mandetory4051

Contents

Part1

1.a Deriving the maximum likelihood estimator for beta.

$$l(b|y) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{-(y_i-\beta_i)^2}{2\sigma^2}}$$

log-likelihood:

$$log(l(b|y)) = log(\prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma}} e^{\frac{-(y_i-\beta_i)^2}{2\sigma^2}})$$

using the$log(a*b) = log(a) + log(b)$

$$= log((\frac{1}{\sqrt{2\pi\sigma}})^n e^{\sum_{i=1}^{n} \frac{-(y_i-\beta_i)^2}{2\sigma^2}})$$

Differentiate with respect to β_i and find max by setting it equal to zero

$$\frac{2\beta_i}{2\sigma^2} - \frac{2y_i}{2\sigma^2} = 0$$

$$\hat{\beta}_i = y_i$$

1.b

solving:

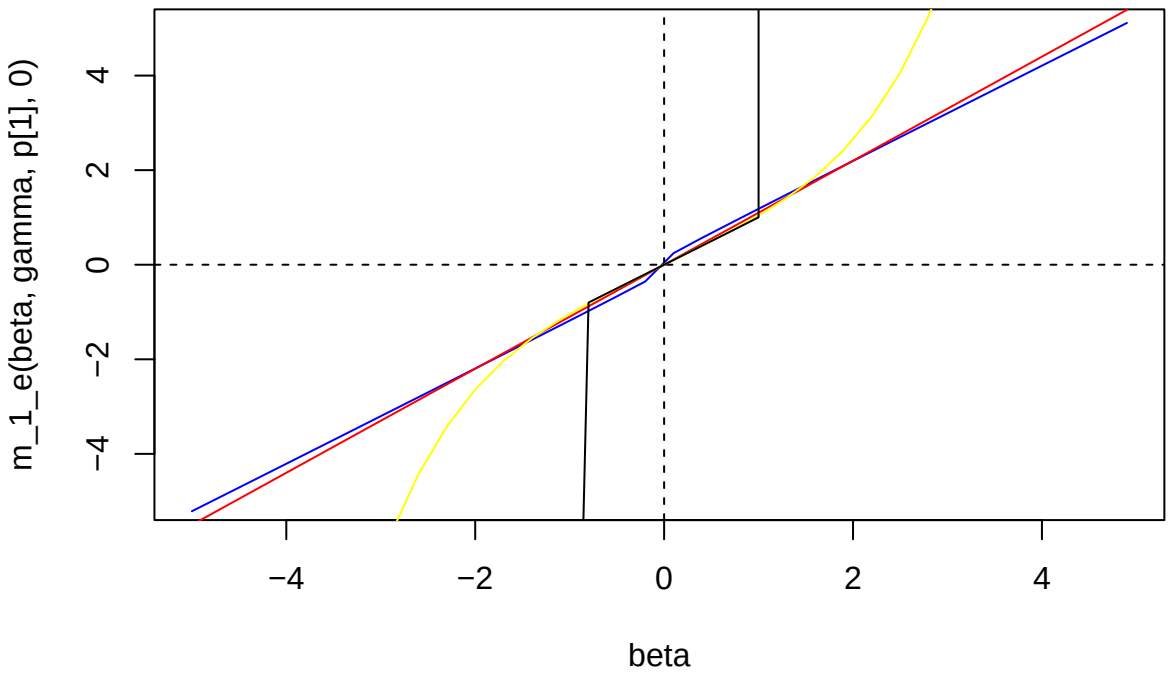
$$max_\beta(log(l(b|y))) + \frac{d}{d\beta_i}\frac{\gamma}{p}\sum_{i=1}^{n}|\beta_i|^p = 0$$

$$y_i = \beta_i + \gamma sign(\beta_i)|\beta_i|^{p-1}$$

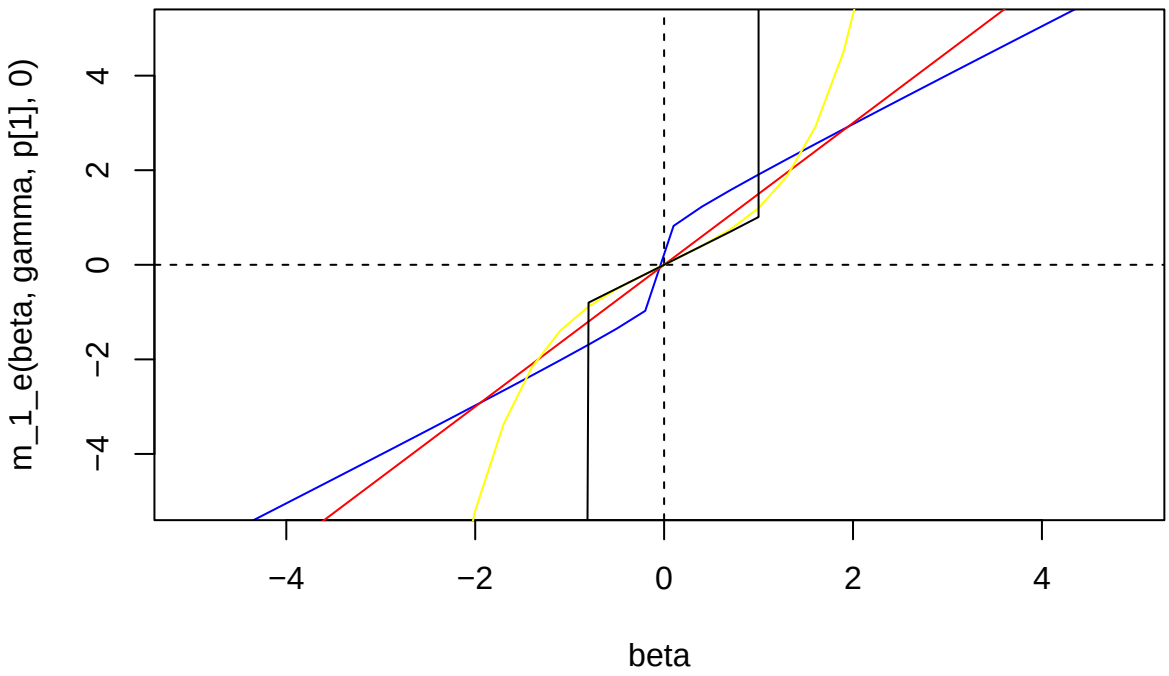
$$f_{p,\gamma}(\beta_i) = \beta_i + \gamma sign(\beta_i)|\beta_i|^{p-1}$$

1.c

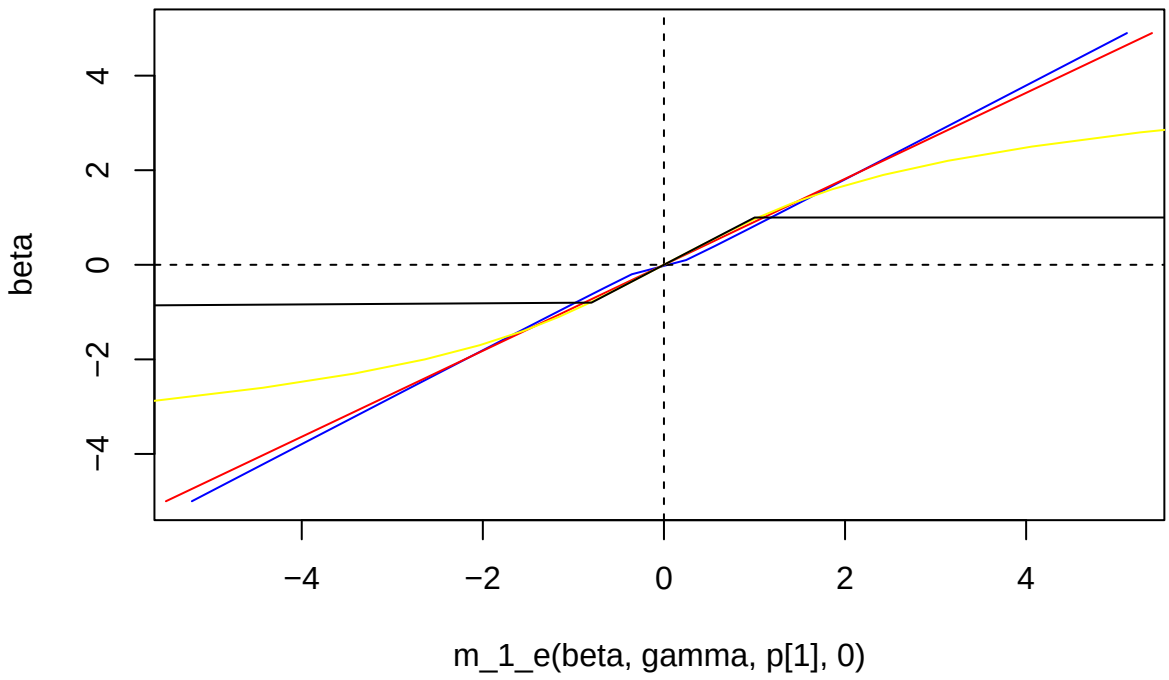
```
p=c(1.1,2,5,100)
beta=c(seq(from =-5,to=5,by=0.3))
gamma=0.2
m_1_e <- function(b,g,p,y){
  return(b+g/p*sign(b)*abs(b)^(p-1)-y)
}

color=c("blue","red","yellow","black")
plot(beta,m_1_e(beta,gamma,p[1],0),ylim=c(-5,5),type="l",col=color[1])
par(new=TRUE)

abline(h=0,lty=2)
abline(v=0,lty=2)
for (i in 2:4){


lines(beta,m_1_e(beta,gamma,p[i],0),type="l",col=color[i])

}
```



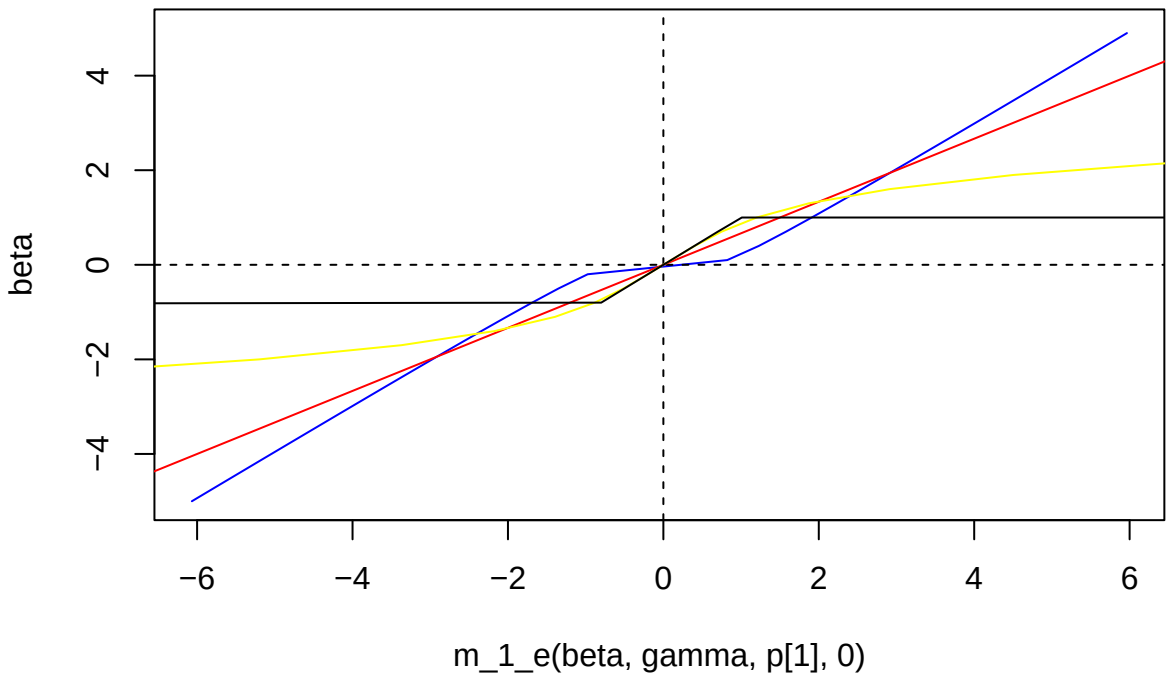
for gamma=0.2



for gamma=1



For gamma=0.2 inverse function

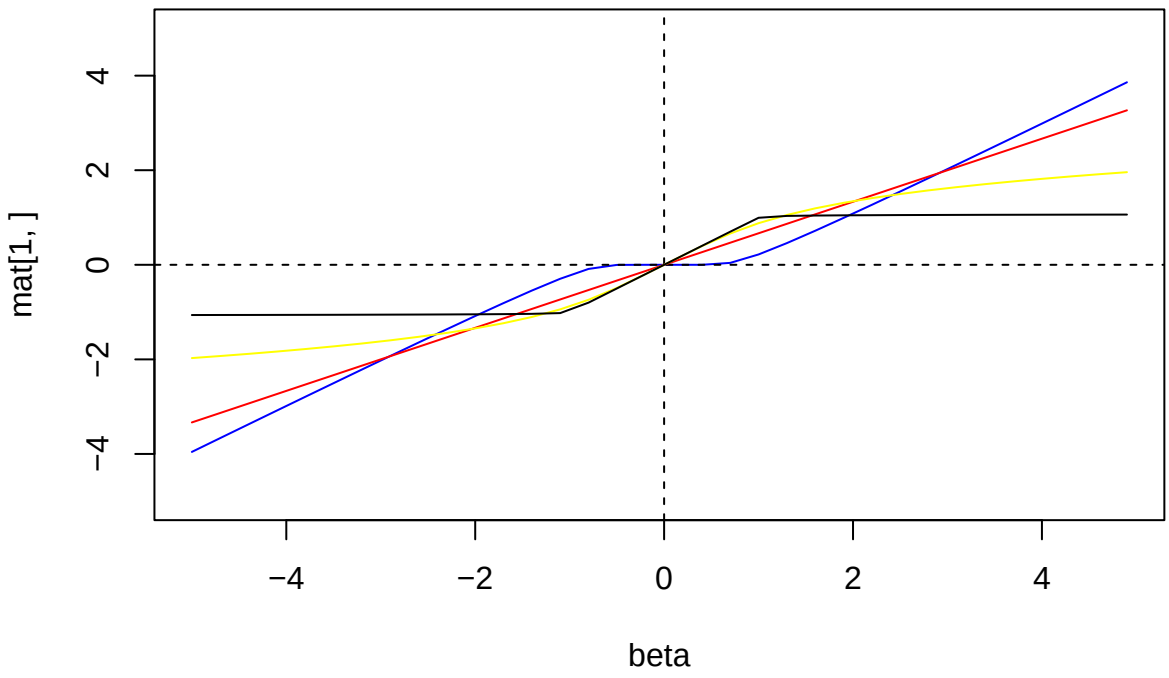


For gamma=1 inverse function

1.d

```
y_inte=seq(-5,5,0.3)
mat=matrix(rep(0,4*length(y_inte)),4,length(y_inte))
vec=c()

for (k in 1:length(p)){
for (j in y_inte){
a=-max(y_inte)
b=max(y_inte)
e=0.0001
gamma=1
ya=m_1_e(a,gamma[1],p[k],j)
yb=m_1_e(b,gamma[1],p[k],j)


c=b

i<-1

if(sign(ya)==sign(yb)){

  print("0")
}else {
  while(abs(m_1_e(c,gamma[1],p[k],j))>e){
```

```
    c<-(a+b)/2

    yc<-m_1_e(c,gamma[1],p[k],j)

    if(sign(yb)==sign(yc)){b<-c}else{a<-c}
    ya<-m_1_e(a,gamma[1],p[k],j)
    yb<-m_1_e(b,gamma[1],p[k],j)

    i<-i+1

  }

  #cat(c,"\n")
  vec=c(vec,c)
  mat[k,match(j,y_inte)]=c
  }

}
  }


#plot


color=c("blue","red","yellow","black")
plot(beta,mat[1,],ylim=c(-5,5),type="l",col=color[1])
par(new=TRUE)

abline(h=0,lty=2)
abline(v=0,lty=2)
for (i in 2:4){


  lines(beta,mat[i,],type="l",col=color[i])

}
```



