Computational Statistics
732A90

Lab 4

Albin Västerlund
albva223

Eric Herwin
Erihe068

Contents

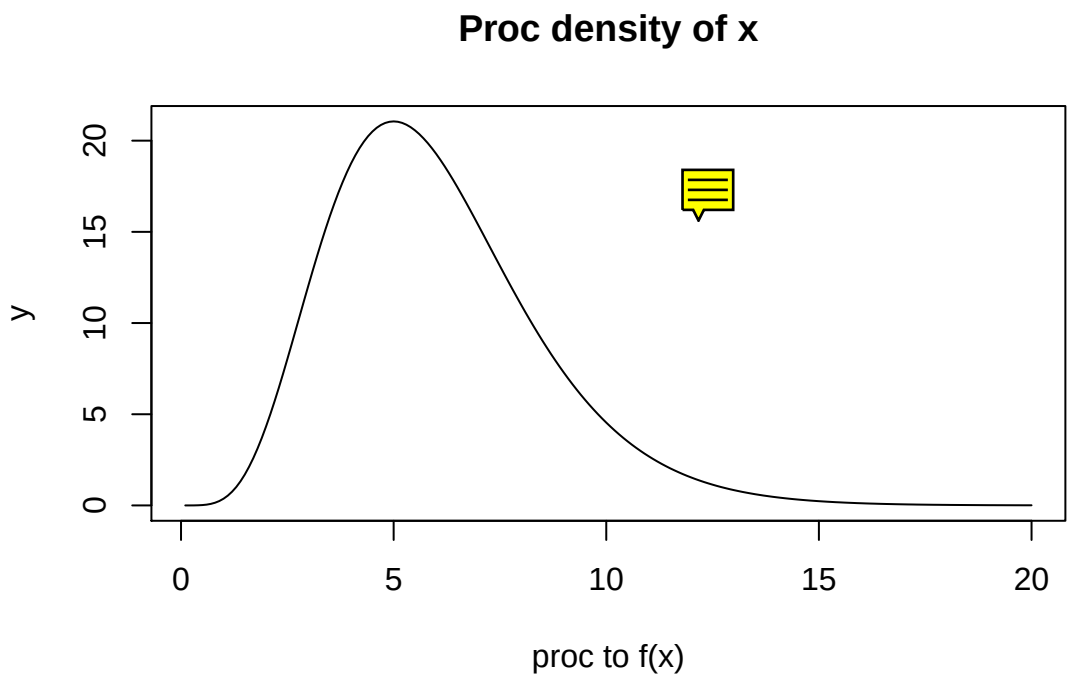
Assigment 1: Computations with Metropolis-Hastings

1

In this assigment we will use the Metropolis-Hastings algorithm to generate samples from $f(x) \propto x^5 e^{-x}$ distribution usning the the $LN(X_t, 1)$ as our proposal distribution.

But we will start to plot the PDF of the target distribution just to get a hint of what we will expect from the assigment.

```
x<-seq(0.1,20,0.1)
f_x <- function(x){
  x^5 * exp(-x)
}
y<-f_x(x)
plot(x,y,type="l",xlab="proc to f(x)",main = "Proc density of x")
```



It seems like we will expect a mode on about 5 and the a mean around 6.

We will therefore set a starting point thats a lot bigger then 5-6 so we can see the chain.

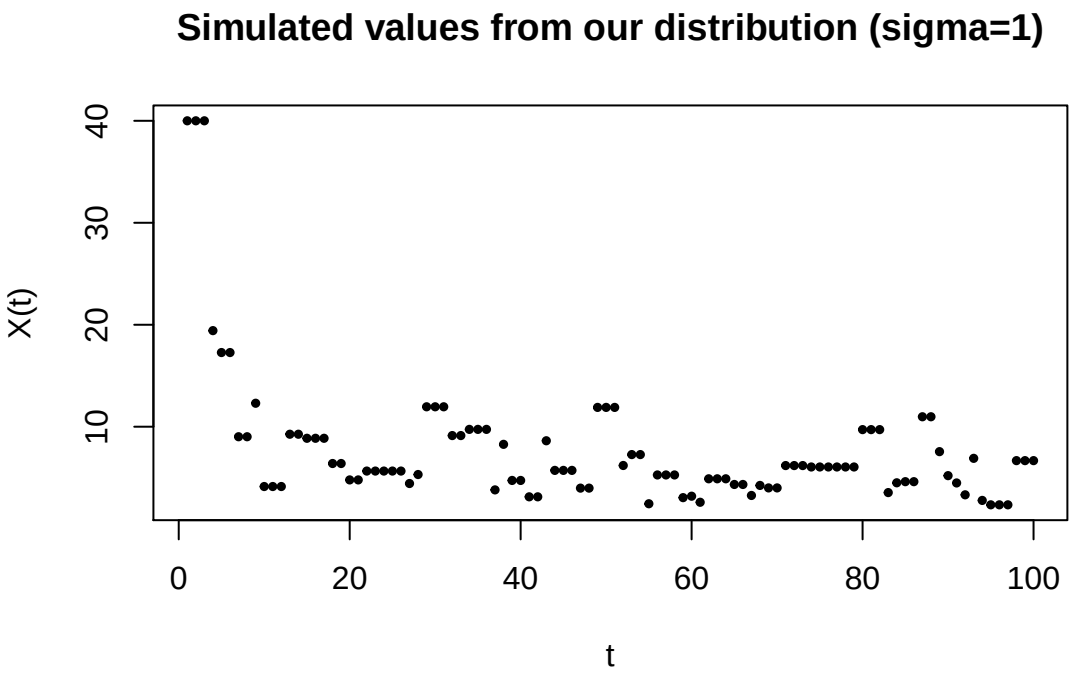
We start to do a function called `MCMC_function` that use the Metropolis-Hastings algorithm to simulate values from the target distribution given three function, a starting point and number of wanted values.

```
MCMC_function<-function(nstep,X0){
  vN<-1:nstep          # values from 1 to nstep
  vX<-rep(X0,nstep)    # Our saved X values
  for (i in 2:nstep){
```

