Mandatory 3

Håvard Godal, 245609

November 6, 2020

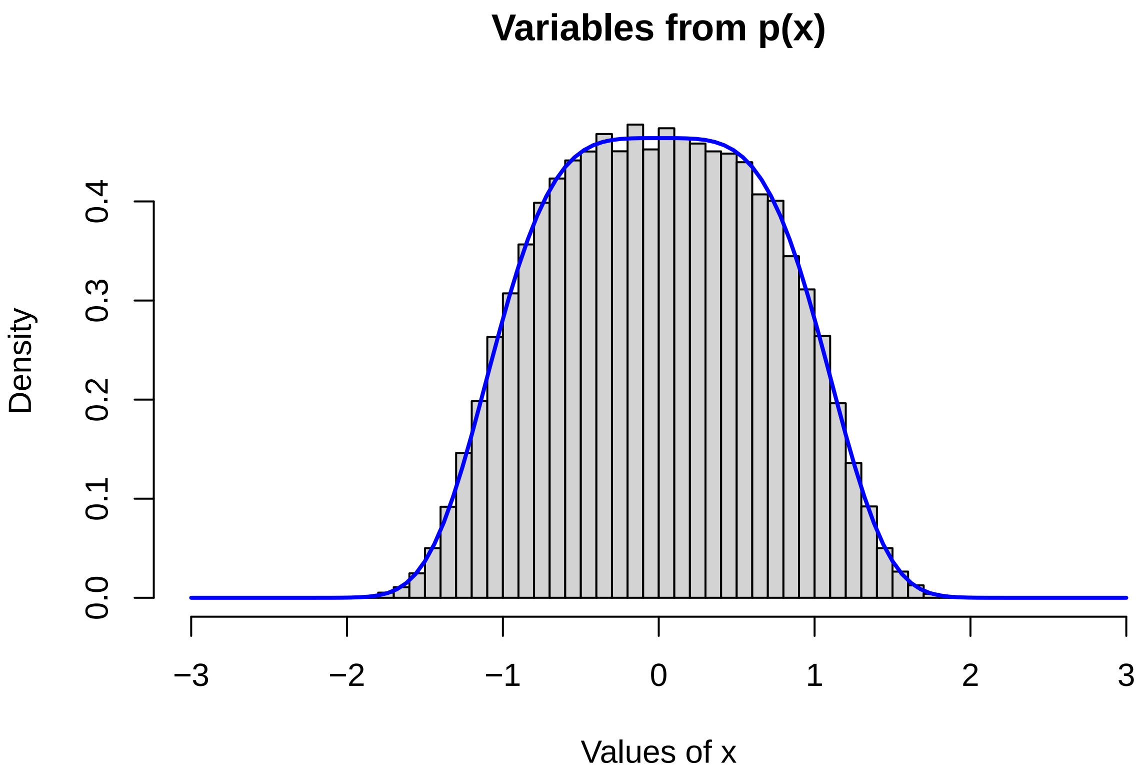
```r
rm(list = ls())
library(coda)
library(mvtnorm)
library(deSolve)
```

Problem 1: Random number generation and Monte Carlo integration Problem 1A
Generate n independent random variables from the distribution-density $p(x)$.

$$p(x) = \frac{2^{\frac{1}{4}}\Gamma\left(\frac{3}{4}\right)}{\pi} \exp\left(-\frac{x^4}{2}\right), \qquad -\infty < x < \infty$$

```r
# The probability density p(x)
p <- function(x) {
  2^(1/4)*gamma(3/4)/pi*exp(-(x^4)/2)
}

oneD.IRWMH <- function(prob, sigma, theta, Nsim = 10000){
  res <- vector(length = Nsim)
  # allocate memory
  res[1] <- theta
  # old importance weight
  wt.old <- prob(res[1])/dnorm(res[1], sd = sigma)
  Nacc <- 0
    for(i in 2:Nsim){
      # proposal (note, independent of past)
      thetaStar <- rnorm(1, sd = sigma)
      # new importance weight
      wt.star <- prob(thetaStar)/dnorm(thetaStar, sd = sigma)
      # accept probability
      alpha <- min(1.0, wt.star/wt.old)
      # accept/reject
      if(runif(1) < alpha){
        res[i] <- thetaStar
        wt.old <- wt.star
        Nacc <- Nacc + 1
      } else {
        res[i] <- res[i-1]
      }
    }
  res
}
```

```r
Nsim <- 100000
IndRandVariables <- oneD.IRWMH(p, theta = 0.0, sigma = 0.8, Nsim = Nsim)
IndRandVariables.burned <- IndRandVariables[201:length(IndRandVariables)]

# Plotting the random variables together with the probability density.
hist(IndRandVariables.burned, probability = TRUE, breaks = 50, xlim = c(-3, 3),
     main = "Variables from p(x)", xlab = "Values of x")
curve(p, add = TRUE, -3, 3, col = "blue", lwd = 2)
```