1.e

```
load("sparseDataWithErrors.dat")
dat_b=betaGT
dat_y=y


low=min(y)
high=max(y)

gamma=1

m_1_e <- function(b,g,p,y){
  return(b+g/p*sign(b)*abs(b)^(p-1)-y)
}

bisection=function(low,high,p,gamma,y){
e=0.0001
ya=m_1_e(low,gamma,p,y)
yb=m_1_e(high,gamma,p,y)


c=high

i<-1
```

```
if(sign(ya)==sign(yb)){

  print("0")
}else {
  while(abs(m_1_e(c,gamma,p,y))>e){
    c<-(low+high)/2

    yc<-m_1_e(c,gamma,p,y)

    if(sign(yb)==sign(yc)){high<-c}else{low<-c}
    ya<-m_1_e(low,gamma,p,y)
    yb<-m_1_e(high,gamma,p,y)

    i<-i+1

  }



}

return(c)
}

p=c(1.1,2,100)
mat=matrix(rep(0,4*length(dat_y)),4,length(dat_y))


for (k in 1:length(p)){
  for (j in dat_y){
    mat[k,match(j,dat_y)]=bisection(low,high,p[k],gamma,j)

  }
}


res_GT=rep(0,3)
for(i in 1:3){
  for(j in 1:dim(mat)[2])
res_GT[i]=sum(betaGT[j]-mat[i,j])^2
}


res=y-mat[3,]
mat=matrix(rep(0,4*length(res)),4,length(res))
for (k in 1:length(p)){
  for (j in res){
    mat[k,match(j,res)]=bisection(low,high,p[k],gamma,j)

  }
}

res_res_GT=rep(0,3)
```

```
for(i in 1:3){
  for(j in 1:dim(mat)[2])
res_res_GT[i]=sum(betaGT[j]-mat[i,j])^2
}

for (i in 1:1000){
 MLE= sum((betaGT[i]-dat_y[i])^2)
}



print(res_GT)
```

```
## [1] 0.414583 1.018577 1.081079
```

```
print(res_res_GT)
```

```
## [1] 2.094795e-06 9.990076e-02 2.248380e-01
```

```
print(MLE)
```

```
## [1] 2.291918
```

The MLE seam to be worse then bought estimation methods.

1.f

We want to:

$$\min_b(-l(b|y,X))$$

we first use the penalty method. The nice thing with this is that by adding constrains to the original minimization problem we get a function that behaves smoother,has the same minimum,and we have equal number of unknowns as the original minimization problem.

We get:

$$\max_b(l(b|y,X)) + \frac{\gamma}{p}\sum_{i=1}^{n}|\beta_i|^p$$

know we use the Lagrangian method on this.Nice thing with this is that the derivative test of the unconstrained problem still can be applied. We express the gradient of the original function in terms of a linear combination of the gradient of the constrains(constrain).We dont have to reparametrize the original function to include the constrain we can instead, add the constrain with lambda term. The maximum of the original will be a saddle point in the lagragian problem. Our goal is to find stationary points in the lagragian problem.Ones we have done that the point that is a saddle is the max of the original problem.

We can therefore do.

$$\max_b(l(b|y,X)) + \frac{\gamma}{p}\sum_{i=1}^{n}|\beta_i|^p + \lambda * \sum_{i=1}^{n}|\beta_i|^1$$

we update $\frac{\gamma}{p}$ while we maximize the likelihood.$\lambda = \frac{\gamma}{p}$

2.a

$$l(\theta|y) = l(p, \tau^2|y) = \prod_{i=1}^{n} f(y_i) = \prod_{i=1}^{n} p\,\phi(y_i; 0, 1^2) + (1-p)\,\phi(y_i; 0, \tau^2 + 1^2)$$

2.b

introducing latent variable C_i

$$p(C_i = k) = \pi_k$$

$$log(f(y_i, C_i)) = \sum_{i=1}^{n} I(C_i = 0) log(\pi_0\, p\,\phi(y_i; 0, 1^2)) + I(C_i = 1) log(\pi_1\,(1-p)\,\phi(y_i; 0, \tau^2 + 1^2))$$

2.c

$$Q(\theta|\theta^{(t)} = E[log(f(y_i, C_i))] = \sum_{i=1}^{n} p(C_i = 0|y_i, \theta^{(t)}) log(\pi_0\, p\,\phi(y_i; 0, 1^2)) + p(C_i = 1|y_i, \theta^{(t)}) log(\pi_1\,(1-p)\,\phi(y_i; 0, \tau^2 + 1^2))$$

where:

$$\theta^{(t)} = [p^{(t)}, ((\tau)^2)^{(t)}]$$

to find the estimates for p and τ^2 we differentiate the Q function with respect to these parameters and set the derivative =0 and solve for the parameter.

$$\frac{d}{dp} Q(\theta|\theta^{(t)}) = \frac{\sum_{i=1}^{n} p(C_i = 0|y_i, \theta^{(t)})}{p} - \frac{\sum_{i=1}^{n} p(C_i = 1|y_i, \theta^{(t)})}{1-p} = 0$$

we get

$$p^{(t+1)} = \frac{\sum_{i=1}^{n} p(C_i = 0|y_i, \theta^{(t)})}{n}$$

since

$$n = \sum_{i=1}^{n} p(C_i = 0|y_i, \theta^{(t)} + \sum_{i=1}^{n} p(C_i = 1|y_i, \theta^{(t)}$$

then for τ^2

only term contributing to the derivative is:

$$\sum_{i=1}^{n} p(C_i = 1|y_i, \theta^{(t)}) * (log(\phi(y_i; 0, \tau^2 + 1^2)))$$

differentiating this is the the same as differentiation

$$\sum_{i=1}^{n} p(C_i = 1|y_i, \theta^{(t)})(\log \frac{1}{\sqrt{\tau^2+1}} - \frac{y_i^2}{2(\tau^2+1^2)})$$

differentiating with respect to τ and setting equal to 0 gives:

$$\sum_{i=1}^{n} p(C_i = 1|y_i, \theta^{(t)})(\frac{-\tau}{\tau^2+1} + \frac{y_i^2\tau}{(\tau^2+1)^2}) = 0$$

$$(\tau^2)^{(t+1)} = \frac{\sum_{i=1}^{n} p(C_i = 1|y_i, \theta^{(t)})y_i^2}{\sum_{i=1}^{n} p(C_i = 1|y_i, \theta^{(t)})} - 1$$

2.d

Using the code form lectures modifying it a little bit gives:

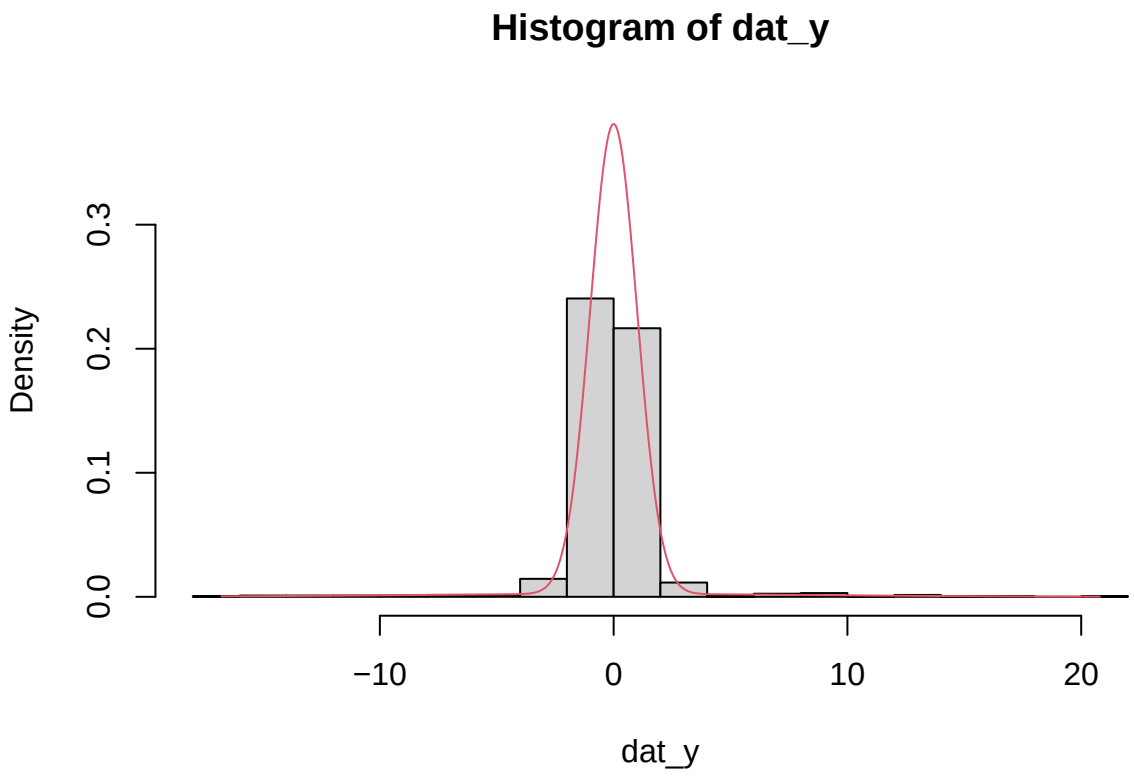
```
load("sparseDataWithErrors.dat")
dat_b=betaGT
dat_y=y

n = length(dat_y)
K=2  # number of classes
#Initial values
pi = c(0.5,0.5)
p=0.2

Tao_2 = rep(var(dat_y)+1)
sig = sqrt(Tao_2)
#Log-likelihood value
l = 0
for(i in 1:n)
  #modification to likelihood
  l = l+log(sum(pi*(p*dnorm(dat_y[i],0,1)+(1-p)*dnorm(dat_y[i],0,sig))))
l.old = l

more = TRUE
eps = 0.001
while(more)
{

  prob1 = matrix(NA,nrow=n,ncol=1)
  prob2 = matrix(NA,nrow=n,ncol=1)
  prob = matrix(NA,nrow=n,ncol=2)

  for(i in 1:n)
  {
    #modification to probability of C_i|y_i,theta^(t)
    prob1[i,]= pi[1]*(dnorm(dat_y[i],0,1))
    prob2[i,]=pi[2]*(dnorm(dat_y[i],0,sig))



    prob[i,]=cbind((prob1[i,])/(prob1[i,]+prob2[i,]),(prob2[i,])/(prob1[i,]+prob2[i,]))
```

```
  }


  pi = colMeans(prob)

  # calculting parameters for each step
  Tao_2 = (sum(prob[,2]*(dat_y)^2)/sum(prob[,2]))-1
  p=(sum(prob[,1])/(sum(prob[,1])+sum(prob[,2])))
  print(Tao_2)
  sig = sqrt(Tao_2+1)
  l = 0

  for(i in 1:n)
    l = l+log(sum(pi*(p*dnorm(dat_y[i],0,1)+(1-p)*dnorm(dat_y[i],0,sig))))

  show(round(c(pi,p,sig,l),2))
  more = abs(l-l.old)>eps
  l.old = l
}
```

```
## [1] 10.33019
## [1]     0.59     0.41     0.59     3.37 -1852.08
## [1] 14.84335
## [1]     0.72     0.28     0.72     3.98 -1774.15
## [1] 22.37122
## [1]     0.81     0.19     0.81     4.83 -1716.21
## [1] 33.65546
## [1]     0.88     0.12     0.88     5.89 -1682.59
## [1] 47.22961
## [1]     0.91     0.09     0.91     6.94 -1667.98
## [1] 59.27226
## [1]     0.93     0.07     0.93     7.76 -1663.26
## [1] 67.21086
## [1]     0.94     0.06     0.94     8.26 -1662.07
## [1] 71.44177
## [1]     0.94     0.06     0.94     8.51 -1661.82
## [1] 73.44084
## [1]     0.94     0.06     0.94     8.63 -1661.77
## [1] 74.33132
## [1]     0.95     0.05     0.95     8.68 -1661.76
## [1] 74.71753
## [1]     0.95     0.05     0.95     8.70 -1661.76
## [1] 74.88309
## [1]     0.95     0.05     0.95     8.71 -1661.76
```

```
p=0.95
x = seq(min(dat_y),max(dat_y),length=500)

f = rep(0,100)
for(k in 1:K)
  f = f+pi[k]*(p*dnorm(x,0,1)+(1-p)*dnorm(x,0,sig))
hist(dat_y,20,freq=FALSE,ylim=c(0,max(f)))
```

```
lines(x,f,col=2)
```



getting a lot of mass at the top,maybe because of small data set.

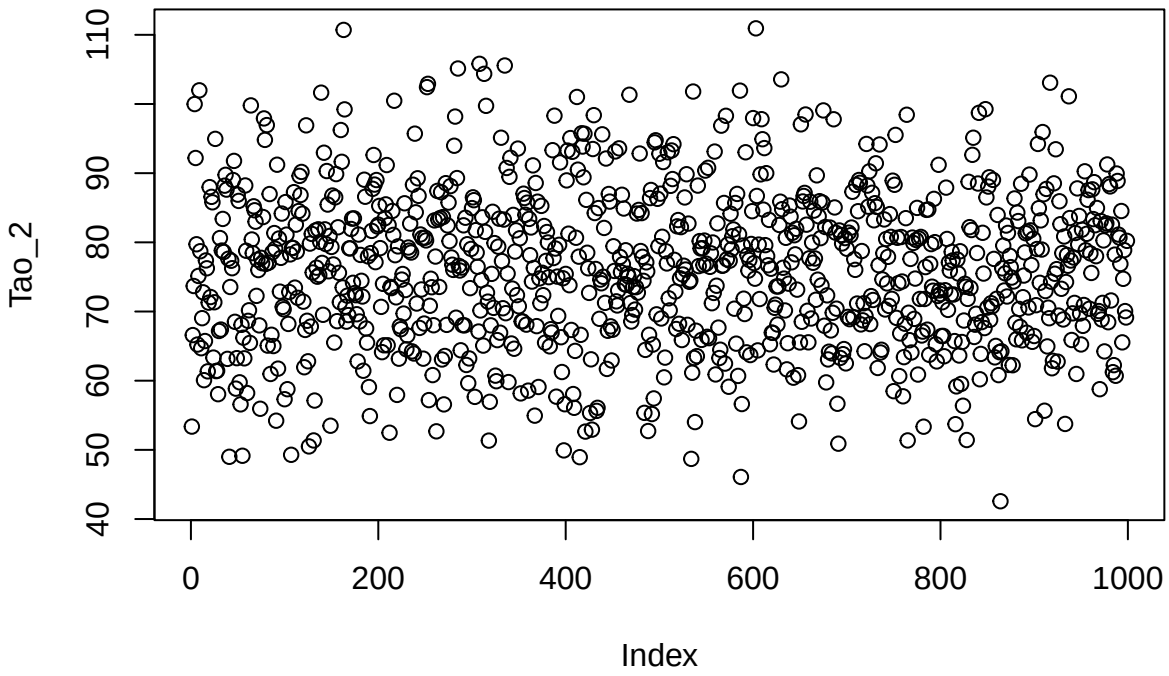
2.e

I did not quite get the comment i got in this exercise.I thought the point was to give a confidence on the true estimates using bootstrap. To me it was therefore natural to start with the estimated parameter calculated in 2.d and bootstrap a lot of times. So i sampled new observations y 1000 times for each observation in every iteration i calculate the probability of being in either class,that why i am using y[i],not y.