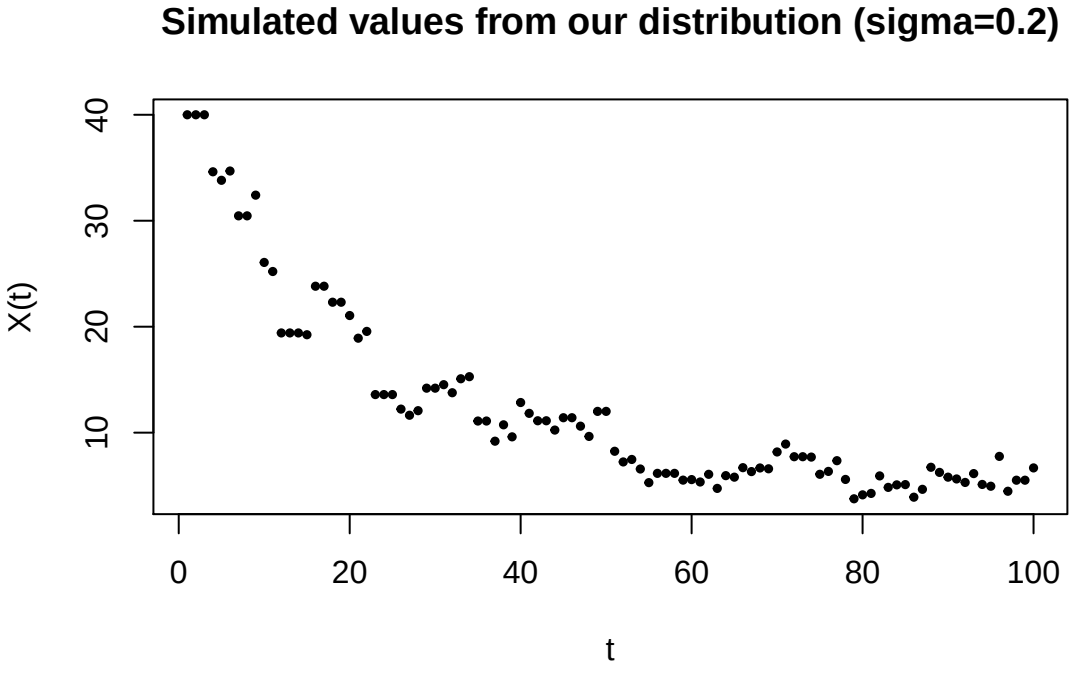
```
    X<-vX[i-1]


    Y<-Sim_Y_from_q_given_X(X)          #Sim Y~ q(.|X_t)
    u<-runif(1)                         #Sim u~ unif(0,1)

    # Calc alpha ##########
    nr1<-f_x(Y)
    nr2<-q_x_given_y(X,Y)

    nr3<-f_x(X)
    nr4<-q_x_given_y(Y,X)

    alpha<-1
    alpha[2] <-(nr1*nr2)/(nr3*nr4)
    alpha<-min(alpha)


    # Store or throw our Y value ####

    if (u <=alpha){
      vX[i]<-Y}
    else{
      vX[i]<-X}
  }
  #plot(vN,vX,pch=19,cex=0.3,col="black",xlab="t",ylab="X(t)",main="")

  return(vX)

}
```

Now when we have a function that can simulate values from the target distribution we want to use it. But first we need to specify the three functions $f(t)$, $q(t|z)$ and a function that simulate values from $y \sim q(\cdot|y)$. We also specify the starting point to 40. We will then plot the values in a scatterplot in the order they where simulated.

```
q_x_given_y <- function(x,y){           # pdf q(x|y)
  dlnorm(x,meanlog = log(y), sdlog = 1)
}
f_x <- function(x){                     # pdf of target
  x^5 * exp(-x)
}
Sim_Y_from_q_given_X<-function(X){       # sim y~q(.|y)
  rlnorm(1,meanlog = log(X),1)
}

set.seed(123456789)
x_from_target<-MCMC_function(100,40)    # sim from target
x_from_target_ass_1_1<-x_from_target    # Save values for 1.5

plot(x_from_target,pch=19,cex=0.5,col="black",xlab="t",ylab="X(t)",main="Simulated values from our dist
```



When we usning the $\sigma^2 = 1$ in the $LN(X_t, \sigma^2)$ its hard to see the chain. So we will also plot simulated values thats usning $LN(X_t, 0.2)$ as our proposal distribution just to see the difference.