n random independent variables has to be generated. This is achieved by using the independent MH sampler, as finding the CDF and using the inverse transform method proves to be difficult for the given density distribution. The main difference between an independent MH sampler and a regular MH random walk is that the independent MH sampler preserves the independence of each random variable.

Although a regular MH random walk would return random variable values following the density in the long run, the variables would loose their independence as the regular MH random walk generates new random values based on the previous random value, making them autocorrelated.

The first part of the MCMC simulation is spent "locating" the high density region. As the task of the function is to generate values rather than "locating" the high density region, the values from the first part of the simulation can be discarded through burn-in.

Problem 1B

Generate n independent random variables from the distribution-density $p(x)$.

$$p(x) = 2x\exp(-x^2), \qquad 0 < x < \infty$$

To generate independent random variables the most effective way, inverse transform sampling should be used. This is done by first finding the CDF of $p(x)$:

$$F(x) = \int_0^x p(x)dx = \int_0^x 2x\exp(-x^2)dx = 1 - e^{-x^2}$$

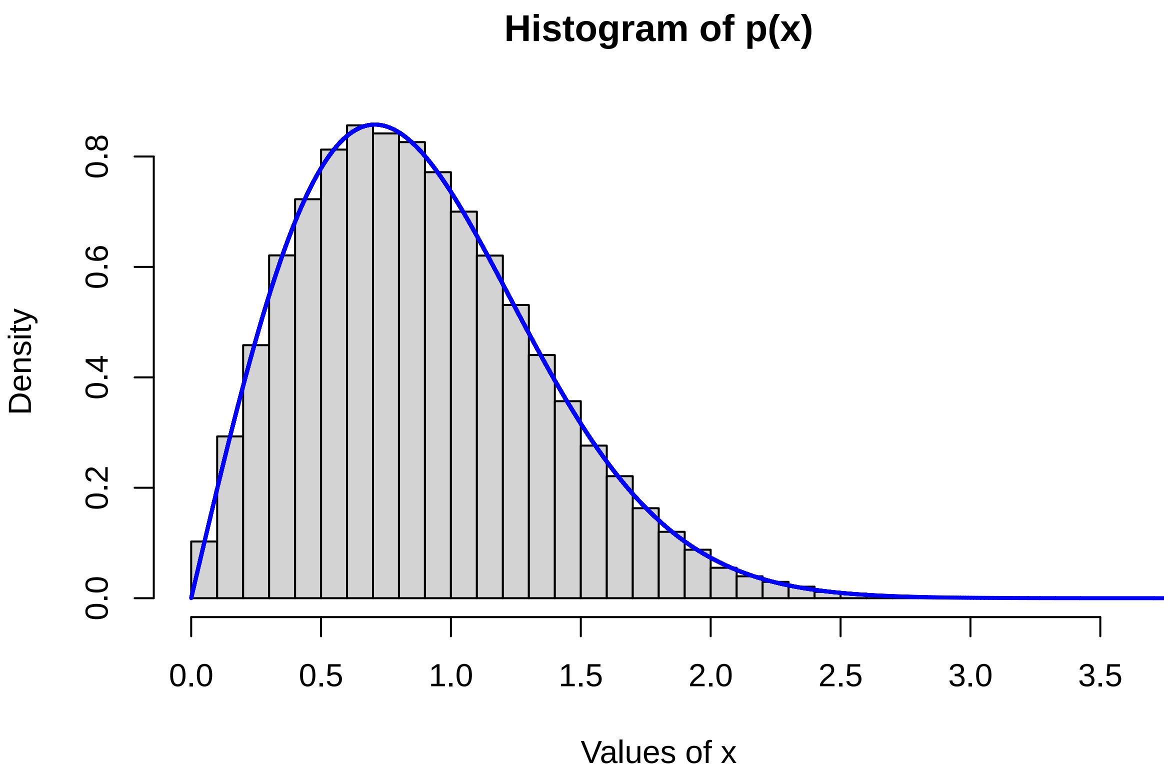
Because $0 \leq F(x) \leq 1$ and $F(x)$ is uniformly distributed, we set $F(X) = U$ and solve for x:

$$X = F^{-1}(U) = \pm\sqrt{-\log(1-U)}$$

Calculating the square root results in a positive and a negative solution. Because the problem states that $0 \leq x \leq \infty$ and the square root always gives a positive value, the negative solution is discarded.

$$X = \sqrt{-\log(1-U)}$$

```
# The probability density p(x)
p <- function(x){
  2*x*exp(-x^2)
}

Nsim <- 100000
u <- runif(Nsim)
# x-values from the inverse transform method
x.variables <- sqrt(-log(1-u))
x.variables.burned <- x.variables[201:length(x.variables)]

hist(x.variables.burned, freq = FALSE, main = 'Histogram of p(x)', xlab = 'Values of x', breaks = 50)
curve(p, 0, 4, add = TRUE, n=Nsim, col="blue", lwd=2)
```



Because the x-values are generated directly from $p(x)$ through the inverse transform method, the random variables plotted into a histogram corresponds with the curve of the distribution density.

Problem 1C

Perform importance sampling with the integral:

$$\int_0^\infty \exp(\sqrt{x})\exp(-20(x-4)^2)dx = \int_A g(x)dx$$

To perform importance sampling, $g(x)$ has to be decomposed into $g(x) = h(x)f(x)$ where $f(x)$ is a probability density on the set A.

$$\exp(\sqrt{x})\exp(-20(x-4)^2) = \sqrt{2\pi}\sigma\exp(\sqrt{x})\frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{1}{2}\frac{(x-4)^2)}{\frac{1}{40}}\right)$$

Thus resulting in:

$$h(x) = \sqrt{2\pi}\sigma\exp(\sqrt{x})$$

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma}\exp\left(-\frac{1}{2}\frac{(x-4)^2)}{\frac{1}{40}}\right)$$

$f(x)$ is therefore normally distributed with $f(x) = N(4, \frac{1}{40})$, which can be used to generate x-values.

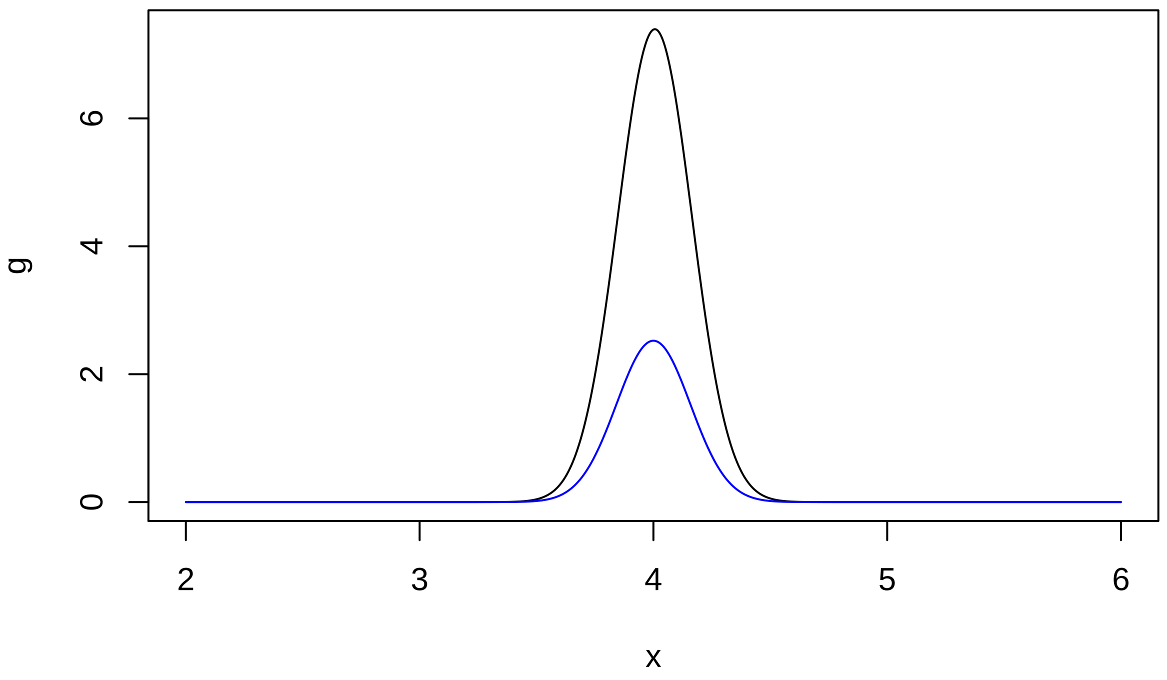
Further, we have that:

$$\int_A g(x)dx = \int_A \frac{g(x)}{f(x)}f(x)dx = E\left(\frac{g(X)}{f(X)}\right)$$

```r
# The probability density g(x)
g <- function(x){ exp(sqrt(x))*exp(-20*(x-4)^2)}

Nsim <- 10000
# Generating random variables based on f(x)
x <- rnorm(Nsim, 4, sqrt(1/40))
# Storing all values larger than 0, as specified in the problem.
x <- x[x > 0]

intiReal <- integrate(g, 0, Inf)$value
# Solving the integral by importance sampling
intiMC <- mean(g(x) / dnorm(x, 4, sqrt(1/40)))

c("Acutal integral" = intiReal, "Sampled integral" = intiMC)
```

```
##  Acutal integral Sampled integral
##         2.929669         2.927438
```

```r
plot(g, xlim = c(2, 6), n= 1000)
curve(dnorm(x, 4, sqrt(1/40)), 2, 6, n=1000, add= TRUE, col = "blue ")
```



The plot illustrates that $f(x)$ is proportional to $g(x)$, meaning that $f(x)$ can be used to generate x-values.

The result from the definite integral of $g(x)$ and the value from the importance sampling are approximately the same, showing that the importance sampling was successful.

Problem 2: Smile-shaped target Problem 2A
Use a MCMC algorithm to obtain MCMC samples.

As the problem states, a MCMC algorithm has to be used as direct sampling from the distribution is too complicated to derive.