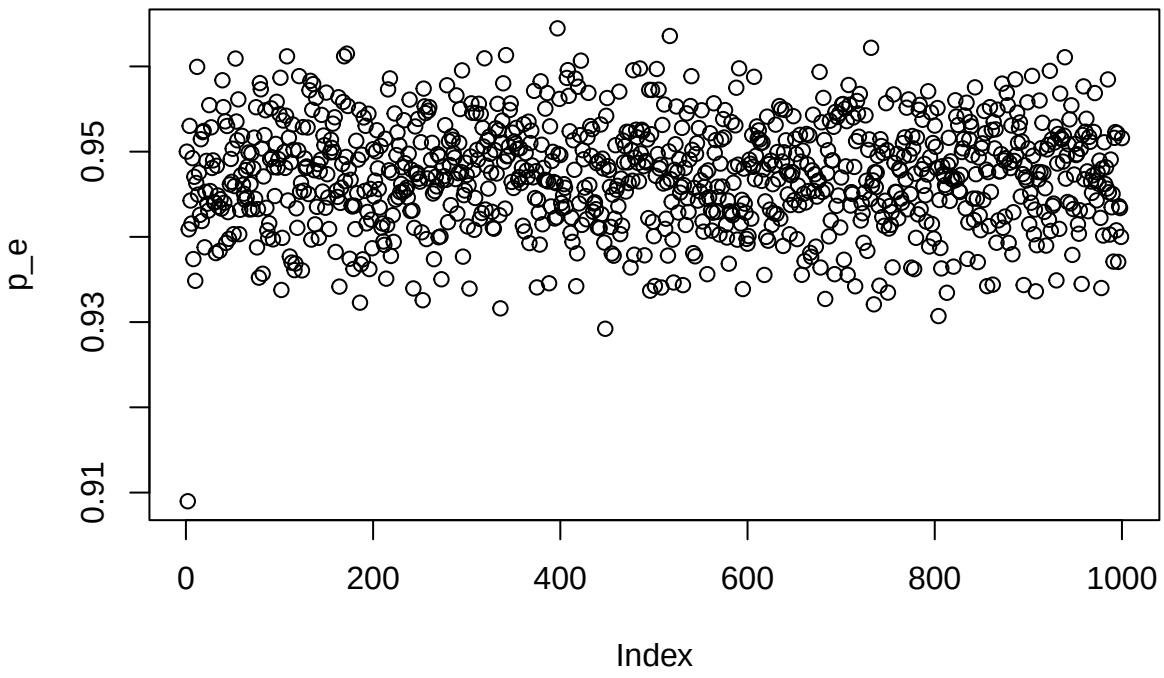
I did not have time to re-code and reflect over the comment you gave.

```
load("sparseDataWithErrors.dat")
dat_b=betaGT
dat_y=y

n = length(dat_y)
B=1000

#p=0.95


Tao_2=rep(0,B)

p_e=rep(0,B)
```

```
pi=c(0.95,0.05)

for (b in 1:B+1){

  y=sample(dat_y,size=length(dat_y),replace = TRUE)
  prob1 = matrix(NA,nrow=n,ncol=1)
  prob2 = matrix(NA,nrow=n,ncol=1)
  prob = matrix(NA,nrow=n,ncol=2)

  for(i in 1:n)
  {
    prob1[i,]= pi[1]*(dnorm(y[i],0,1))
    prob2[i,]=pi[2]*(dnorm(y[i],0,sqrt(Tao_2[b]+1)))



    prob[i,]=cbind((prob1[i,])/(prob1[i,]+prob2[i,]),(prob2[i,])/(prob1[i,]+prob2[i,]))
  }

  Tao_2[b+1] = (sum(prob[,2]*(y)^2)/sum(prob[,2]))-1
  p_e[b]=(sum(prob[,1])/n)




}

Tao_2=Tao_2[-1:-3]
p_e=setdiff(p_e,head(p_e,1))
mea_Tao=mean(Tao_2 )
sd_Tao=sd(Tao_2)

mea_pe=mean(p_e )
sd_pe=sd(p_e)


plot(Tao_2)
```




```
plot(p_e)
```




```
sd_Tao=sd(Tao_2)

sd_Tao
```

```
## [1] 11.18675
```

```
sd_pe=sd(p_e)

sd_pe
```

```
## [1] 0.006217
```

2.f

I desided to use a function from library numDeriv.I get Na values for $\frac{d^2}{d^2p}$ when p=0.95.Which also makes the cross terms Na. It looks like the term in the second derivative which has (1-p)^2 in the fraction is to close to 0 for this p value. I ended up adding 0.004 into the log term in the log likelihood of $f(y_i;\theta^{(t)})$

In order to calculate the index (1,1),(1,2),(2,1) in the matrix.Then take away 0.004 to calculate index (2,2) and then joining the results

```
library(numDeriv)
load("sparseDataWithErrors.dat")
likelihood.f <- function(theta){
```

```
  sum(log(theta[1]*dnorm(y,0,1)+(1-theta[1])*dnorm(y,0,sqrt(theta[2]+1))))

}

likelihood.f_2 <- function(theta){




  sum(log(theta[1]*dnorm(y,0,1)+(1-theta[1])*dnorm(y,0,sqrt(theta[2]+1))+0.004))

}


p=9.459121e-01
sig= 8.71109
u=c(p,(sig^2-1))


hess_1 <- hessian(func=likelihood.f, x=u)
hess_2=hessian(func=likelihood.f_2, x=u)
hess_2
```

```
##              [,1]        [,2]
## [1,] -1721.928405 1.608907716
## [2,]     1.608908 0.001129204
```

```
hess_2[2,2]=hess_1[2,2]

hess_1
```

```
##      [,1]         [,2]
## [1,]  NaN          NaN
## [2,]  NaN -0.004101216
```

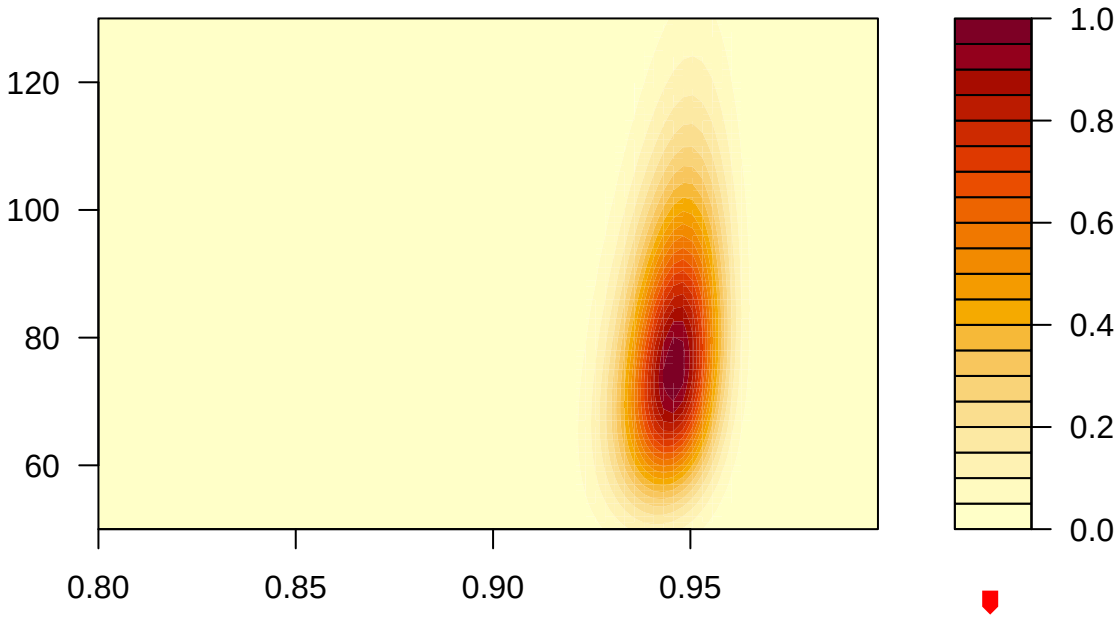
```
covar_matrix=-1/(hess_2)



covar_matrix
```

```
##               [,1]        [,2]
## [1,]  0.0005807442  -0.6215397
## [2,] -0.6215396880 243.8301408
```

The variance of sigma is much larger,could most likely be due to calculation error.

2.f

```r
load("sparseDataWithErrors.dat")

#log-likelihood

likelihood.f <- function(theta){

  f=sum(log(theta[1]*dnorm(y,0,1)+(1-theta[1])*dnorm(y,0,sqrt(theta[2]+1))))


  return(f)
}


#Making grd
vec2=sort(rep(seq(50,130,1),81))
vec1=rep(seq(0.8,1-1/(5*81),1/(5*81)),81)

vec22=rbind(vec1,vec2)

#calculating log-likelihood for each element in grid
store=c()
theta=0
for(i in 1:(length(vec22)/2)){
  theta=vec22[,i]


  store=c(store,likelihood.f(theta))
  theta=0
}

#tranforming back to likelihood
store=exp(store-max(store))

#creating likelihood matrix
store_mat=(matrix(store,nrow=81,ncol=81))

#setting
x = seq(0.8,1-1/(5*81),1/(5*81))
y = sort(seq(50,130,1))




#plotting countour plot
filled.contour(x,y,store_mat, nlevels = 20)
```



3.a,

Two classes two cases

C=0

$$E[y_i] = E[\beta_i] + E[\epsilon_i]$$

$$0 = E[\beta_i] + 0$$

$$E[\beta_i] = 0$$

for variance

$$Var[y_i] = Var[\beta_i] + Var[\epsilon_i]$$

$$1 = Var[\beta_i] + 1$$

$$Var[\beta_i] = 0$$

this means that: $p(\beta_i = 0/C_i = 0) = 1$ .C=0 corresponds to Beta_i=0.

Now for

C=1

$$E[y_i] = E[\beta_i] + E[\epsilon_i]$$

$$E[\beta_i] = 0$$

for the variance

$$Var[y_i] = Var[\beta_i] + Var[\epsilon_i]$$

$$\tau^2 + 1 = Var[\beta_i] + 1$$

$$Var[\beta_i] = \tau^2$$

the betas are normally distributed.Mean that:

$$p(\beta_i | C_i = 1) = \phi(\beta_i; 0, \tau^2)$$

$$p(\beta_i = 0 | y_i = y) = \frac{p(y_i = y | \beta_i = 0) p(\beta_i = 0)}{p(y_i = y)}$$

we can rewrite $p(\beta_i = 0)$

$$p(\beta_i = 0) = \sum_k p(\beta_i = 0 | C_i = k) p(C_i = k) = p(\beta_i = 0 | C_i = 0) p(C_i = 0)$$

for $k \in [0, 1] \in N$

$$= \frac{p(y_i | \beta_i = 0) p(\beta_i = 0 | C_i = 0) p(C_i = 0)}{p(y_i = y)}$$

$$= \frac{\phi(y_i; 0, 1) \ * 1 * p}{p \ \phi(y_i; 0, 1) + (1 - p) \, \phi(y_i; 0, \tau^2 + 1^2)}$$

3.b.

$$E[\beta_i | y_i = y] = \int \beta_i \, p(\beta_i, C_i = 1 | y_i = y) \, d\beta_i + \int \beta_i \, p(\beta_i, C_i = 0 | y_i = y) \, d\beta_i$$

$$= E[\beta_i, C_i = 1 | y_i = y] + E[\beta_i, C_i = 0 | y_i = y]$$

$$E[\beta_i, C_i = 0 | y_i = y] = 0$$

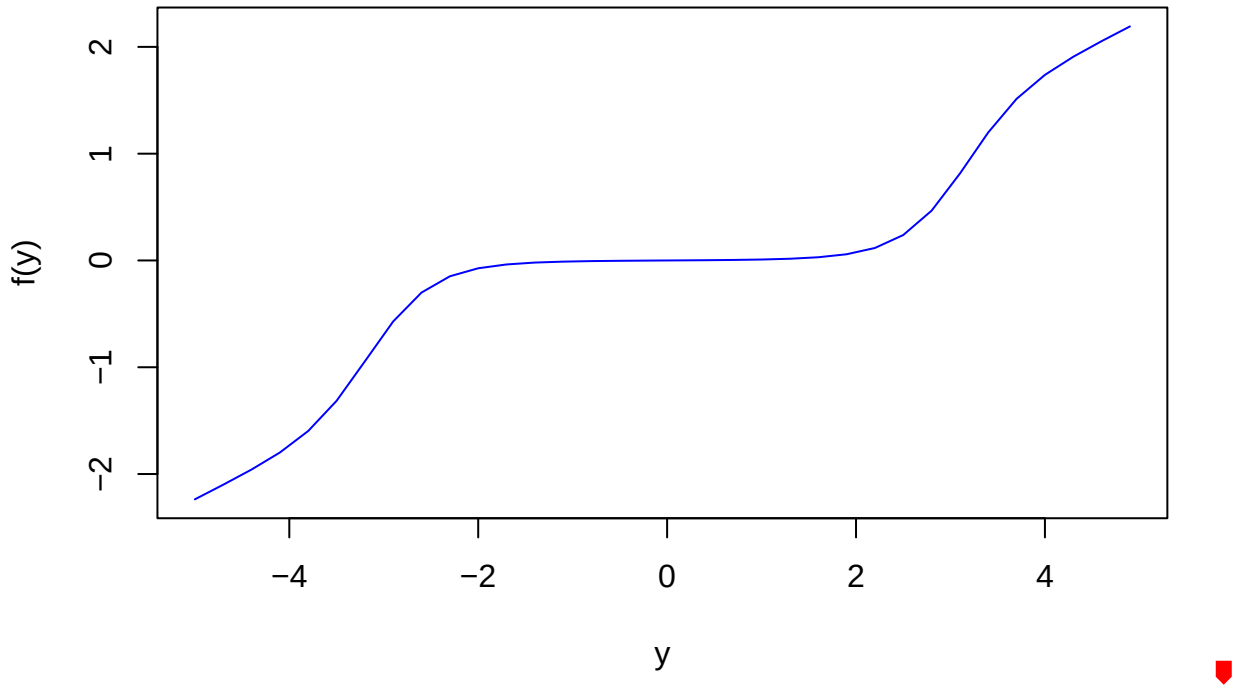
$$E[\beta_i | y_i = y] = E[\beta_i, C_i = 1 | y_i = y] = \ p(C_i = 1 | y_i = y) E[\beta_i | C_i = 1, y_i = y]$$

$$E[\beta_i|y_i = y] = p(C_i = 1|y_i = y)E[\beta_i|C_i = 1, y_i = y]$$

$$= \frac{p(y_i = y|C_i = 1) * p(C_i = 1)}{p(y_i = y)} * \frac{\tau^2}{\tau^2 + 1} y$$

$$= \frac{\phi(y_i; 0, \tau^2 + 1^2) * (1 - p)}{p\,\phi(y_i; 0, 1) + (1 - p)\,\phi(y_i; 0, \tau^2 + 1^2)} * \frac{\tau^2}{\tau^2 + 1} y$$

```
y=seq(-5,5,0.3)

f=function(y){

  return((0.1*dnorm(y,0,sqrt(80+1))/(0.9*dnorm(y,0,1)+0.1*dnorm(y,0,sqrt(80+1))))*(0.9^2/(0.9^2+1))*y)
}



plot(y,f(y),type="l",col="blue")
```