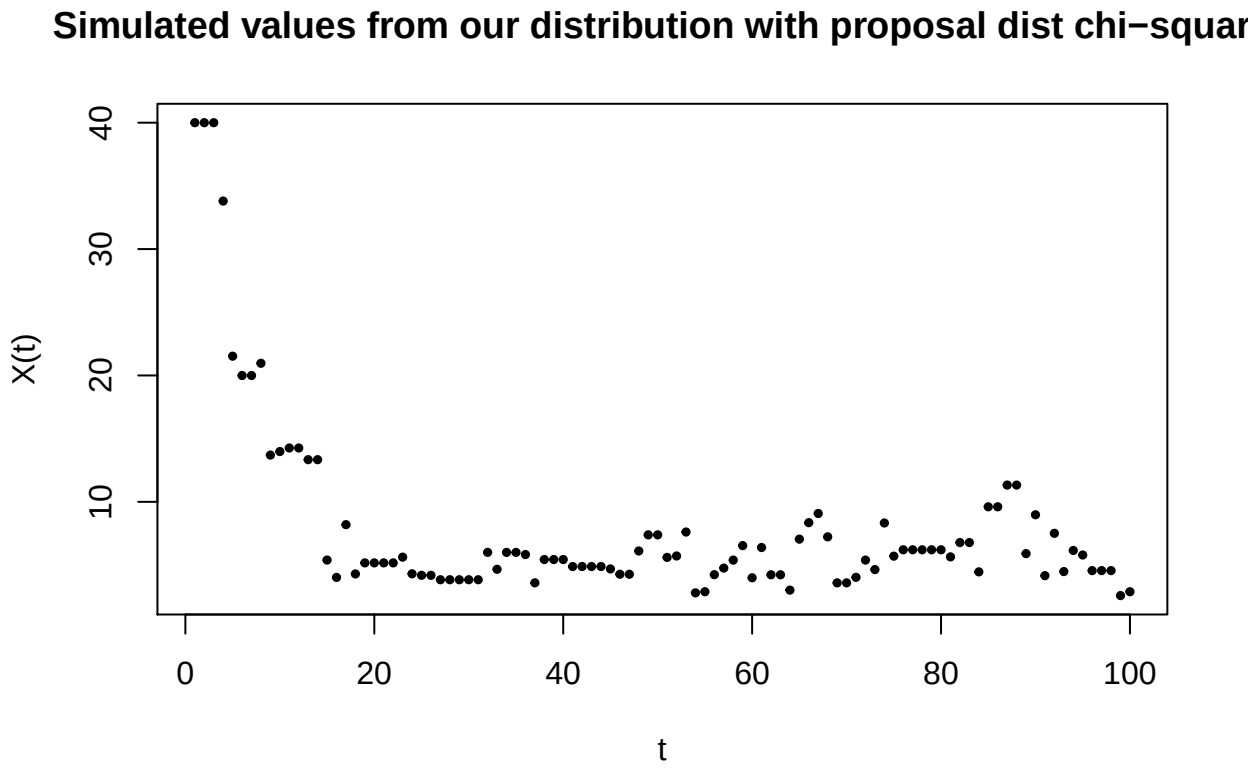


In this plot its "easy" to see that the values converge into the target distribution $y \sim f(x)$ thats have a mode on around 5 and a mean around 6. The converge happens after approximately 60 simulated values.

2

In this assigment we will do the same thing as in the assiment 1.1. But in this assigment we will use $\chi^2_{x_t+1}$ as our proposal distribution. We specify the starting point to 40 so we see the chain.

```
q_x_given_y <- function(x,y){          # pdf q(x|y)

  dchisq(x,df = floor(y+1))
}
f_x <- function(x){                    # pdf of target
  x^5 * exp(-x)
}
Sim_Y_from_q_given_X<-function(x){      # sim y~q(.|y)
  rchisq(1,df = floor(x+1))
}
set.seed(123456789)

x_from_target<-MCMC_function(100,40)   # sim from target
x_from_target_ass_1_2<-x_from_target   # Save values for 1.5


plot(x_from_target,pch=19,cex=0.5,col="black",xlab="t",ylab="X(t)",main="Simulated values from our dist
```



We get a similar result as in 1.1. The values converge into the target distribution $y \sim f(x)$ after approximately 20 simulated values.

3

Both the $LN(X_t, 1)$ and $\chi^2_{x_t+1}$ is a fairly good proposal distribution to our wanted distribution (it converge to a intervall). Both the proposal distribution simulates new potential Y that we dont want to use because thay are so extreme for our wanted distribution. Thats why its the same simulated values in a row. When we use $LN(X_t, 0.2)$ we can see that we dont get that many values thats are the same in a row. Thats because the variance is smaller and therefore we dont simulate values that is extrem for our wanted distribution so often. But the values close in time to each other are more correlated in some sense.

4

In this assigment we will generate 10 MCMC sequences using the generator $\chi^2_{x_t+1}$ as proposal distribution with starting point 1...19. In each sequences we will simulate 1000 values. We will then use Gelman-Rubin method to analyze convergence of these sequences.

