sequences

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

```
# Generation 10 MCMC sequences with start points 1~10
```

```
set.seed(123456)
df <- data.frame(vN=1:t_max)
for (i in 1:10) {
  df <- cbind(df, MCMC2(t_max, i)$vX)
}
```

```
## Reject rate: 0.3988
## Reject rate: 0.4042
## Reject rate: 0.3955
## Reject rate: 0.3852
## Reject rate: 0.3997
## Reject rate: 0.4016
## Reject rate: 0.407
## Reject rate: 0.4011
## Reject rate: 0.4001
## Reject rate: 0.3959
```

```
mcmc_list = list()

for (i in 1:10){
  mcmc_list[[i]] = as.mcmc(df[,i+1], start = i)
}
```

```
# Gelman-Rubin method:
gelman.diag(mcmc_list)
```

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

Since both potential scale reduction factor and their upper confidence limits are 1's, we can ensure that our samples are convergent.

## 5. Estimate

$$\int_0^\infty x f(x) dx$$

using the samples from Steps 1 and 2.

Down here follows the rule on how to calculate a definite integral based on lecture slide.

To estimate a  $\theta$  which is

$$\theta = \int_D f(x) dx,$$

we can decompose such  $f(x)$  into:

$$f(x) = g(x)p(x),$$

where

$$\int_D p(x)dx = 1.$$

- Thus, if  $X \sim p(\cdot)$  and

$$\theta = E[g(X)] = \int_D g(x)p(x)dx,$$

we can conclude that

$$\hat{\theta} = \frac{1}{n} \sum_i^n g(x_i), \forall_i \sim p(\cdot)$$

Our target function is a probability density function, whose integral must be 1. Since we already have the samples from the target distribution by Metropolis-Hastings algorithm, the final integral of  $xf(x)$  can be estimated as the mean of such samples.

**Mean of sample 1:**

```
## [1] 6.061493
```

Mean of sample 2:

```
## [1] 6.085729
```

## 6. In fact

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

PDF of gamma distribution  $X \approx \Gamma(\alpha, \beta)$ :

$$f(x; \alpha, \beta) = \frac{\beta^\alpha x^{\alpha-1} e^{-\beta x}}{\Gamma(\alpha)}$$

for  $x > 0$ , and  $\alpha, \beta > 0$ .

Based on the standard formula, we know that  $\alpha = 6$  and  $\beta = 1$  in our target PDF. Thus, we can calculate the mean of our target PDF as

$$E[X] = \frac{\alpha}{\beta} = \frac{6}{1} = 6,$$

which closes to our estimated results.