3.c

```
load("sparseDataWithErrors.dat")

beta=betaGT
```

```
for(i in length(beta)){
res=sum((beta[i]-f(y[i]))^2)
}
res
```

```
## [1] 0.0006241451
```

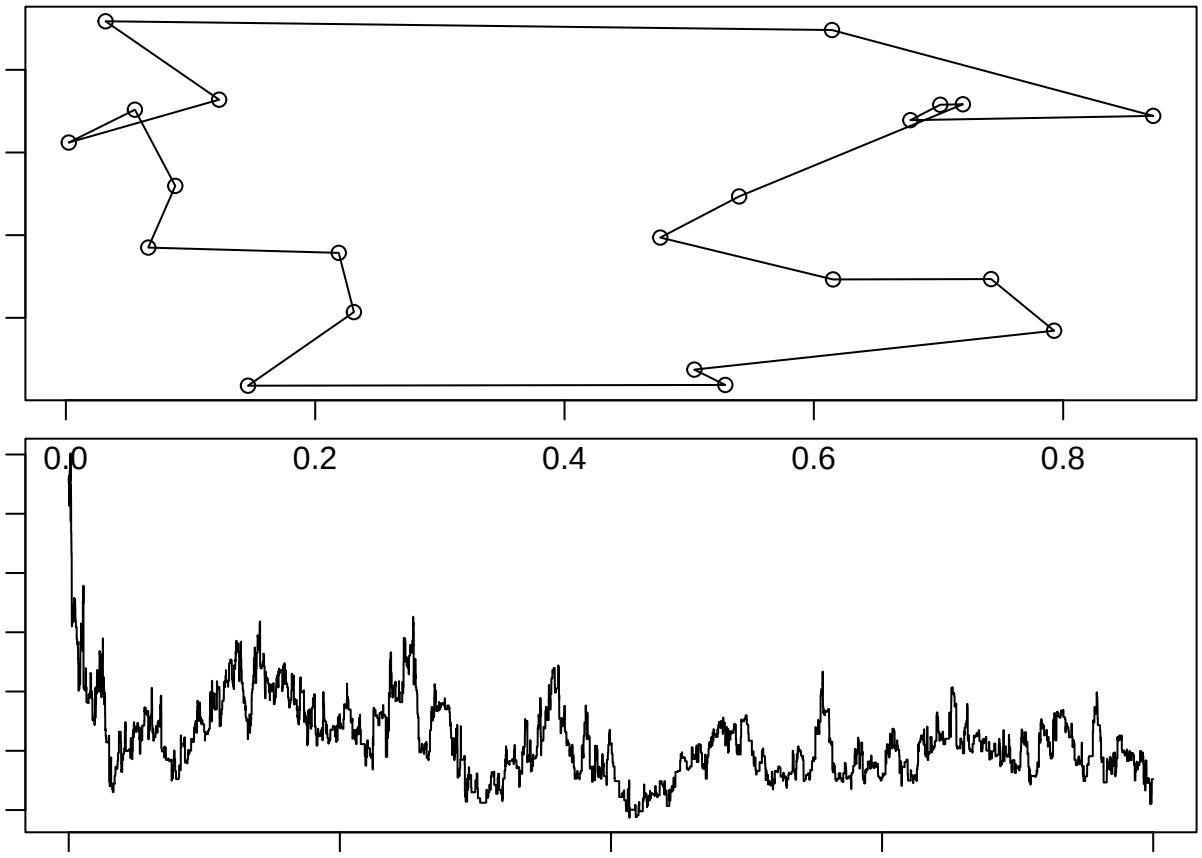
Comparing this to 1.e.It looks like the bayesian approach gives better results then the MLE and the bisection method,but if we estimate the betas through$\beta_{1,100}^{\hat{Alt}}$ this gives the best result,but in this way we are doing besection on the residuals between the MLE and the a bisection estimate.

4.a

Using code from lectures,modifying it a bit.

```
load("optimaltransport.dat")

set.seed(3453443)

pos=pos









d = as.matrix((dist(pos,diag=TRUE,upper=TRUE)))

d = as.vector(d)



#Modific of neigbour intial value,start and end with city 18
theta=c(18)
theta = c(theta,sample(setdiff(1:p,18),p-1))
theta=c(theta,18)



ind = (theta[-(p+1)]-1)*p+theta[-1]

V = sum(d[ind])

Vseq = V

Numit= 200000
V2=0
for(i in 1:Numit)
{
  #tau = 100/i
  tau = 1/log(i+1)
  #flip values exept for first and last element in initial list,beacause
```

```
  #we want to start and end in city 18
  ind2 = sample(2:p,2,replace=F)
  theta2 = theta
  theta2[ind2[1]] = theta[ind2[2]]
  theta2[ind2[2]] = theta[ind2[1]]
  ind2 = (theta2[-(p+1)]-1)*p+theta2[-1]

  #modification to the cost funtion start and end with 18
  V2=sum(d[ind2])+2*sum(d[1])
  prob = exp((V-V2)/tau)
  u = runif(1)
  if(u<prob)
  {
    theta = theta2
    V = V2
  }
  Vseq = c(Vseq,V)
}

par(mfrow=c(2,1),mar=c(0.5,1,0.5,1))
plot(pos)
lines(pos[theta,1],pos[theta,2])

plot.ts(Vseq[1:20000])
```



```
show(min(Vseq))
```

```
## [1] 3.770697
```

```
which.min(Vseq)
```

```
## [1] 168507
```

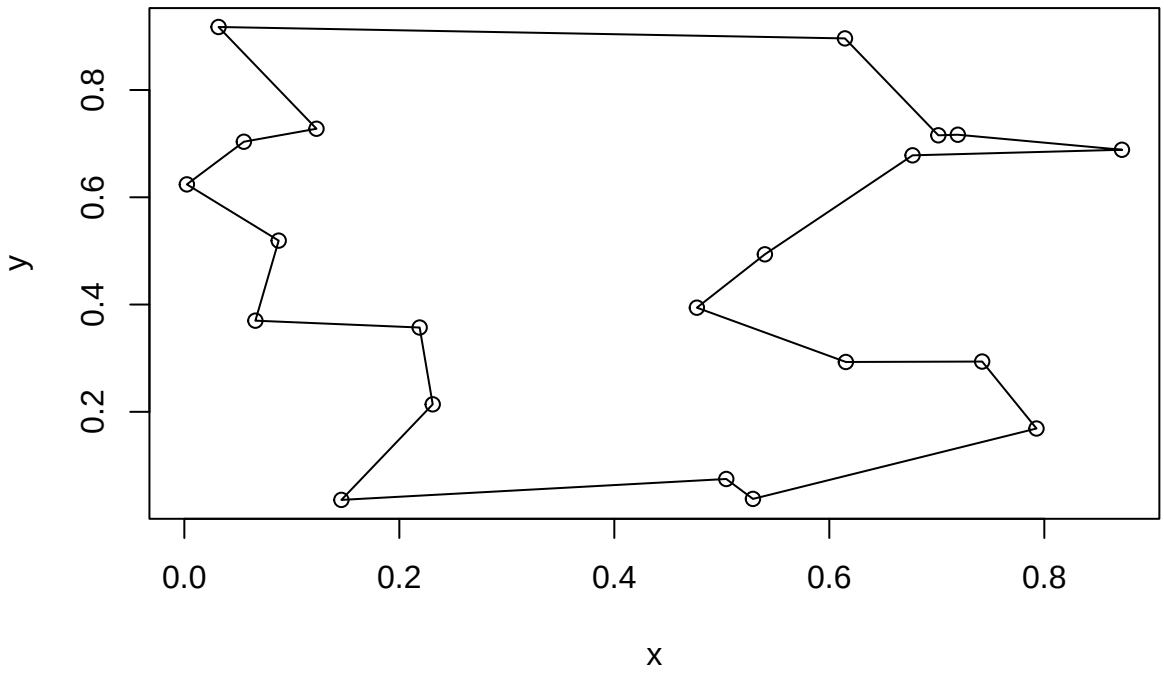
4.b

Using code from lectures,modifying it a bit.

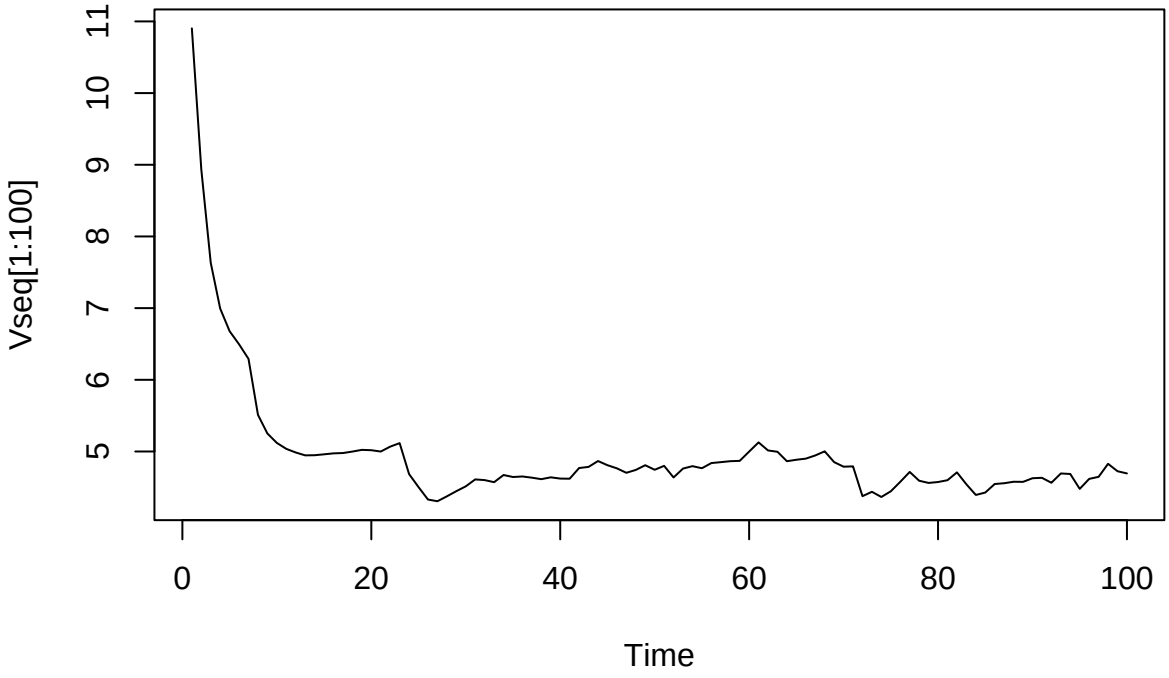
```
set.seed(2323)

load("optimaltransport.dat")

d = as.matrix((dist(pos,diag=TRUE,upper=TRUE)))

d = as.vector(d)


theta=c(18)
theta = c(theta,sample(setdiff(1:p,18),p-1))
theta=c(theta,18)

ind = (theta[-(p+1)]-1)*p+theta[-1]

V = sum(d[ind])
Vopt = V
Vseq = V

num = (p-1)*((p-1)-1)/2
searchtab = matrix(0,nrow=num,ncol=2)
ind = 1
for(i1 in 2:(p-1))
  for(i2 in (i1+1):p)
  {
    searchtab[ind,1:2] = c(i1,i2)
    ind = ind+1
  }




more = TRUE
tabu = NULL
H = NULL
tau = 20
store=c()

for(it in 1:10000)
```

```
{
  V2opt = V+1000
  i1opt = NA
  for(i in 1:num)
  {
    if(is.na(pmatch(i,H)))
    {

      i1 = searchtab[i,1]
      i2 = searchtab[i,2]

      theta2 = theta
      theta2[i1] = theta[i2]
      theta2[i2] = theta[i1]

      ind2 = (theta2[-(p+1)]-1)*p+theta2[-1]

      V2 = sum(d[ind2])+2*sum(d[1])

      if(V2<V2opt)
      {
        V2opt = V2
        iopt = i
        i1opt = i1
        i2opt = i2

        store=c(store,iopt)
      }
    }
  }

  theta2 = theta
  theta2[i1opt] = theta[i2opt]
  theta2[i2opt] = theta[i1opt]
  theta = theta2
  V = V2opt
  Vseq = c(Vseq,V)

  H = c(H,iopt)

  if(length(H)>tau)
    H = H[-1]


  if(V < Vopt)
  {
    theta.opt = theta
    Vopt = V
  }

}
#par(mfrow=c(2,1),mar=c(0.5,1,0.5,1))
par(mfrow=c(1,1))
```

```
plot(pos)
lines(pos[theta.opt,1],pos[theta.opt,2])
```




```
plot.ts(Vseq[1:100])
```




```
show(min(Vseq))
```

```
## [1] 3.790939
```

```
which.min(Vseq)
```

```
## [1] 5151
```

4.c

This problem is NP hard. It is not possible to know if any provided solution is optimal.

Simulating multiple times and taking choosing the path that shows up most of the times,in the end one can choose the result that took the least time.

4.d

We can include the home city node into our sample nodes,in addition to adding home city node at the beginning and at the end.Divide this sequence into two sequences,by having the sampled city node as a the indicator of where to separate.Then calculate the length of the two paths that resulted from separation.The neighborhood is defined in the same,as in the tasks above,but know we have two paths.Since the home node ends up in different places for every sample,if we sample enough times we will reach all possible paths.