We start to study the formulas.

$$\sqrt{R} = \sqrt{\frac{\hat{Var}(v)}{W}}$$

$$\hat{Var}(v) = \frac{n-1}{n}W + \frac{1}{n}B$$

$$B = \frac{n}{k-1}\sum_{i=1}^{k}(\bar{v}_{i\cdot} - \bar{v}_{\cdot\cdot})$$

$$W = \sum_{i=1}^{k}\frac{s_i^2}{k}$$

$$s_i^2 = \sum_{j=1}^{n}\frac{\bar{v}_{ij} - \bar{v}_{i\cdot}}{n-1}$$

Where k(=10) is the number of generated sequences and n(=1000) number of observations in eatch sequences.

Now when we have the formulas we can calculate Gelman-Rubin.

```
set.seed(123456789)

s_i<-c()
mean_i<-c()
n_l<-1000
k<-10
x_from_target<-list()
for(i in 1:k){
  x_from_target[[i]]<-MCMC_function(n_l,i)  # sim from target
  s_i[i]<-var(x_from_target[[i]])
  mean_i[i]<-mean(x_from_target[[i]])
}
```

```r
W <- mean(s_i)

B <- n_l/(k-1) * sum((mean_i - mean(mean_i))^2)

var_v <- (n_l-1)/n_l * W + 1/n_l * B

R <- var_v/W

sqrt(R) #Gelman-Rubin
```

```
## [1] 1.001
```

The Gelman-Rubin factor is very close to 1. Witch indicates that we not suffer of lack of convergence.

5

$$E(x) = \int_0^\infty x f(x) dx \to \bar{x} = \frac{\sum x_i}{n}$$

```r
mean(x_from_target_ass_1_1) #Estimate mean assigment 1.1
```

```
## [1] 7.699
```

```r
mean(x_from_target_ass_1_2) #Estimate mean assigment 1.2
```

```
## [1] 7.943
```

6

The distribution we have been handling with is actually a $gamma(\alpha = 6, \beta = 1)$. So the true value of the integral is:

$$E(x) = \int_0^\infty x f(x) dx = \frac{\alpha}{\beta} = 6$$

Our obtained values of the mean is around 1.8 to big. But we have onley simulated 100 values and the first 20 is a burning period. So if we would remove the first 20 values of increase n we would probably obtain a mean closer to the true mean value.

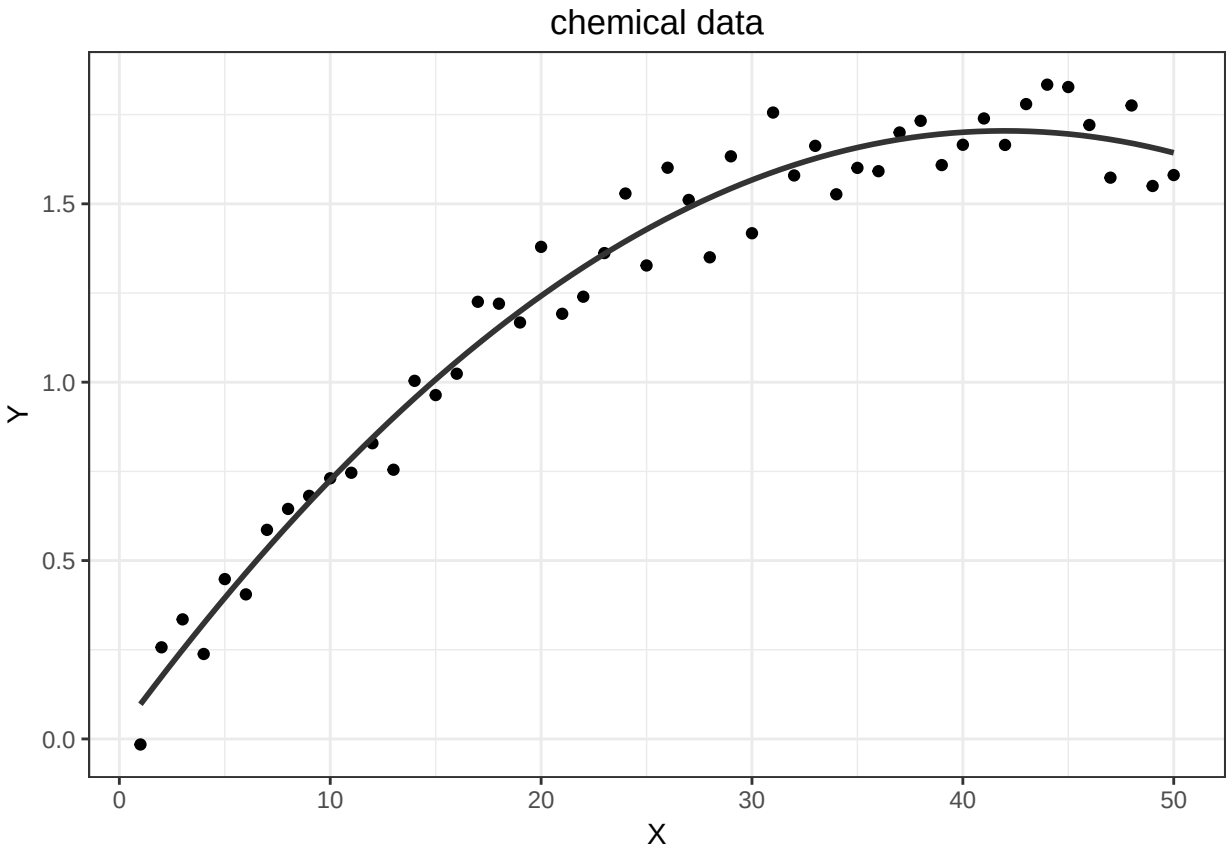
Assigment 2: Gibbs sampling

1

In this assigment we will import the dataset `chemical.RData` and plot the two variables.

```r
load("chemical.RData")

ggplot(mapping = aes(x=X,y=Y))+
  geom_point()+
```

```
  theme_bw()+
  ggtitle("chemical data")+
  theme(plot.title = element_text(hjust = 0.5))+
  geom_smooth(method = "lm", formula = y ~ poly(x,2), se = FALSE,colour="gray20")
```



It look like a linjer model $y \sim \beta_0 + \beta_1 X + \beta_2 X^2 + \varepsilon$ seems like a good model.