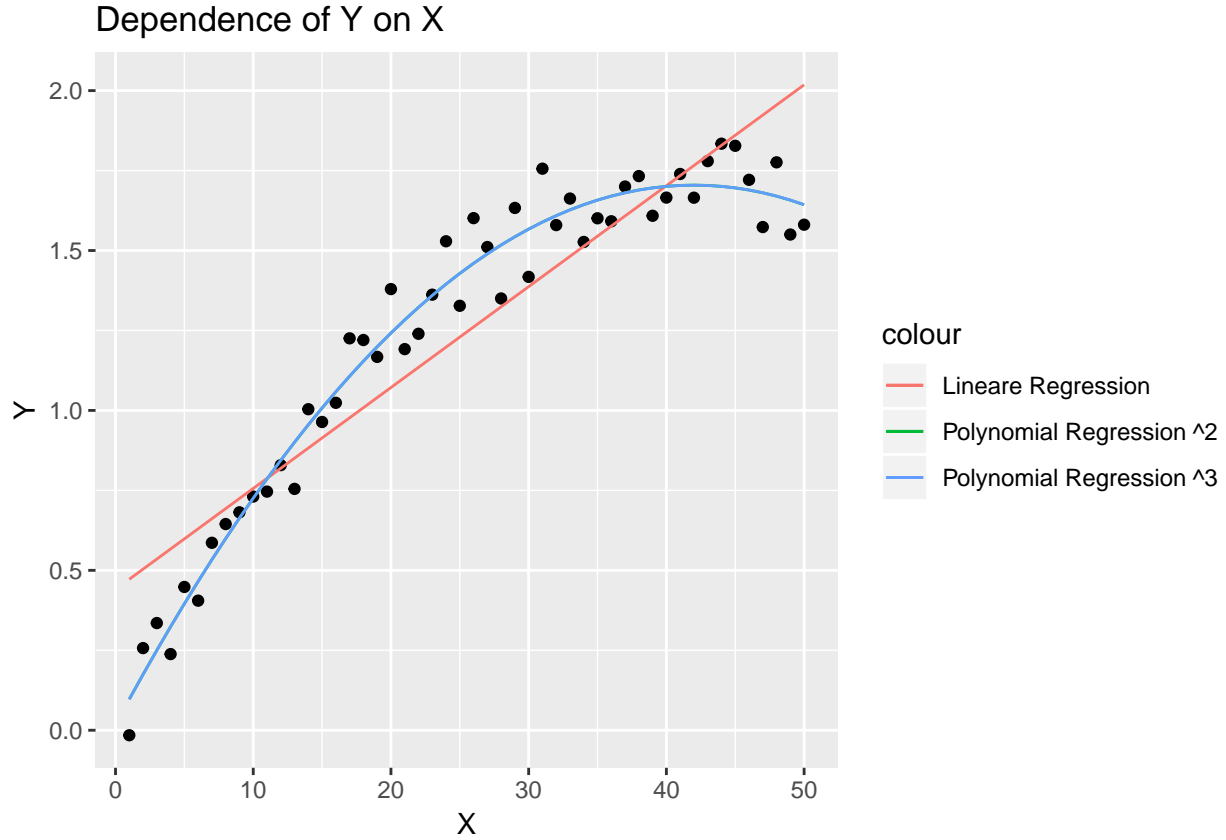
## Question 2: Gibbs sampling

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?



It could already be seen that a linear regression does not describe the data too much. However, a second degree polynomial regression is well consistent with the course of the data. As expected can be seen that the progression of polynomials 2 and 3 overlap, so a second grade is sufficient.

2. A researcher has decided to use the following (random-walk) Bayesian model ( $n$ =number of observations,  $\vec{\mu} = (\mu_1, \dots, \mu_n)$  are unknown parameters):

$$Y_i = \mathcal{N}(\mu, \sigma = 0.2), i = 1, \dots, n$$

where the prior is

$$p(\mu_1) = 1$$

$$p(\mu_{i+1} | \mu_i) = \mathcal{N}(\mu_i, 0.2) i = 1, \dots, n-1$$

## 2. Present the formulae showing the likelihood and the prior:

$p(\vec{Y} | \vec{\mu})$  and  $p(\vec{\mu})$ .

Likelihood:

$$\begin{aligned} \mathcal{L}(y_1, \dots, y_n | \mu_i, 0.2) &= \prod_{i=1}^n \frac{1}{\sqrt{2\pi} \cdot 0.2} \exp\left(-\frac{(y_i - \mu_i)^2}{2 \cdot 0.2^2}\right) \\ &= \left(\frac{1}{\sqrt{0.08\pi}}\right)^n \exp\left(-\frac{\sum_{i=1}^n (y_i - \mu_i)^2}{0.08}\right) \end{aligned}$$



Prior:

$$\begin{aligned} p(\vec{\mu}) &= \prod_{i=1}^{n-1} \frac{1}{\sqrt{2\pi}0.2^2} \exp\left(-\frac{(\mu_{n+1} - \mu_i)^2}{2 * 0.2^2}\right) \\ &= \left(\frac{1}{\sqrt{0.08\pi}}\right)^{n-1} \exp\left(-\frac{\sum_{i=1}^n (\mu_{n+1} - \mu_n)^2}{0.08}\right) \end{aligned}$$