5.a,

$$Q(\theta) = \sum_{i=1}^{n} (y_i - \hat{f}(x_i))^2$$

$$f(x_i) = \sum_{j=1}^{50} \beta_j \sigma(\alpha_j x_i + \alpha_{0,j}) + \beta_0$$

$$\frac{dQ}{d\beta_j} = -2 \sum_{i=1}^{n} (y_i - \hat{f}(x_i)) \frac{d\hat{f}}{d\beta_j}$$

$$\frac{dQ}{d\beta_j} = -2 \sum_{i=1}^{n} (y_i - \hat{f}(x_i)) \sigma(\alpha_j x_i + \alpha_{0,j})$$

$$\frac{dQ}{d\alpha_j} = -2 \sum_{i=1}^{n} (y_i - \hat{f}(x_i)) \sigma'(\alpha_j x_i + \alpha_{0,j}) x_i$$

$$\frac{dQ}{d\alpha_{0,j}} = -2 \sum_{i=1}^{n} (y_i - \hat{f}(x_i)) \sigma'(\alpha_j x_i + \alpha_{0,j})$$

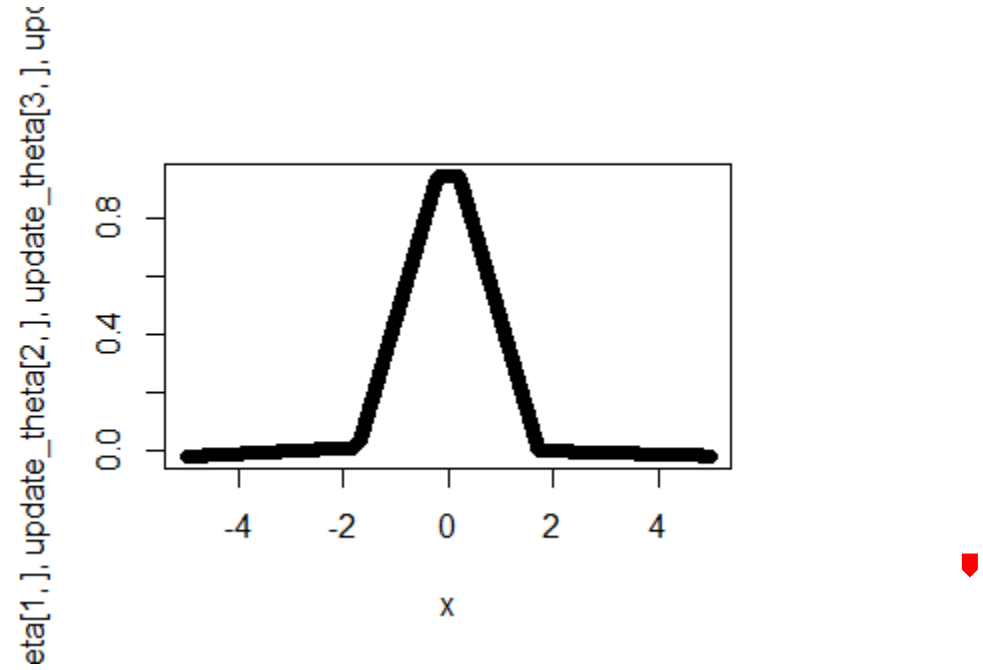
$$\frac{dQ}{d\beta_0} = -2 \sum_{i=1}^{n} (y_i - \hat{f}(x_i))$$

5.b

```
library(sigmoid)
#a=load("sparseDataWithErrors.dat")
load("functionEstimationNN.dat")
plot(x,fGT)
n_obs=length(x)



alpha=runif(50, min = -0.25, max = 0.25)
alpha_0=runif(50, min = -0.25, max = 0.25)
beta=runif(50, min = -0.25, max = 0.25)
beta_0=runif(1, min = -0.25, max = 0.25)




n_iter=40000


batch_size <- 50


func=function(beta,alpha,alpha_0,beta_0,x){
  f=0
  for(i in 1:50){
    #print(beta[i])
    f=f+(beta[i]*relu(alpha[i]*x+alpha_0[i]))
  }
  f_l=f+beta_0
```

```
  return(f_1)
}




#
#gradient using created function func
Gradient=function(beta,alpha,alpha_0,beta_0,x,y,fGT){

  dQdb=rep(0,length(beta))
  dQda=rep(0,length(alpha))
  dQda_0=rep(0,length(alpha_0))
  dQdb_0=rep(0,1)
  ker_in=0
  for(i in 1:50){

    dQdb[i]=-2*sum((y-func(beta,alpha,alpha_0,beta_0,x))*(relu(( alpha[i]*x+alpha_0[i]))))
    dQda[i]=-2*sum((y-func(beta,alpha,alpha_0,beta_0,x))*beta[i]*ifelse(alpha[i]*x+alpha_0[i]>0,1,0)*x)
    dQda_0[i]=-2*sum((y-func(beta,alpha,alpha_0,beta_0,x))*(beta[i]*ifelse(alpha[i]*x+alpha_0[i]>0,1,0))

  }
  dQdb_0=-2*sum((y-func(beta,alpha,alpha_0,beta_0,x)))

  return(rbind(dQdb,dQda,dQda_0,dQdb_0) )
}


#Iteration without taking into acount that prevous obs cannot be used in
#next sample
update_theta=rbind(beta,alpha,alpha_0,beta_0)
eps=1e-03
up_theta=matrix(rep(0,150),nrow = 3,ncol=50)
n=0
vec_idx=0
ind=c()
for(iter in seq_len(n_iter - 1)) {
  learning_rate=0.0019
  #learning_rate=0.0000019
  ind=c(ind,sample(seq_len(n_obs), size=batch_size))
  d=setdiff(seq_len(n_obs),ind)
  vec_idx <- sample(d, size=batch_size)
  if(length(d>=length(n_obs))){
    ind=c()

  }
  vec_idx <- sample(seq_len(n_obs), size=batch_size)
  update_theta=update_theta-learning_rate*(Gradient(
    update_theta[1,],update_theta[2,],update_theta[3,],update_theta[4,1],x[vec_idx],y[vec_idx],fGT[vec_
  #print(sum(update_theta-up_theta))
  b=func(update_theta[1,],update_theta[2,],update_theta[3,],update_theta[4,1],x[vec_idx])
  #print(b)
  c=abs(sum(y[vec_idx]-b))
```

```
  print(c)
  if(abs(c<eps)){
    break
  }

  up_theta=update_theta
  n=n+1
  print(n)

}



plot(x,func(update_theta[1,],update_theta[2,],update_theta[3,],update_theta[4,1],x))
b=func(update_theta[1,],update_theta[2,],update_theta[3,],update_theta[4,1],0)
```



5.c

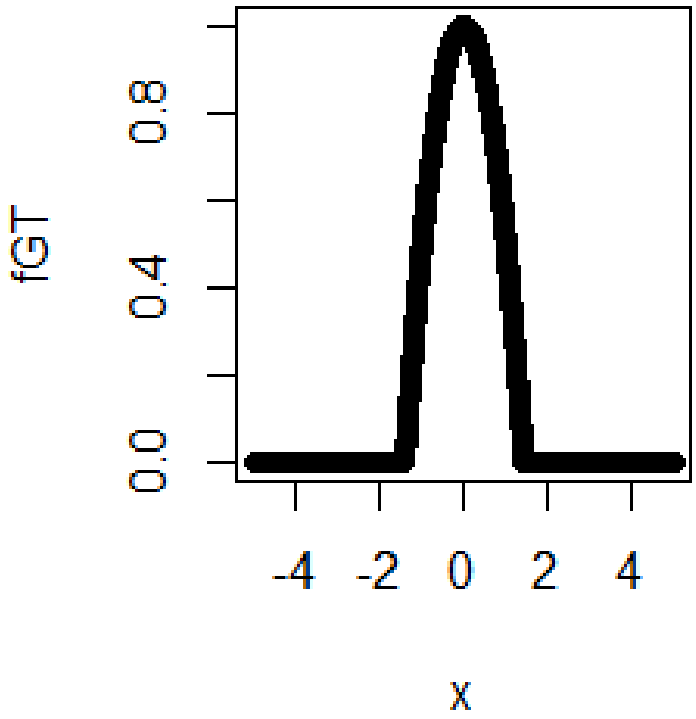
I tested for different values of the learning rate and for the stopping criteria , which is to stop when the

absolute value of the difference between the y values and the calculated function value is smaller then epsilon. If epsilon is 10^-3 it stops,if epsilon is smaller then that it never stops but as you increase the iteration size it gets closer and closer to the true curve.The picture over is from epsilon 10^-3. I did not want to sit and wait for 60000 iterations.

If the learning rate gets greater or equal to around 0.19 the algorithm diverges instantly to infinity.Same for the initial values if they are to large the same thing happens.Based on the function expression and the maximum value of the true function, i choose a small starting value for all hyper-parameters.The the alpha_j and alpha_0,j do not have to be small.For example , alpha_0,j could be close to alpha_j*x_i but with opposite sign.

Anyways i thought small initial values made sense.

plotting fGT



Part2

1.a

We sample from $U \sim \ Uniform[0,1]$. We define $X = F^{-1}(U)$ then we have that.

$$P(X < x) = P(F^{-1}(U) \leq x) = P(U \leq F(x)) = F(x)$$

Doing this in our case leads to:

$$F(X) = 1 - e^{\lambda x}$$
$$U = 1 - e^{-\lambda x}$$
$$X = \frac{-\log(1-U)}{\lambda}$$

1.b

standard normal

$$\frac{1}{\sqrt{2\pi}} e^{\frac{-x^2}{2}}$$

Taking the -log of the standard normal.

$$-\log \frac{1}{\sqrt{2\pi}} + \frac{x^2}{2}$$

tailor expansion of this

around -1

$$-\log \frac{1}{\sqrt{2\pi}} + \frac{1}{2} - (x+1)$$

around 1

$$-\log \frac{1}{\sqrt{2\pi}} + \frac{1}{2} + (x-1)$$

around 0

$$-\log \frac{1}{\sqrt{2\pi}}$$

Now transforming back the negative,easier to see.

around -1

$$\log \frac{1}{\sqrt{2\pi}} - \frac{1}{2} + (x+1)$$

around 1

$$\log \frac{1}{\sqrt{2\pi}} - \frac{1}{2} - (x-1)$$

around 0

$$\log \frac{1}{\sqrt{2\pi}}$$

taking the exponential :

$$\frac{1}{\sqrt{2\pi}} e^{x+\frac{1}{2}}$$

Same for taylor expansion around 1.

$$\frac{1}{\sqrt{2\pi}} e^{-x+\frac{1}{2}}$$

These two cases leads to an expression which is greater or equal to the standard normal:

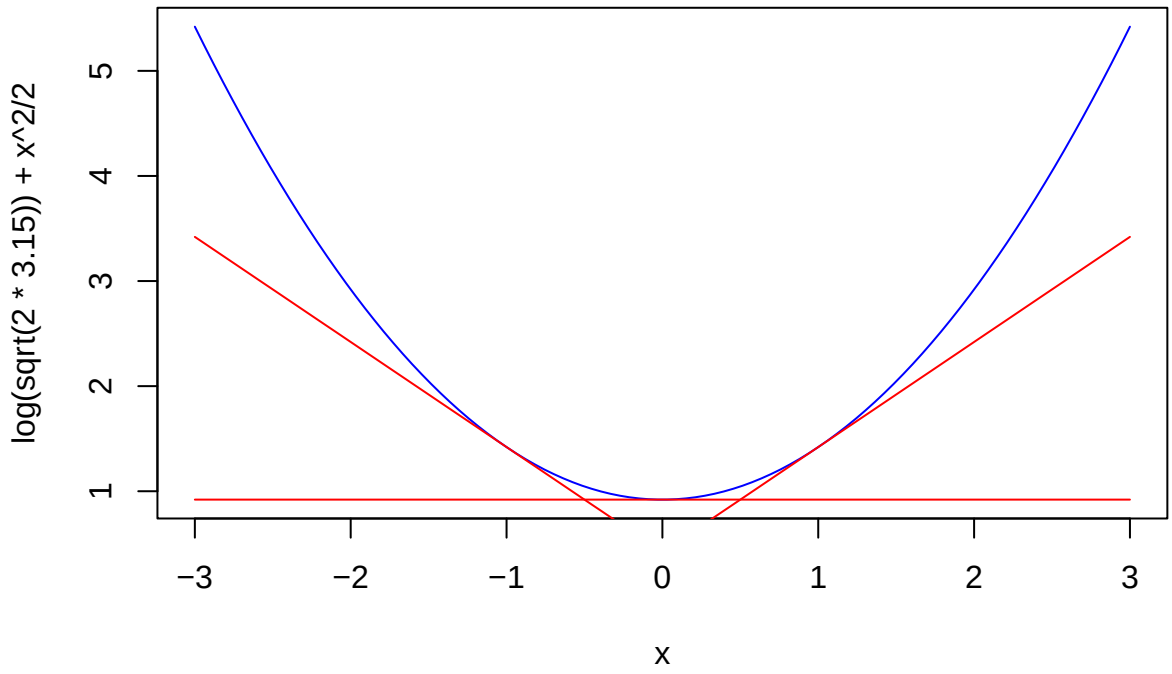
$$\frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}} \leq \frac{1}{\sqrt{2\pi}} e^{-|x|+\frac{1}{2}}$$

multiplying by - and taking the exponential for the taylor expansion around 0 gives the expression.

$$\frac{1}{\sqrt{2\pi}}$$

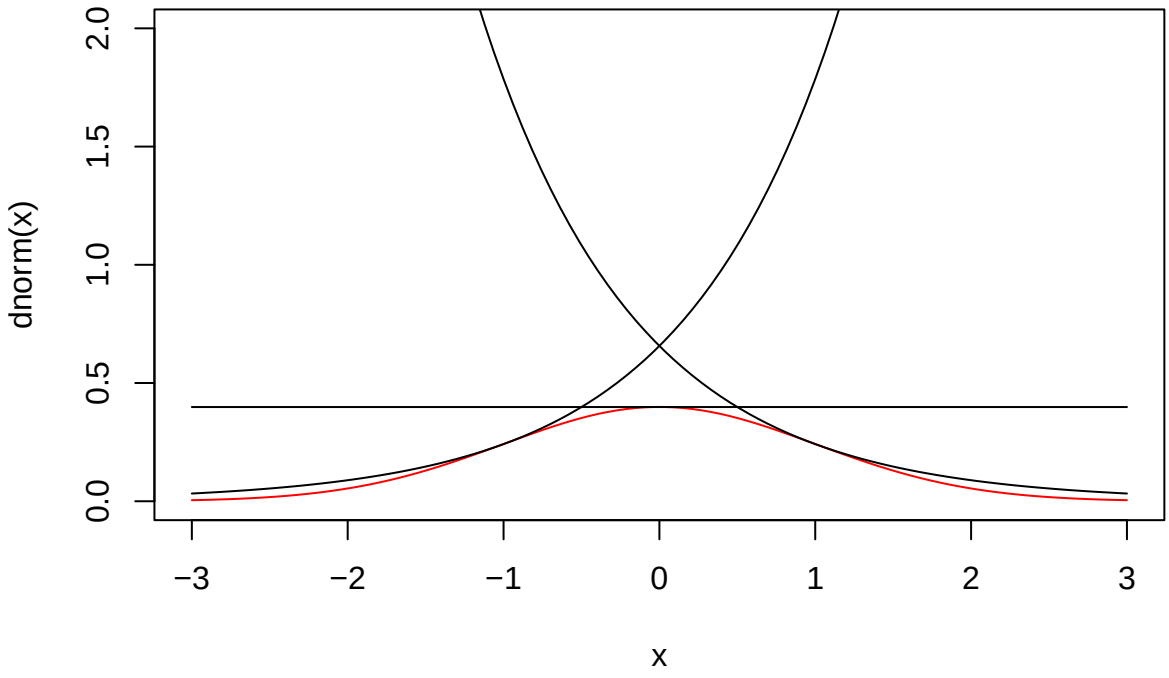
plotting the negative log density and the linearilizations:

```
x=seq(-3,3,length.out=100)
plot(x,log(sqrt(2*3.15))+x^2/2,type = "l",col="blue")
lines(x,log(sqrt(2*3.15))+0.5-(x+1),type = "l",col="red")
lines(x,log(sqrt(2*3.15))+0.5+(x-1),type = "l",col="red")
lines(x,rep(log(sqrt(2*3.15)),length(x)),type = "l",col="red")
```



plotting the bounding-function.

```
x=seq(-3,3,length.out=100)
plot(x,dnorm(x),type = "l",col="red",ylim = c(0,2))
lines(x,(1/sqrt(2*3.15))*exp(-x+0.5))
lines(x,(1/sqrt(2*3.15))*exp(x+0.5))
lines(x,rep(exp(-log(sqrt(2*3.15))),length(x)))
```



the bounding function is:

$$\begin{cases} \frac{1}{\sqrt{2\pi}} & -0.5 < x \leq 0.5 \\ \frac{1}{\sqrt{2\pi}} e^{-|x|+\frac{1}{2}} & else \end{cases}$$