2

The prior is:

$$p(\vec{\mu}) = p(\mu_1) \cdot p(\mu_2|\mu_1) \cdot ... \cdot p(\mu_n|\mu_{n-1}) \propto e^{-\frac{\sum_{j=2}^{n}(\mu_j - \mu_{j-1})^2}{2\sigma^2}}$$

The likelihood is:

$$L(\vec{y}|\vec{\mu}) = p(y_1|\mu_1) \cdot ... \cdot p(y_n|\mu_n) \propto e^{-\frac{\sum_{j=1}^{n}(y_j - \mu_j)^2}{2\sigma^2}}$$

3

$$p(\mu_i|\vec{\mu_{-i}}, \vec{y}) \propto p(\mu_i) \cdot p(\vec{\mu_{-i}}|\mu_i) \cdot p(\vec{y}|\mu_i|)$$

$$p(\mu_i) = \prod_{j=2}^{i} p(\mu_j|\mu_{j-1}) \propto e^{-\frac{\sum_{j=2}^{i}(\mu_j-\mu_{j-1})^2}{2\sigma^2}}$$

$$p(\vec{\mu_{-i}}|\mu_i) = \prod_{j=i+1}^{n} p(\mu_j|\mu_{j-1}) \propto e^{-\frac{\sum_{j=i+1}^{n}(\mu_j-\mu_{j-1})^2}{2\sigma^2}}$$

$$p(\vec{y}|\mu_i) = p(y_1) \cdot ... \cdot p(y_i|\mu_i) \cdot ... \cdot p(y_n) \propto e^{-\frac{\sum_{j=1}^{n}(y_j-\mu_j)^2}{2\sigma^2}}$$

$$p(\mu_i) \cdot p(\vec{\mu_{-i}}|\mu_i) \cdot p(\vec{y}|\mu_i|) \propto e^{-\frac{\sum_{j=2}^{i}(\mu_j-\mu_{j-1})^2}{2\sigma^2}} \cdot e^{-\frac{\sum_{j=i+1}^{n}(\mu_j-\mu_{j-1})^2}{2\sigma^2}} \cdot e^{-\frac{\sum_{j=1}^{n}(y_j-\mu_j)^2}{2\sigma^2}} =$$

$$= exp\left(-\frac{1}{2\sigma^2}\left(\sum_{j=2}^{i}(\mu_j-\mu_{j-1})^2 + \sum_{j=i+1}^{n}(\mu_j-\mu_{j-1})^2 + \sum_{j=1}^{n}(y_j-\mu_j)^2 \right)\right) =$$

$$= exp\left(-\frac{1}{2\sigma^2}\left(\sum_{j=2}^{n}(\mu_j-\mu_{j-1})^2 + \sum_{j=1}^{n}(y_j-\mu_j)^2 \right)\right) =$$

$$exp\left(-\frac{1}{2\sigma^2}\left(\underbrace{\sum_{j=2}^{i-1}(\mu_j-\mu_{j-1})^2}_{constant} + (\mu_i-\mu_{i-1})^2 + (\mu_{i+1}-\mu_i)^2 + \underbrace{\sum_{j=i+2}^{n}(\mu_j-\mu_{j-1})^2}_{constant} + \underbrace{\sum_{j=1}^{i-1}(y_j-\mu_j)^2}_{constant} + (y_i-\mu_i)^2 + \underbrace{\sum_{j=i+1}^{n}(y_j-\mu_j)^2}_{constant} \right)\right) =$$

$$exp\left(-\frac{1}{2\sigma^2}\left(\underbrace{\sum_{j=2}^{i-1}(\mu_j-\mu_{j-1})^2}_{constant} + (\mu_i-\mu_{i-1})^2 + (\mu_{i+1}-\mu_i)^2 + \underbrace{\sum_{j=i+2}^{n}(\mu_j-\mu_{j-1})^2}_{constant} + \underbrace{\sum_{j=1}^{i-1}(y_j-\mu_j)^2}_{constant} + (y_i-\mu_i)^2 + \underbrace{\sum_{j=i+1}^{n}(y_j-\mu_j)^2}_{constant} \right)\right) =$$

$$= exp\left(-\frac{1}{2\sigma^2}\left((\mu_i-\mu_{i-1})^2 + (\mu_{i+1}-\mu_i)^2 + (y_i-\mu_i)^2 \right)\right)$$

$$= exp\left(-\frac{1}{2\sigma^2}\left((\mu_i-\mu_{i-1})^2 + (\mu_i-\mu_{i+1})^2 + (\mu_i-y_i)^2 \right)\right) = /use\ hint\ c/ \propto$$

$$\propto exp\left(-\frac{1}{\frac{2\sigma^2}{3}} \cdot \left(\mu_i - \frac{\mu_{i-1}+y_i+\mu_{i+1}}{3} \right)^2 \right)$$