## Appendix

```
knitr::opts_chunk$set(echo = TRUE, out.width = "480px")
# libraries used in this lab
library(ggplot2)
# install.packages("coda")
library(coda)
library(gridExtra)
# information & material
# very nice blog - MCMC and the Metropolis-Hastings algorithm
# https://blog.stata.com/2016/11/15/introduction-to-bayesian-statistics-part-2-mcmc-and-the-metropolis-h
rm(list=ls())
set.seed(123456)
# pdf - probability density function
pdf_target = function(x){
  if(x <= 0){
    stop("x needs to be bigger than 0")
  }
  # stopifnot(x > 0) # check - x > 0
  return(x^5 * exp(-x))
}
# Metropolis-Hastings Sampler

# Initialize chain to X_0, t=0
t_max = 10000
x_0 = 1 # starting point

# algorithm from the slide
MCMC <- function(t_max, x_0){
  rej = 0
  # browser()
  x_t = rep(x_0, t_max) # vector to save y or x_t
  for (t in 2: t_max){
    X = x_t[t-1]
    Y = rlnorm(n = 1, meanlog = log(X), sdlog = 1)
    u = runif(1, 0, 1)
    num = pdf_target(Y) * dlnorm(X, meanlog = log(Y), sdlog = 1)
    den = pdf_target(X) * dlnorm(Y, meanlog = log(X), sdlog = 1)
    alpha = min(1, num/den)
    if(u < alpha){
      x_t[t] = Y
    } else{
      # reject the Y when is FALSE
      # then we keep x_{t-1} as x_{t}
      x_t[t] = X
      rej = rej+1
    }
  }
}
cat("Reject rate: ", rej/t_max, "\n")
return(data.frame(vN=1:t_max, vX=x_t))
}

data_1_1 = MCMC(t_max, x_0)
```

```

# create time series plot

# create the plot object
points11= ggplot(data = data_1_1, aes(x= vN, y= vX)) +
  geom_point()+
  ggtitle("Metropolis-Hastings Sampler 10000 iterations")+
  ylab("x(t)")+
  xlab("t")

density11= ggplot(data_1_1,aes(x=vX))+
  geom_histogram(aes(y=..density..),
    colour="black",
    fill="white",
    bins=30)+
  geom_density(alpha=.2, fill="#FF6666")

grid.arrange(points11,density11,nrow=2)
ggplot(data = data_1_1[1:200,], aes(x= vN, y= vX)) +
  geom_point()+
  ggtitle("Metropolis-Hastings Sampler (200 iterations)")+
  ylab("x(t)")+
  xlab("t")

MCMC2 <- function(t_max, x_0){
  rej = 0
  x_t = rep(x_0, t_max) # vector to save y or x_t
  for (t in 2: t_max){
    X = x_t[t-1]
    Y = rchisq(1,df=floor(X+1))
    u = runif(1,0,1)
    num = pdf_target(Y)*dchisq(X,floor(Y+1))
    den = pdf_target(X)*dchisq(Y,floor(X+1))

    alpha = min(1, num/den)
    if(u < alpha){
      x_t[t] = Y
    } else{
      # reject the Y when is FALSE
      # then we keep x_{t-1} as x_{t}
      x_t[t] = X
      rej = rej+1
    }
  }
  cat("Reject rate: ",rej/t_max,"\n")
  return(data.frame(vN=1:t_max, vX=x_t))
}

data_1_2 = MCMC2(t_max, x_0)
# create the plot - t_max = 10.000

# create the plot object
points12= ggplot(data = data_1_2, aes(x= vN, y= vX)) +
  geom_point()+

```

```

ggtitle("Metropolis-Hastings Sampler (10000 iterations)") +
ylab("x(t)") +
xlab("t")

density12= ggplot(data_1_2,aes(x=vX)) +
  geom_histogram(aes(y=..density..),
    colour="black",
    fill="white",
    bins=30) +
  geom_density(alpha=.2, fill="#FF6666")

grid.arrange(points12,density12,nrow=2)

# create the plot object
ggplot(data = data_1_2[1:200,], aes(x= vN, y= vX)) +
  geom_point() +
  ggtitle("Metropolis-Hastings Sampler (200 iterations)") +
  ylab("x(t)") +
  xlab("t")

# Generation 10 MCMC sequences with start points 1~10
set.seed(123456)
df <- data.frame(vN=1:t_max)
for (i in 1:10) {
  df <- cbind(df,MCMC2(t_max,i)$vX)
}
mcmc_list = list()

for (i in 1:10){
  mcmc_list[[i]] = as.mcmc(df[,i+1], start = i)
}

# Gelman-Rubin method:
gelman.diag(mcmc_list)
mean_set1 = mean(data_1_1$vX)
mean_set1
mean_set2 = mean(data_1_2$vX)
mean_set2
rm(list=ls())
set.seed(123456)
# set working directory
# import the data
load("chemical.RData")
# create table for plot
data = data.frame("X" = X,
  "Y" = Y)
# plot dependence of Y on X
X_Y_dependence = ggplot(data, aes(x = X, y = Y)) +
  geom_point() +
  ggtitle("Dependence of Y on X") +
  geom_smooth()
# which model would fit the data
# looks like a polynomial regression -> test until poly 3
modell1 = lm(Y ~ X,

```

```

      data = data)
model2 = lm(Y ~ X + I(X^2),
      data = data)
model3 = lm(Y ~ X + I(X^2) + I(X^3),
      data = data)

cols <- c("Lineare Regression"="#62c76b",
      "Polynomial Regression ^2"="#3591d1",
      "Polynomial Regression ^3"="#f04546")

X_Y_dependence_fit = ggplot(data, aes(x = X, y = Y)) +
  geom_point() +
  ggtitle("Dependence of Y on X") +
  geom_line(aes(x = X, y = predict(model1), colour = "Lineare Regression")) +
  geom_line(aes(x = X, y = predict(model2), colour = "Polynomial Regression ^2")) +
  geom_line(aes(x = X, y = predict(model3), colour = "Polynomial Regression ^3"))
X_Y_dependence_fit

```