This bounding function does not integrate to 1.If we normalize it by a normalizing constant by dividing by.$\frac{3}{\sqrt{2\pi}}$

we get:

$$\begin{cases} \frac{1}{3} & -0.5 < x \leq 0.5 \\ \frac{1}{3} e^{-|x|+\frac{1}{2}} & else \end{cases}$$

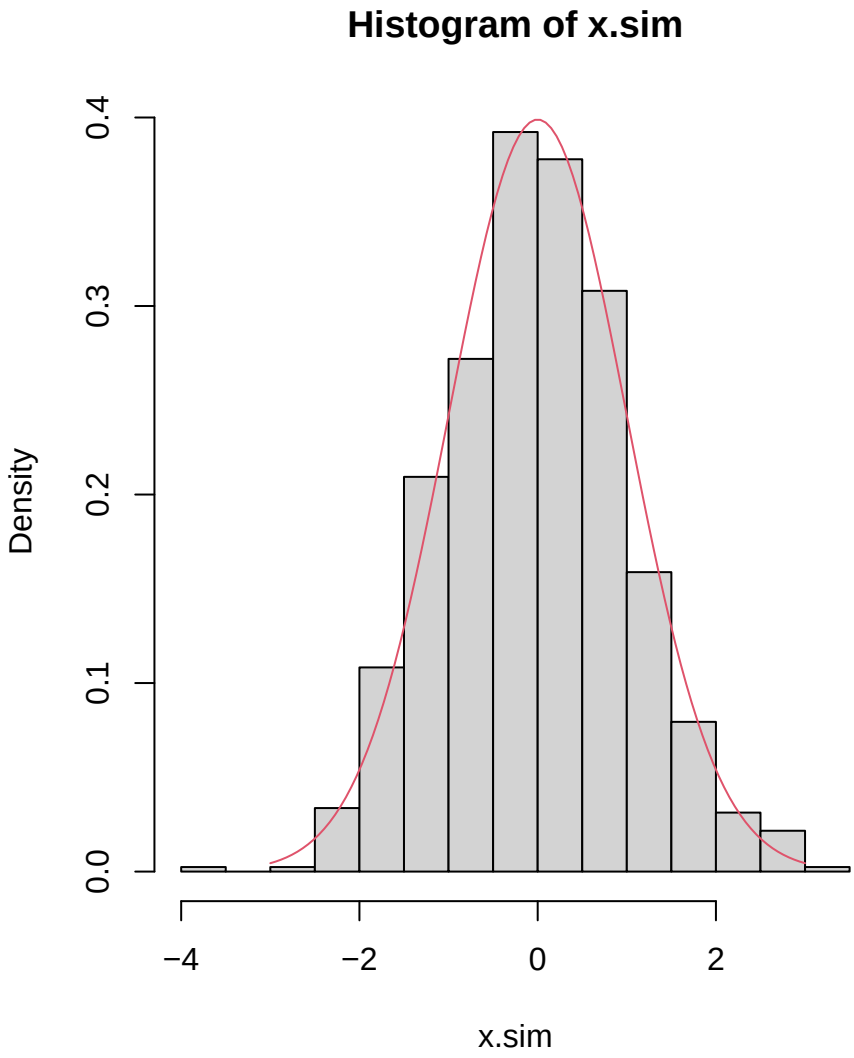
This function does:

$\int_{-\infty}^{-0.5} \frac{e^{-|x|+0.5}}{3} = \frac{1}{3}$ also for $\int_{0.5}^{\infty} \frac{e^{-|x|+0.5}}{3} = \frac{1}{3}$.

1.c

```
set.seed(1234534322)

g_funtion=function(y){
  if(y==0.5){
    f=1/3
  }
  if(abs(y)<0.5){
    f=1/3
```

```
  }else{
    f=exp(-abs(y)+0.5)/3
  }
  return(f)
}




n = 1000



num=sample(1:3,n,replace = TRUE)
u2 =runif(n)
func=function(num){
u1 = runif(1)

  if(num==1){
    x_p=u1-0.5
  }
  if(num==2){
    x_p=log(1-u1)-0.5
  }
  if(num==3){
    x_p=-log(1-u1)+0.5
  }
  return(x_p)
}



y=sapply(num,func)


acc = u2 < sqrt(2*3.15)*dnorm(y,0,1)/(sapply(y,g_funtion)*3)
x.sim = y[acc]
x = seq(-3,3,length=100)
hist(x.sim,freq=F)

lines(x,dnorm(x),col=2)
```

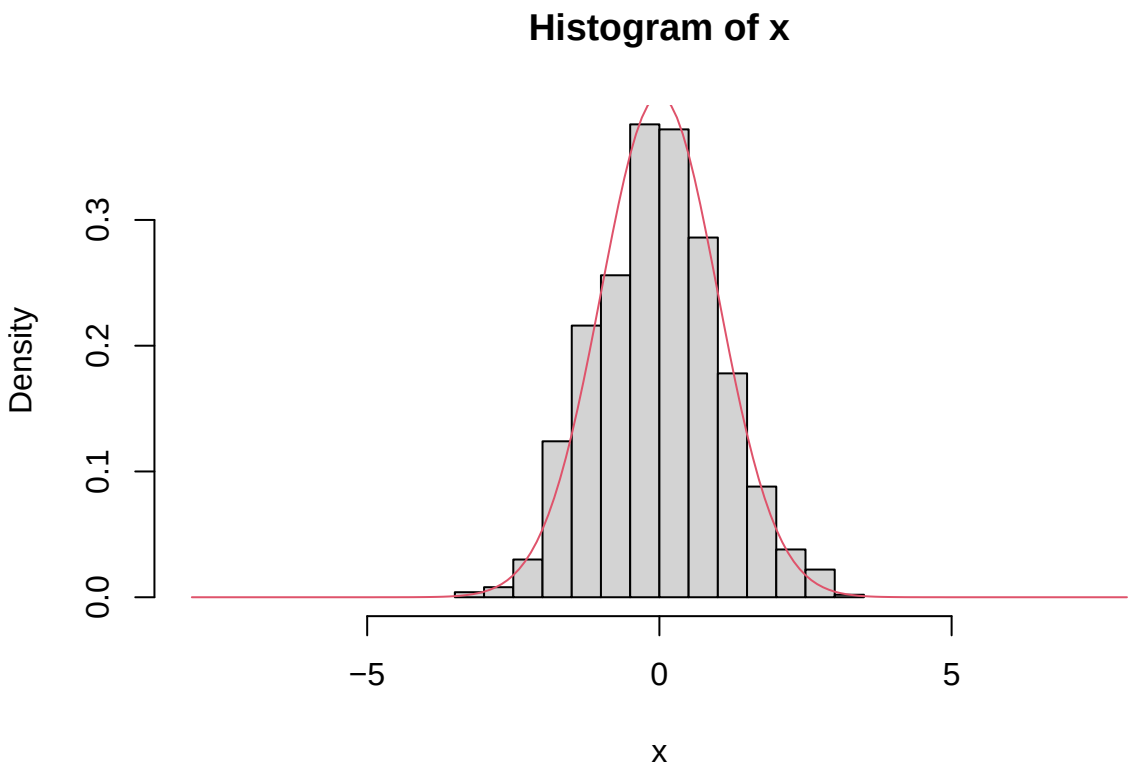



```
av_accept=mean(acc)
av_accept
```

```
## [1] 0.831
```

1.d

```
w = (dnorm(y,0,1)/sapply(y,g_funtion))
w=w/sum(w)
x = sample(y,n,replace=T,prob=w)
x = sort(x)
hist(x,20,freq=F,xlim=c(-8,8))
xp=seq(-8,8,by=0.1)
lines(xp,dnorm(xp),col=2)
```



Did normalize the weights . h(x) is equal to $1.\mu = 1$

$$MSE(\mu\hat{_{IS}}) - MSE(\mu^*_{IS})$$

gives:

$$cov(t(X), w^*(X)) = Var(w^*(X))$$

we get a gain if:

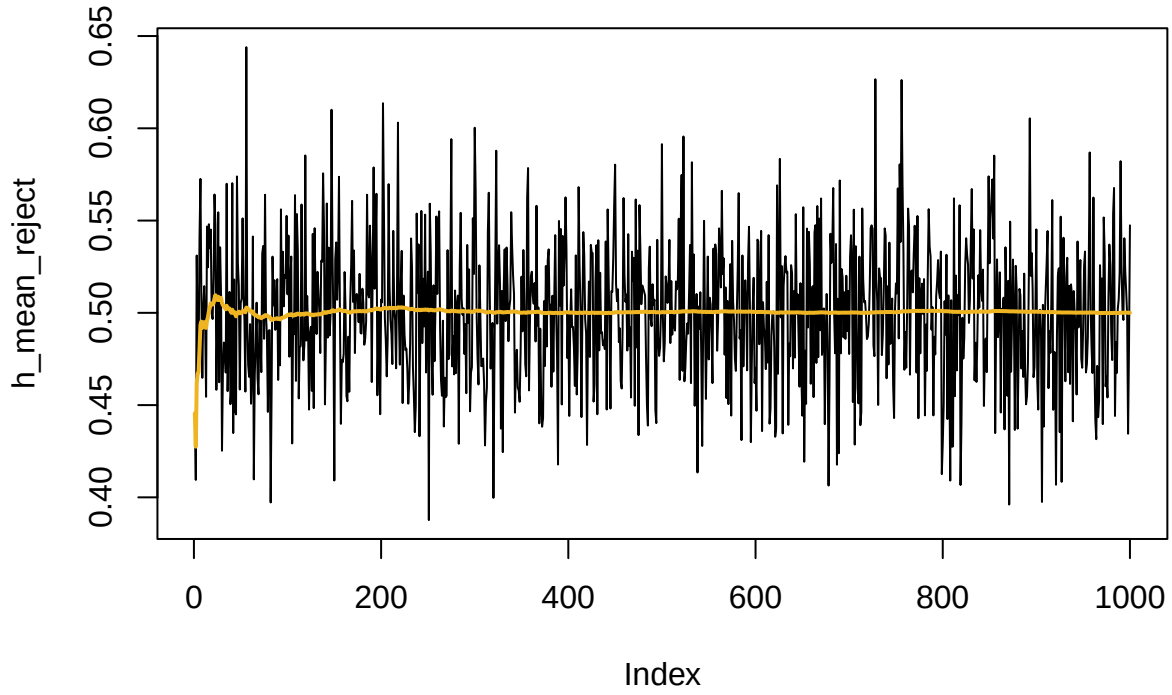
$$Var(w^*(X)) > \frac{Var(w^*(X))}{2}$$

1.e

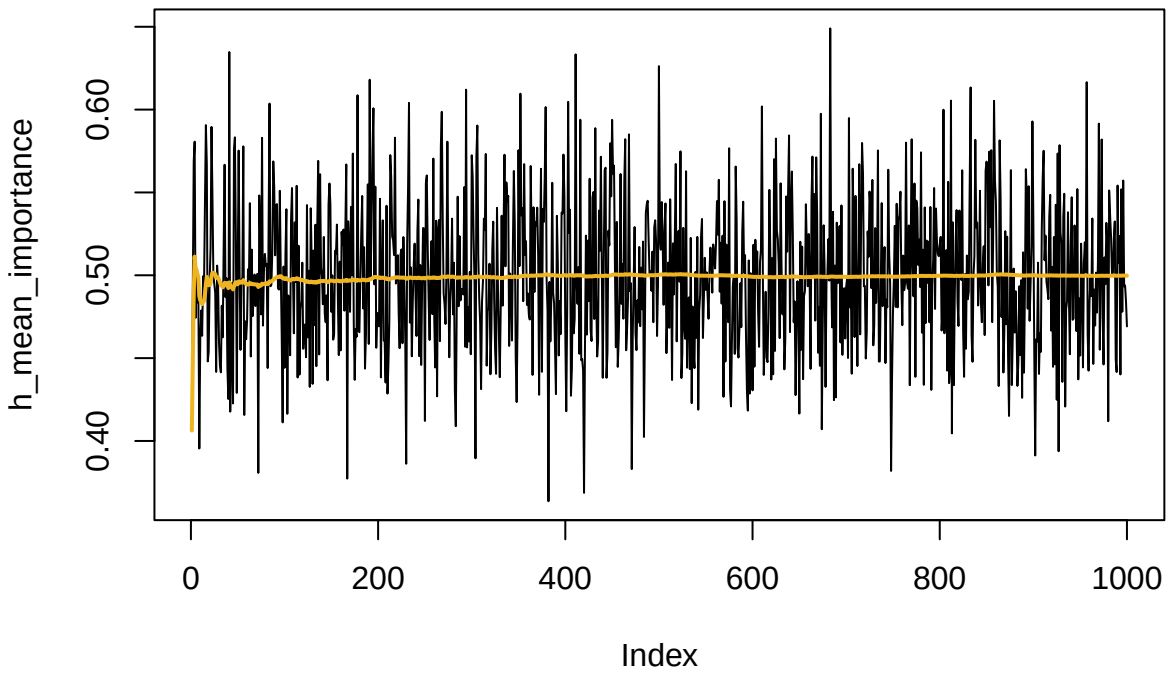
```
set.seed(1234534)

g_funtion=function(y){
  if(y==0.5){
    f=1/3
  }
  if(abs(y)<0.5){
    f=1/3

  }else{
```

```
    f=exp(-abs(y)+0.5)/3
  }
  return(f)
}

func=function(num){
  u1 = runif(1)

  if(num==1){
    x_p=u1-0.5
  }
  if(num==2){
    x_p=log(1-u1)-0.5
  }
  if(num==3){
    x_p=-log(1-u1)+0.5
  }
  return(x_p)
}


hx=function(y){

  return(y^2*ifelse(y>0,1,0))

}


n = 1000


h_mean_importance=rep(0,1000)
h_mean_reject=rep(0,1000)

for(i in 1:1000){

num=sample(1:3,n,replace = TRUE)
u2 =runif(n)

y=sapply(num,func)
#rejection sample
acc = u2 < sqrt(2*3.15)*dnorm(y,0,1)/(sapply(y,g_funtion)*3)
bind=sapply(y[acc],hx)
h_mean_reject[i]=mean(bind)

#importance sample
w = (dnorm(y,0,1)/sapply(y,g_funtion))

w=w/sum(w)

  x = sample(y,n,replace=T,prob=w)
  x = sort(x)
  bind=sapply(x,hx)
```

```
  h_mean_importance[i]=mean(bind)


}


plot(h_mean_reject,type = "l")

lines(cumsum(h_mean_reject)/(1:n),col="goldenrod2", lwd=2)
```




```
plot(h_mean_importance,type = "l")

lines(cumsum(h_mean_importance)/(1:n),col="goldenrod2", lwd=2)
```




```
print("Mean rejection sample")
```

```
## [1] "Mean rejection sample"
```

```
print(mean(h_mean_reject))
```

```
## [1] 0.5000772
```

```
print("Mean importance sample")
```

```
## [1] "Mean importance sample"
```

```
print(mean(h_mean_importance))
```

```
## [1] 0.4997683
```