This is a normal dist:

$$N\left(\frac{\mu_{i-1}+y_i+\mu_{i+1}}{3}, \frac{\sigma^2}{3} \right)$$

We will obtain different formula if $i = 1$ and if $i = n$. We will then get:

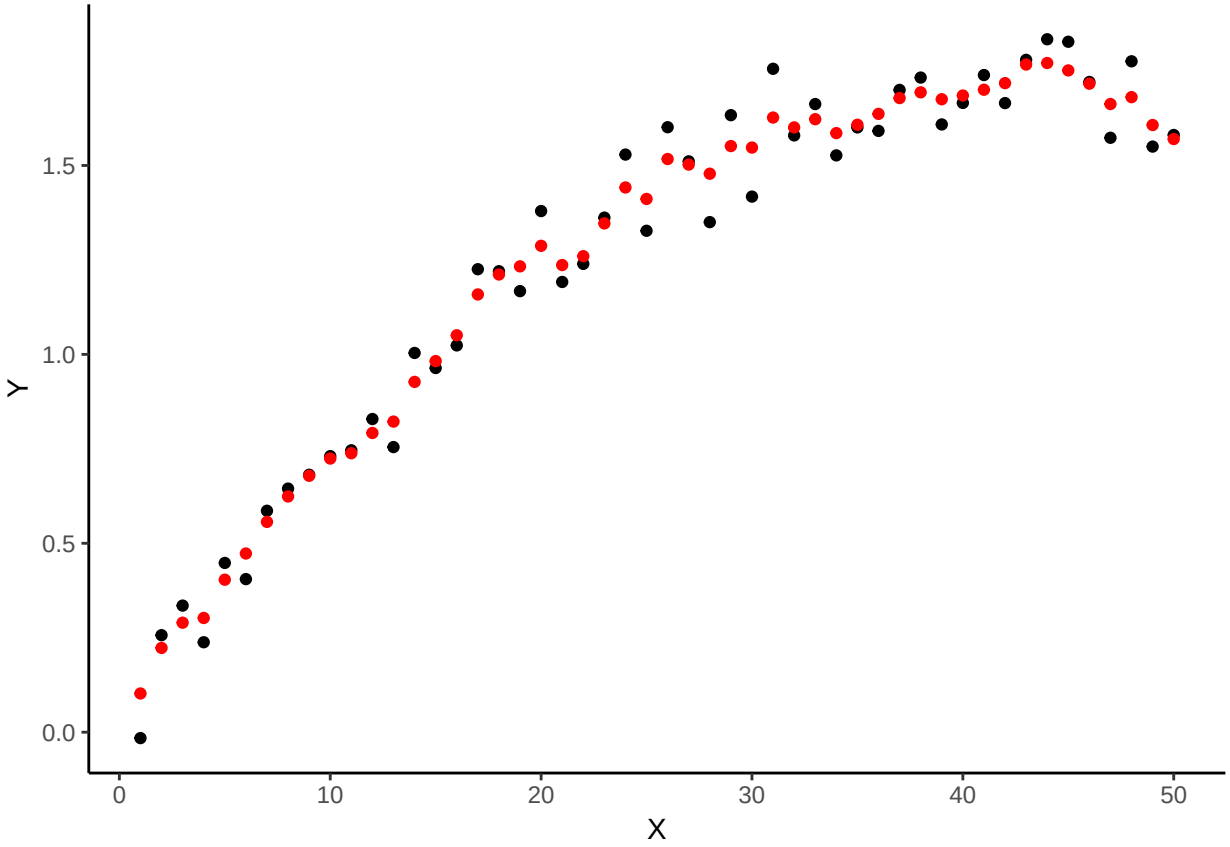
$$i = 1, \quad N\left(\frac{y_i + \mu_{i+1}}{2}, \frac{\sigma^2}{2}\right)$$
$$i = n, \quad N\left(\frac{\mu_{i-1} + y_i}{2}, \frac{\sigma^2}{2}\right)$$

4

In this task we will use the distributions derived in Step 3 to implement a Gibbs sampler where the initial $\mu_0 = (0, ..., 0)$. We will run this sampler 1000 times and then compute the expected values of μ.

```
mu <- rep(0, length(Y)) #Start value of mu
n <- length(mu)
sigma <- 0.2

matr <- matrix(0, nrow = 1000, ncol = n) #Save the mu:s in the loop



set.seed(12345)
for(j in 2:1000){
  mu_tmp <- c()
  for(i in 1:n){     # Sample new mu for each i. i goes from 1 to n

    if(i == 1){      #i=1 then we simulate from this dist
      mu_tmp[i] <- rnorm(1, mean = (Y[i] + mu[i+1])/2, sd = sqrt(sigma/2))
    }
    else if(i == n){ #i=n then we simulate from this dist
      mu_tmp[i] <- rnorm(1, mean = (mu[i-1] + Y[i])/2, sd = sqrt(sigma/2))
    }

    else{            #if i not 1 or n we sim from this dist
      mu_tmp[i] <- rnorm(1, mean = (mu[i-1] + Y[i]+ mu[i+1])/3, sd = sqrt(sigma/3))
    }


  }
  mu <- mu_tmp          #save mu for next iteration
  matr[j,] <- mu_tmp    #Save our mu
}


# Plot the estimated values of y
# extimated values of y=colMeans(matr)
ggplot() +
  geom_point(aes(x = X, y = Y)) +
  geom_point(aes(x = X, y = colMeans(matr)), colour = "red") +
```