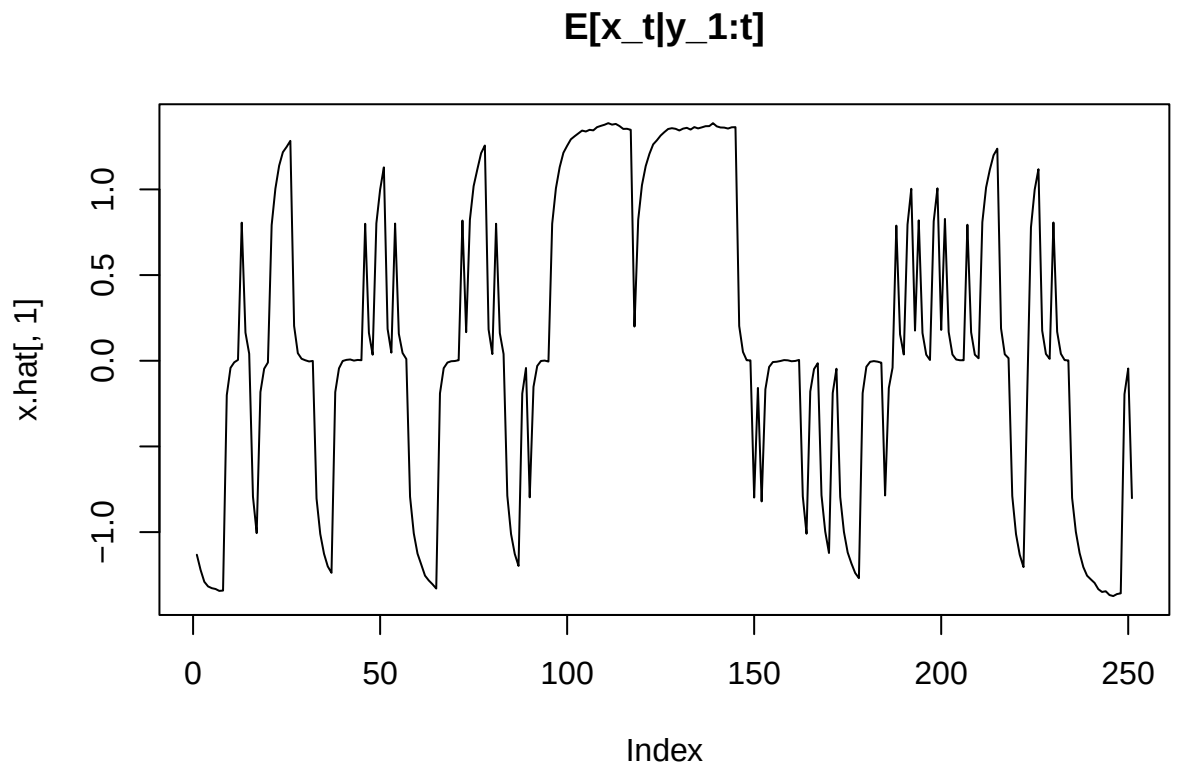
pros:Rejection sampling produces iid observations from the target distribution f .Cons: as the dimension increases the method is computationally heavy,since one potentially need a lot of uniform numbers to generate a specific value of the target random variable

pros: Good proposal function can lead to very high improvement.Cons:problem with large dimensions. Re-sampling creates bias because one is dividing by the sum of the weights if one is using the normalized weights.Re-sampling leads to inefficiency.

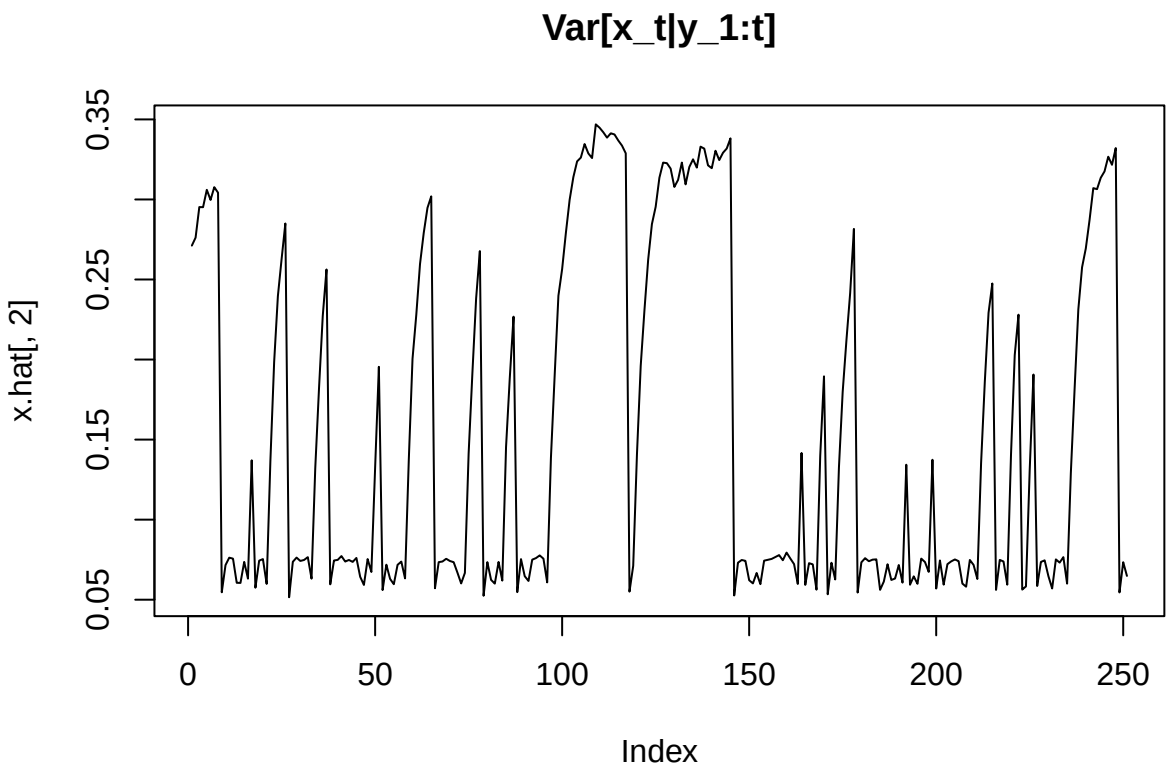
2.a

```
N=10000
sig=0.5
sig2=sig^2
a=0.85

#Initialization
load("TGsim.dat")
n_t=length(y)
x.sim = matrix(nrow=n_t,ncol=N)
x.sim[1,]=rnorm(N,0,1)




y_t_func=function(x){
  if(x< -0.5){
    y=1
  }
  if(x>=-0.5 && x<0.5){
    y=2
  }
  if(x>0.5){
    y=3
  }
  return(y)
}



y_x =sapply(x.sim[1,],y_t_func)
w=as.numeric(y[1]==y_x)


ind = sample(1:N,N,replace=T,prob=w)
x.sim[1,] = x.sim[1,ind]
w = rep(1/N,N)
x.hat=matrix(nrow = length(y),ncol = 2)
x.hat[1,1] = mean(x.sim[1,])
x.hat[1,2] =sd(x.sim[1,])^2


for(i in 2:n_t)
{

  x.sim[i,]=rnorm(N,a*x.sim[i-1,],sig)

  y_x =sapply(x.sim[i,],y_t_func)
  w=as.numeric(y[i]==y_x)

  ind = sample(1:N,N,replace=T,prob=w)
  x.sim[1:i,] = x.sim[1:i,ind]

  w = rep(1/N,N)
```

```
  x.hat[i,1] = mean(x.sim[i,])
  x.hat[i,2] = sd(x.sim[i,])^2



}


plot(x.hat[,1],type = "l",main="E[x_t|y_1:t]")
```




```
plot(x.hat[,2],type = "l",main="Var[x_t|y_1:t]")
```



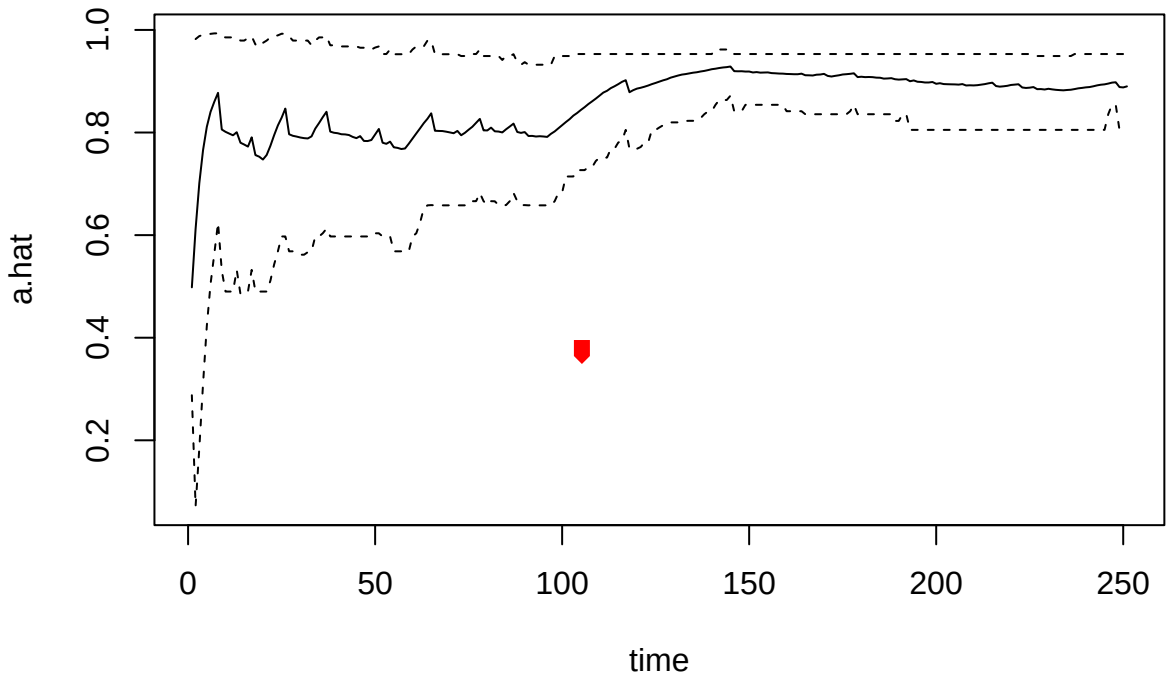
When some of the weights are close to 1 and a lot of the weights are close to zero,this lead to degeneracy.In this case we have a trivial degeneracy case,since the way we sample the x_t in each time set is use a categorical function if the some observation x is between -0,5 and 0.5 it get the value 2 and so on.Which means for every t,we set the weights of all observations corresponding to y_t=2 to 1 and the rest to zero.

The easiest way to fix this problem is re-sample and resetting the weights to 1/N . In the book their their is a condition where if the effective sample size is is below a threshold we do this step. But in our case we have a trivial case,we ended up doing it every time.

2.b

```
N=10000


#Initialization
load("TGsim.dat")
n_t=length(y)
x.sim = matrix(nrow=n_t,ncol=N)




y_t_func=function(x){
  if(x< -0.5){
    y=1
  }
```

```
  if(x>=-0.5 && x<0.5){
    y=2
  }
  if(x>0.5){
    y=3
  }
  return(y)
}



#Initialization
x.sim[1,]=rnorm(N,0,1)
y_x =sapply(x.sim[1,],y_t_func)
w = as.numeric(y[1]==y_x)
#Resample
ind = sample(1:N,N,replace=T,prob=w)
x.sim[1,] = x.sim[1,ind]
w = rep(1/N,N)
x.hat=matrix(nrow = length(y),ncol = 3)
x.hat[1,1] = mean(x.sim[1,])
x.hat[1,2:3] = quantile(x.sim[1,],c(0.025,0.975))
a.sim=matrix(nrow=n_t,ncol=N)
a.sim[1,] = runif(N)


a.hat = matrix(nrow=n_t,ncol=3)
a.hat[1,1] = mean(a.sim[1,])
a.hat[1,2] = sd(a.sim[1,])
for(i in 2:n_t)
{
  x.sim[i,]=rnorm(N,a.sim[i-1,]*x.sim[i-1,],sig)
  y_x =sapply(x.sim[i,],y_t_func)
  w = as.numeric(y[i]==y_x)

  #Resample
  ind = sample(1:N,N,replace=T,prob=w)
  x.sim[1:i,] = x.sim[1:i,ind]
  a.sim[i,] = a.sim[i-1,ind]
  w = rep(1/N,N)
  x.hat[i,1] = mean(x.sim[i,])
  x.hat[i,2:3] = quantile(x.sim[i,],c(0.025,0.975))
  a.hat[i,1] = mean(a.sim[i,])
  a.hat[i,2:3] = quantile(a.sim[i,],c(0.025,0.975))
}


matplot(cbind(1:n_t,1:n_t,1:n_t),a.hat,type="l",lty=c(1,2,2),col=1,xlab="time")
```



This approach is generally not recommended, because it leads to degeneracy.The plot looks like convergence,to the value of "a" used in 2.a.Looks acceptable.

3.a

We assume that each $Y_i \in \{0,1\}$ and iid $Y_i \sim Bernulli(p:i)$ where $p_i = \Phi(\beta^T X)$.

Where:

$$Bernulli(p) = \begin{cases} p_i & y_i = 1 \\ 1 - p_i & y_i = 0 \end{cases}$$

Likelihood:

$L(\beta|y) = \prod_{i=1}^{n} Bernulli(y_i, p) = \prod_{i=1}^{n} p_i^{y_i}(1-p_i)^{1-y_i}$

Taking the logarithm of the likelihood creates numerical stability since the it is not effected by one of the probabilities being zero.in addition if we rewrite the likelihood like this:

$$\prod_{i=1}^{n} p_i^{y_i}(1-p_i)^{1-y_i} = \prod_{y_i=1, p_i>0} p_i \prod_{y_i=0,(1-p)>0} (1-p)$$

we sort the probabilities into two categories which is more efficient.In addition we require the components to be strictly positive,which solves the problem of $|\beta^T X|$ being large.This problem will appear when taking the log of the likelihood:

$$l(\beta|y) = \log L(\beta|y) = log \prod_{y_i=1, p_i>0} p_i \prod_{y_i=0,(1-p)>0} (1-p)$$

$$= \sum_{y_i=1, p_i>0} \log(p_i) + \sum_{y_i=0, (1-p_i)>0} \log(1-p_i)$$

we have defined:

$$p_i = \Phi(\beta^T X)$$

if $\beta^T X$ is highly negative $p_i \to 0$,$\log p_i \to -\infty$ which is a problem.