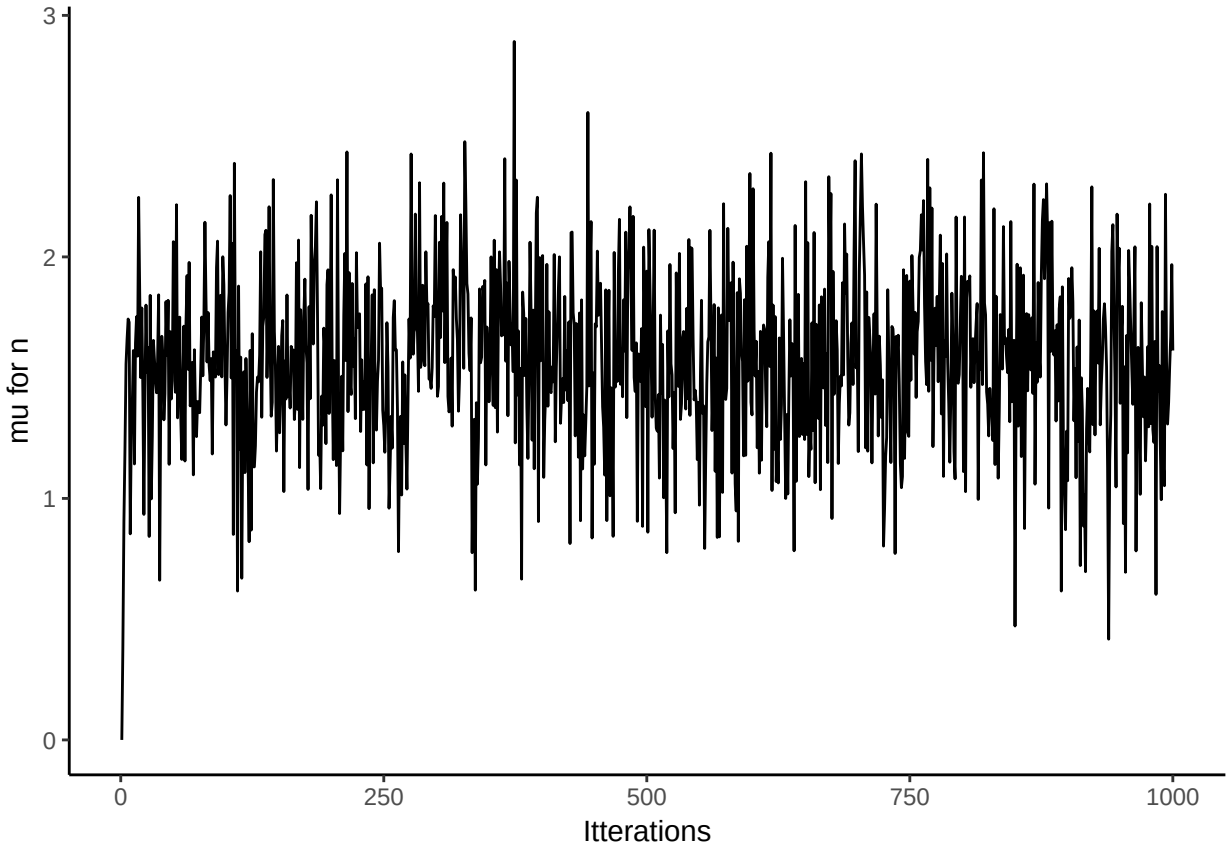
```
  labs(x = "X", y = "Y") +
  theme_classic()
```



We can see that the expected values have removed the noise in the data. We can also observe from the plot that the expected values catches the trend of the data between Y and X.

5

```
ggplot() +
  geom_line(aes(x = 1:1000, y = matr[,50])) +
  labs(x = "Itterations", y = "mu for n") +
  theme_classic()
```




```
Y[50]                                   #True value of y
```

```
## [1] 1.581
```

```
matr[1:10,50]                           #Estimated values
```

```
##  [1] 0.0000 0.3771 0.9043 1.2010 1.5567 1.6584 1.7427 1.7294 0.8533 1.2682
```

We can see from the plot that there is a burn-in period that is adjusting very fast to the mean (around 3 iterations). We can see that values hs a fast converge to a distribution.