One can in a numerically stable way through taking the exponential of the difference between the likelihood of the candidate parameter and the current parameter estimate in the markov chain:

$$R = e^{l(\beta^{cand}|y) - l(\beta^{x(t)}|y)}$$

3.b

we want to sample from the posterior of β given y:$p(\beta|y)$

this is our f so $f = p(\beta|y)$ but $p(\beta|y) \propto L(\beta|y) * p(\beta)$

$p(\beta) \propto 1$, so $p(\beta|y) \propto L(\beta|y) * p(\beta) \propto L(\beta|y)$

therefore:$f(\beta) = L(\beta|y)$

our proposal distribution is normal.This distribution is symmetric so $g(\beta^*|\beta) = g\beta|\beta^*)$.$\beta^*$ is our candidate $\beta^*|\beta = \beta + \epsilon$. Where $\epsilon \sim Normal$

$$g(\beta^*|\beta) = h(\beta^* - \beta) = Normal(\epsilon) = Normal(-\epsilon)$$

$$g(\beta|\beta^*) = h(\beta - \beta^*) = Normal(-\epsilon) = Normal(\epsilon)$$

$$g(\beta|\beta^*) = g(\beta^*|\beta)$$

$$R(x, x^*) = \frac{f(x^*)g(x^*|x)}{f(x)g(x|x^*)} = \frac{L(\beta^*|y) * g(\beta^*|\beta)}{L(\beta|y) * g(\beta|\beta^*)} = \frac{L(\beta^*|y)}{L(\beta|y)}$$

The markov chain has to be aperiodic and recurrent,for it have a stationary distribution.When the proposal is symmetric we also have detailed balance.

The M-H ratio makes the chain satisfy positve recurrant,and thereby irreducability also,and because of this also aperiodicity.

3.c

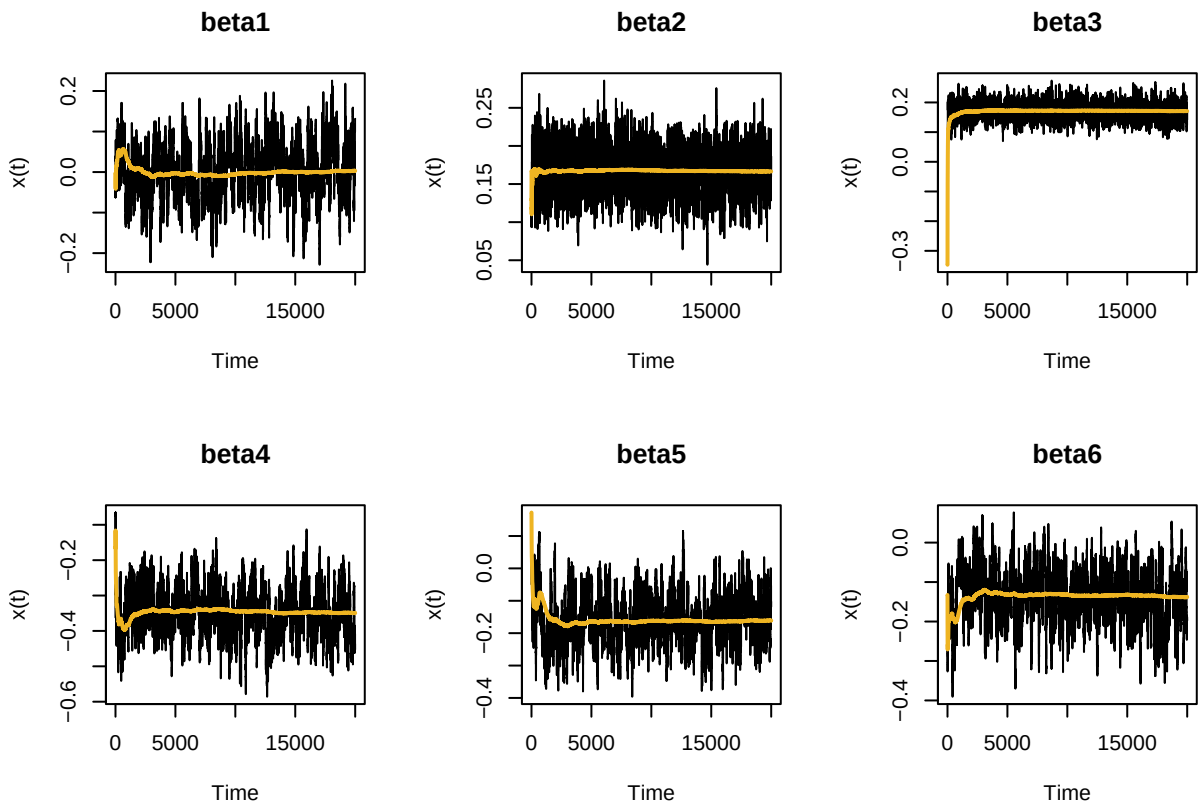
```
load("gambia.dat")


X=scale(gambia[,-c(3,5,6,8)],center = TRUE,scale = TRUE)
X=cbind(X,gambia[,c(3,5,6,8)])

initial=summary(glm(pos~age+netuse+treated+green+phc,data =data.frame(X),family=binomial(link="probit")
intersep=rep(1,length(X[,2]))
X=cbind(intersep,X[,-c(1,2,5)])
```

```
intial_beta=initial$coefficients[,1]
#runif(8)

y=gambia$pos

log_like=function(beta){
  list_y_1=which(y==1)
  list_y_0= which(y==0)

  p_i_1=as.vector(pnorm((beta)%*%t(X[list_y_1,])))
  p_i_0=as.vector(1-pnorm(beta%*%t(X[list_y_0,])))


  like=sum(log(p_i_1[which(p_i_1>0)]))+sum(log(p_i_0[which(p_i_0>0)]))
  return(like)
}



log_like(intial_beta)
```

```
## [1] -1477.742
```

```
N = 20000  # Number of iterations
beta_sec =matrix(rep(0,N*6),nrow = N,ncol = 6)

beta_sec[1,]=intial_beta
varProp=0.2^2 # variance of proposal
burn_in <- 5000



acc = 0

library("MASS")
for(i in 2:N)
{
  beta_candidate =rnorm(6,beta_sec[i-1,],varProp)



  R = min(1,exp(log_like(beta_candidate)-log_like(beta_sec[i-1,])))


  if(runif(1)<R)
  {
    beta_sec[i,] = beta_candidate
    acc = acc+1

  }else{
    beta_sec[i,] = beta_sec[i-1,]
  }
```

```
}


par(mfrow=c(2,3))

l=c("beta1","beta2","beta3","beta4","beta5","beta6")

for(i in 1:6){
plot.ts(beta_sec[,i],type="s",main=l[i],ylab="x(t)")

lines(cumsum(beta_sec[, i])/(1:N), col="goldenrod2", lwd=2)
}
```




```
post_betasec <- colMeans(beta_sec[-(1:burn_in+10000), ])

post_betasec
```

```
## [1]  0.001002521  0.166924428  0.170316202 -0.347248644 -0.156569067
## [6] -0.140205954
```

3.e

$$p(\beta|z,y) \propto \prod_{i=1}^{n}(I(z_i>0)I(y_i=1)+I(z_i<0)I(y_i=0))\phi(z_i-\beta^T x_i)$$

$$p(\beta|y) \propto \int_{-\infty}^{\infty} \prod_{i=1}^{n} (I(z_i > 0)I(y_i = 1) + I(z_i < 0)I(y_i = 0))\phi(z_i - \beta^T x_i)dz_i$$

$$= \prod_{i=1}^{n} \int_{-\infty}^{\infty} (I(z_i > 0)I(y_i = 1) + I(z_i < 0)I(y_i = 0))\phi(z_i - \beta^T x_i)dz_i$$

$$= \prod_{i=1}^{n} \int_{0}^{\infty} I(y_i = 1)\phi(z_i - \beta^T x_i)dz_i + \int_{-\infty}^{0} I(y_i = 0)\phi(z_i - \beta^T x_i)dz_i$$

only part of the integral bound is 0. Since putting in $-\infty$ or ∞ gives $erf(-\infty) = -1$ which cancels out.

$$\prod_{i=1}^{n} I(y_i = 1)(1 - \Phi(-\beta^T x_i)) + I(y_i = 0)(\Phi(-\beta^T x_i))$$

$$\prod_{i=1}^{n} I(y_i = 1)\Phi(\beta^T x_i)) + I(y_i = 0)(1 - \Phi(\beta^T x_i))$$

either z_i is greater than zero or smaller then zero so one of the expressions in the sum will be activated in the sum for each element i in the product.

if we set $p_i = \Phi(\beta^T x_i)$ we get. We get: $\prod_{i=1}^{n} I(y_i = 1)p_i + I(y_i = 0)(1 - p_i)$ which is equivalent to

$\prod_{i=1}^{n} p_i^{y_i}(1 - p_i)^{1-y_i}$

3.f

We see from the expanded posterior that the observations are iid. and also that $I(z_i > 0)$ get activated whenever $I(y_i = 1)$.Same for $I(z_i \leq 1)$ and $I(y_i = 0)$.

hence we get the expression in (10).

Since one of the expression gets activated in the sum for each i in the product.This expression this looks very much like the likelihood function of iid standard normal variables.

$$p(\beta|z, y) \propto e^{-\frac{\sum_{i=1}^{n}(z_i - \beta^T x_i)^2}{2}}$$

Differentiating this expression setting it equal to zero give: $\beta = (XX^T)^{-1}X^T z$

$$E(\beta|z, y) = E[(XX^T)^{-1}X^T z] = (XX^T)^{-1}X^T E[z] = (XX^T)^{-1}X^T \beta^T X$$

$$= (XX^T)^{-1}X^T((XX^T)^{-1}X^T z)^T X = (XX^T)^{-1}X^T z^T X(XX^T)^{-1}X$$

$$= (XX^T)^{-1}X^T z$$

$$Cov(b|, z, y) = Cov((XX^T)^{-1}Xz) = (XX^T)^{-1}XCov(z)X^T(XX^T)^{-1} = (XX^T)^{-1}$$

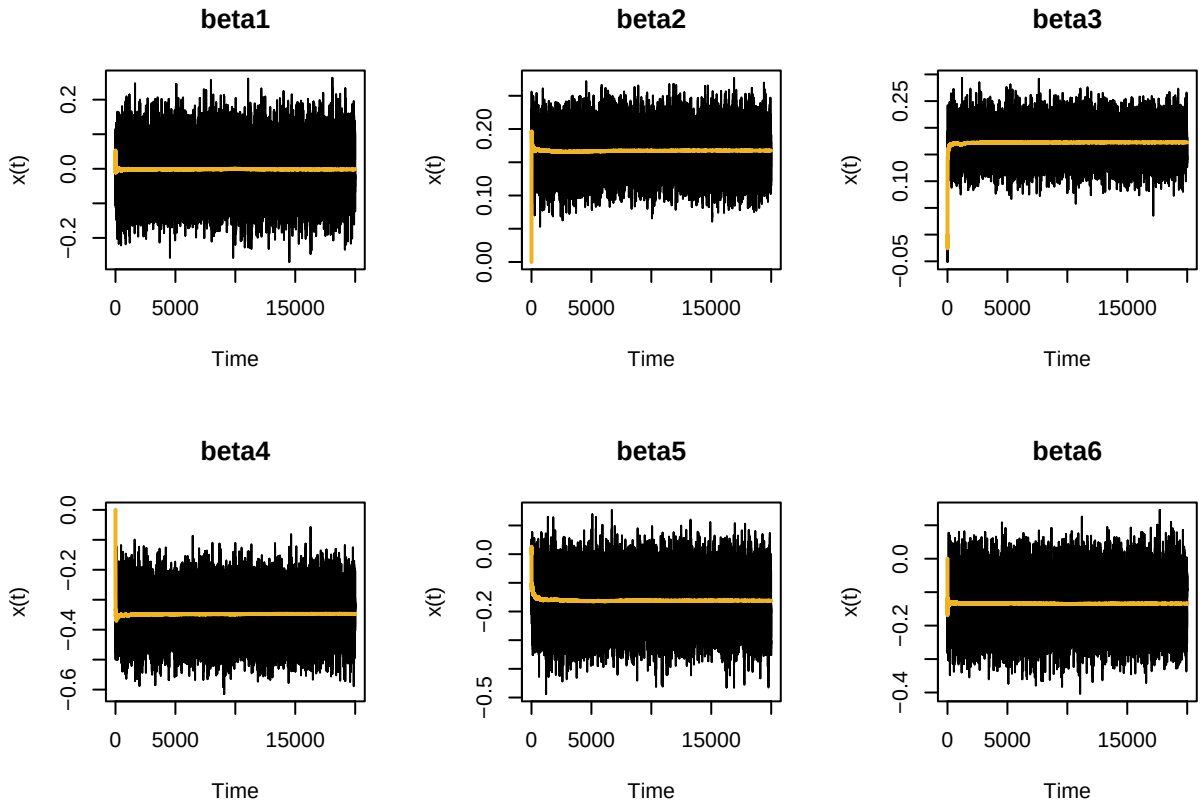
$$= Cov(z)(XX^T)^{-1}XX^T(XX^T)^{-1} = Cov(z)I(XX^T)^{-1} = (XX^T)^{-1}$$

3.g

```
load("gambia.dat")


X=scale(gambia[,-c(3,5,6,8)],center = TRUE,scale = TRUE)
X=cbind(X,gambia[,c(3,5,6,8)])

initial=summary(glm(pos~age+netuse+treated+green+phc,data =data.frame(X),family=binomial(link="probit")
intersep=rep(1,length(X[,2]))
X=cbind(intersep,X[,-c(1,2,5)])


intial_beta=initial$coefficients[,1]


y=gambia$pos


require(mvtnorm)

require(truncnorm)

N=length(y)
# Initialize parameters
beta <- initial$coefficients[,1]
z <- rep(0, N)

# Number of simulations for Gibbs sampler
N_iter <- 20000
# Burn in period
burn_in <- 5000

beta_chain <- matrix(rep(0,N_iter*6),nrow = N_iter,ncol = 6)



X=as.matrix(X)
covar <- solve(crossprod(X, X))

list_y_1=which(y==1)
list_y_0= which(y==0)

for (t in 2:N_iter) {

  mu_z <-  X%*%beta

  z[y == 0] <- rtruncnorm(length(list_y_0), mean = mu_z[y == 0], sd = 1, a = -Inf, b = 0)
  z[y == 1] <- rtruncnorm(length(list_y_1), mean = mu_z[y == 1], sd = 1, a = 0, b = Inf)


  means = solve(t(X)%*%X)%*%t(X)%*%z

  beta = c(rmvnorm(1, means, covar))
```

```
  beta_chain[t, ] = beta
}
```

```
par(mfrow=c(2,3))

l=c("beta1","beta2","beta3","beta4","beta5","beta6")
for(i in 1:6){
  plot.ts(beta_chain[,i],type="s",main=l[i],ylab="x(t)")

  lines(cumsum(beta_chain[, i])/(1:N_iter), col="goldenrod2", lwd=2)
}
```




```
post_beta <- colMeans(beta_chain[-(1:burn_in+10000), ])

post_beta
```

```
## [1] -0.0006231293  0.1673565622  0.1725935719 -0.3480784627 -0.1627597032
## [6] -0.1346408027
```

3.h,

the evaluation of the posterior parameters was done in the code above.Results looks kind of similar. I used the same burn in rate for bought.The mixing is better for the Gibbs sampler.The run-time was better for the Gibbs sampler. Bought algorithms converged pretty fast,since i started with the MLE estimates of beta.

I did not run multiple chains to confirm convergence,but since the Gibbs sampler and M-H sampler gives unambiguous results it is a good sign.