Appendix

R-code

```
# Library ####
library(tidyverse)
library(gridExtra)
options(digits = 4)

x<-seq(0.1,20,0.1)
f_x <- function(x){
```

```
  x^5 * exp(-x)
}
y<-f_x(x)
plot(x,y,type="l",xlab="proc to f(x)",main = "Proc density of x")
MCMC_function<-function(nstep,X0){
  vN<-1:nstep          # values from 1 to nstep
  vX<-rep(X0,nstep)    # Our saved X values
  for (i in 2:nstep){
    X<-vX[i-1]


    Y<-Sim_Y_from_q_given_X(X)          #Sim Y~ q(.|X_t)
    u<-runif(1)                         #Sim u~ unif(0,1)

    # Calc alpha ##########
    nr1<-f_x(Y)
    nr2<-q_x_given_y(X,Y)

    nr3<-f_x(X)
    nr4<-q_x_given_y(Y,X)

    alpha<-1
    alpha[2] <-(nr1*nr2)/(nr3*nr4)
    alpha<-min(alpha)


    # Store or throw our Y value ####

    if (u <=alpha){
      vX[i]<-Y}
    else{
      vX[i]<-X}
  }
  #plot(vN,vX,pch=19,cex=0.3,col="black",xlab="t",ylab="X(t)",main="")

  return(vX)

}
q_x_given_y <- function(x,y){           # pdf q(x|y)
  dlnorm(x,meanlog = log(y), sdlog = 1)
}
f_x <- function(x){                     # pdf of target
  x^5 * exp(-x)
}
Sim_Y_from_q_given_X<-function(X){      # sim y~q(.|y)
  rlnorm(1,meanlog = log(X),1)
}

set.seed(123456789)
x_from_target<-MCMC_function(100,40)    # sim from target
x_from_target_ass_1_1<-x_from_target    # Save values for 1.5

plot(x_from_target,pch=19,cex=0.5,col="black",xlab="t",ylab="X(t)",main="Simulated values from our dist
```

```r
q_x_given_y <- function(x,y){           # pdf q(x|y)
  dlnorm(x,meanlog = log(y), sdlog = 0.2)
}
f_x <- function(x){                     # pdf of target
  x^5 * exp(-x)
}
Sim_Y_from_q_given_X<-function(X){      # sim y~q(.|y)
  rlnorm(1,meanlog = log(X),0.2)
}

set.seed(123456789)
x_from_target<-MCMC_function(100,40)    # sim from target

plot(x_from_target,pch=19,cex=0.5,col="black",xlab="t",ylab="X(t)",main="Simulated values from our dist

q_x_given_y <- function(x,y){         # pdf q(x|y)

  dchisq(x,df = floor(y+1))
}
f_x <- function(x){                   # pdf of target
  x^5 * exp(-x)
}
Sim_Y_from_q_given_X<-function(x){    # sim y~q(.|y)
  rchisq(1,df = floor(x+1))
}
set.seed(123456789)

x_from_target<-MCMC_function(100,40)  # sim from target
x_from_target_ass_1_2<-x_from_target  # Save values for 1.5


plot(x_from_target,pch=19,cex=0.5,col="black",xlab="t",ylab="X(t)",main="Simulated values from our dist
set.seed(123456789)

s_i<-c()
mean_i<-c()
n_l<-1000
k<-10
x_from_target<-list()
for(i in 1:k){
  x_from_target[[i]]<-MCMC_function(n_l,i)  # sim from target
  s_i[i]<-var(x_from_target[[i]])
  mean_i[i]<-mean(x_from_target[[i]])
}


W <- mean(s_i)

B <- n_l/(k-1) * sum((mean_i - mean(mean_i))^2)

var_v <- (n_l-1)/n_l * W + 1/n_l * B

R <- var_v/W
```

```r
sqrt(R) #Gelman-Rubin

mean(x_from_target_ass_1_1) #Estimate mean assigment 1.1
mean(x_from_target_ass_1_2) #Estimate mean assigment 1.2
load("chemical.RData")

ggplot(mapping = aes(x=X,y=Y))+
  geom_point()+
  theme_bw()+
  ggtitle("chemical data")+
  theme(plot.title = element_text(hjust = 0.5))+
  geom_smooth(method = "lm", formula = y ~ poly(x,2), se = FALSE,colour="gray20")
mu <- rep(0, length(Y)) #Start value of mu
n <- length(mu)
sigma <- 0.2

matr <- matrix(0, nrow = 1000, ncol = n) #Save the mu:s in the loop



set.seed(12345)
for(j in 2:1000){
  mu_tmp <- c()
  for(i in 1:n){     # Sample new mu for each i. i goes from 1 to n

    if(i == 1){      #i=1 then we simulate from this dist
      mu_tmp[i] <- rnorm(1, mean = (Y[i] + mu[i+1])/2, sd = sqrt(sigma/2))
    }
    else if(i == n){ #i=n then we simulate from this dist
      mu_tmp[i] <- rnorm(1, mean = (mu[i-1] + Y[i])/2, sd = sqrt(sigma/2))
    }

    else{            #if i not 1 or n we sim from this dist
      mu_tmp[i] <- rnorm(1, mean = (mu[i-1] + Y[i]+ mu[i+1])/3, sd = sqrt(sigma/3))
    }


  }
  mu <- mu_tmp         #save mu for next iteration
  matr[j,] <- mu_tmp   #Save our mu
}


# Plot the estimated values of y
# extimated values of y=colMeans(matr)
ggplot() +
  geom_point(aes(x = X, y = Y)) +
  geom_point(aes(x = X, y = colMeans(matr)), colour = "red") +
  labs(x = "X", y = "Y") +
  theme_classic()
ggplot() +
  geom_line(aes(x = 1:1000, y = matr[,50])) +
  labs(x = "Itterations", y = "mu for n") +
```

```
  theme_classic()


Y[50]                                   #True value of y

matr[1:10,50]                           #Estimated values
##
```