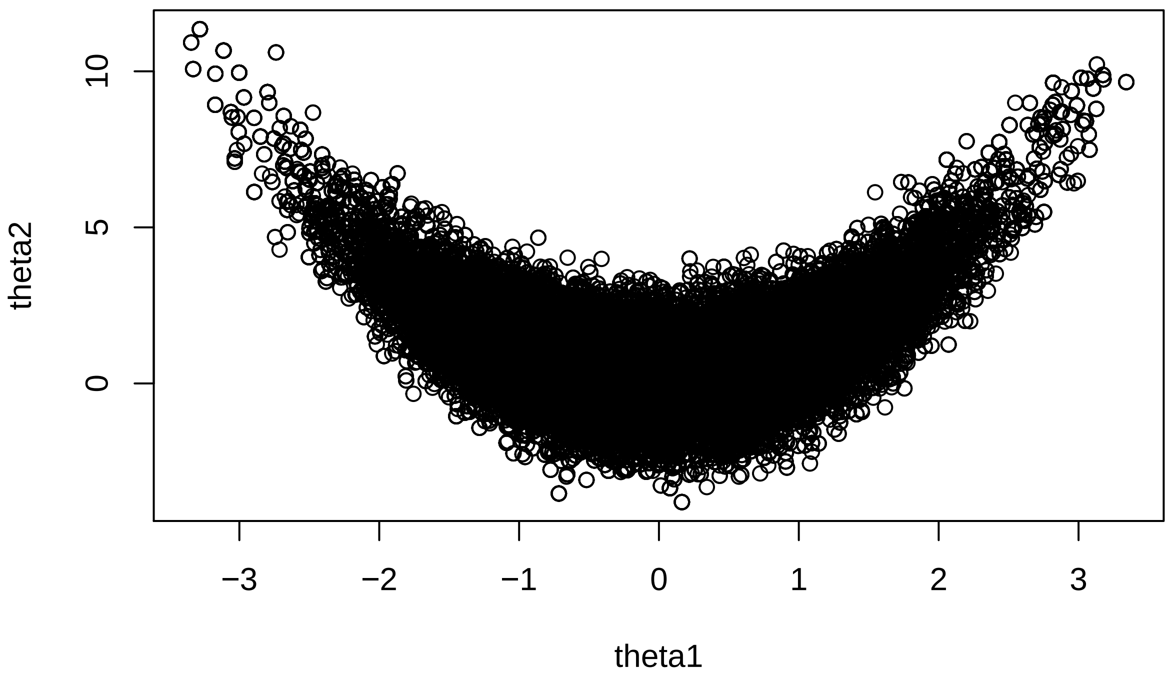
By trial and error, a MALA algorithm is shown to produce results with a very fluctuating effective sample size. Because of this, a two-dimensional random walk Metropolis Hastings algorithm (RWMH) is assumed to be most suitable for obtaining MCMC samples.

```
# Write from scratch!

# If the jumps are "too large", then the chain sticks;
# If the jumps are "too small", then the chain explores the parameter space very slower;
# We want the jumps to be just right.
# Of course the question is, what do we mean by "just right". Essentially, for a particular case they m

lp.log <- function(theta){
  -(theta[1]^2/2)-(theta[2] - theta[1]^2)^2/2 # log(g(theta))
}

Nsim <- 100000

twoDRWMH <- function(lprob, sigma, theta = c(0.0,0.0)){
  # allocate output space
  out <- matrix(0.0, Nsim, 2)
  out[1, ] <- theta

  # store old lprob
```

```r
  lp.old <- lprob(theta)

  # accept counter
  Nacc <- 0
  # main iteration loop
  for(i in 2:Nsim){
    # proposal
    # using rmvnorm and a matrix as sigma to operate at both theta values
    thetaStar <- out[(i-1), ] + rmvnorm(1, theta, diag(sigma, 2, 2))

    # evaluate
    lp.star <- lprob(thetaStar)

    # accept prob
    alpha <- exp(min(0.0, lp.star - lp.old))

    # accept/reject
    if(runif(1) < alpha && is.finite(alpha)){
      # accept
      out[i, ] <- thetaStar
      lp.old <- lp.star
      Nacc <- Nacc + 1
    } else {
      out[i, ] <- out[(i-1), ]
    }
  }

  print(c("Acceptance rate" = round(Nacc/(Nsim - 1), 3)))
  return(out)
}

twoDRWMH.result <- twoDRWMH(lprob = lp.log , sigma = 4.6)
```

```
## Acceptance rate
##            0.23
```

```r
twoDRWMH.result.burned <- twoDRWMH.result[501:length(twoDRWMH.result[ ,1]), ]
colnames(twoDRWMH.result.burned) <- c("theta1","theta2")

plot(twoDRWMH.result.burned)
```



```r
ESS <- function(x){ return(as.numeric(coda::effectiveSize(x))) }
ess <- ESS(twoDRWMH.result.burned)
c("ESS Alpha" = ess[[1]], "ESS Beta" = ess[[2]])
```

```
## ESS Alpha  ESS Beta
##  6858.390  3725.422
```

Tuning the algorithm to make it as efficient as possible is done by tweaking the sigma used. The result should give an acceptance rate close to 0.234, with the reason being that this acceptance rate minimizes the expected square jump distance, which can be seen as minimizing the autocorrelation.
By trial and error, $sigma = 4.6$ is used to make the algorithm the most efficient, and the acceptance rate close to 0.234.

The simulation has a burn-in of the first 501 values, just to be sure that the initial configuration is lost.

-> Add reference to the document linked at https://stats.stackexchange.com/questions/12155/acceptance-rates-for-metropolis-hastings-with-uniform-candidate-distribution

Problem 3: IMH for simple logistic regression problem Problem 3A
Use an independent MH sampler.

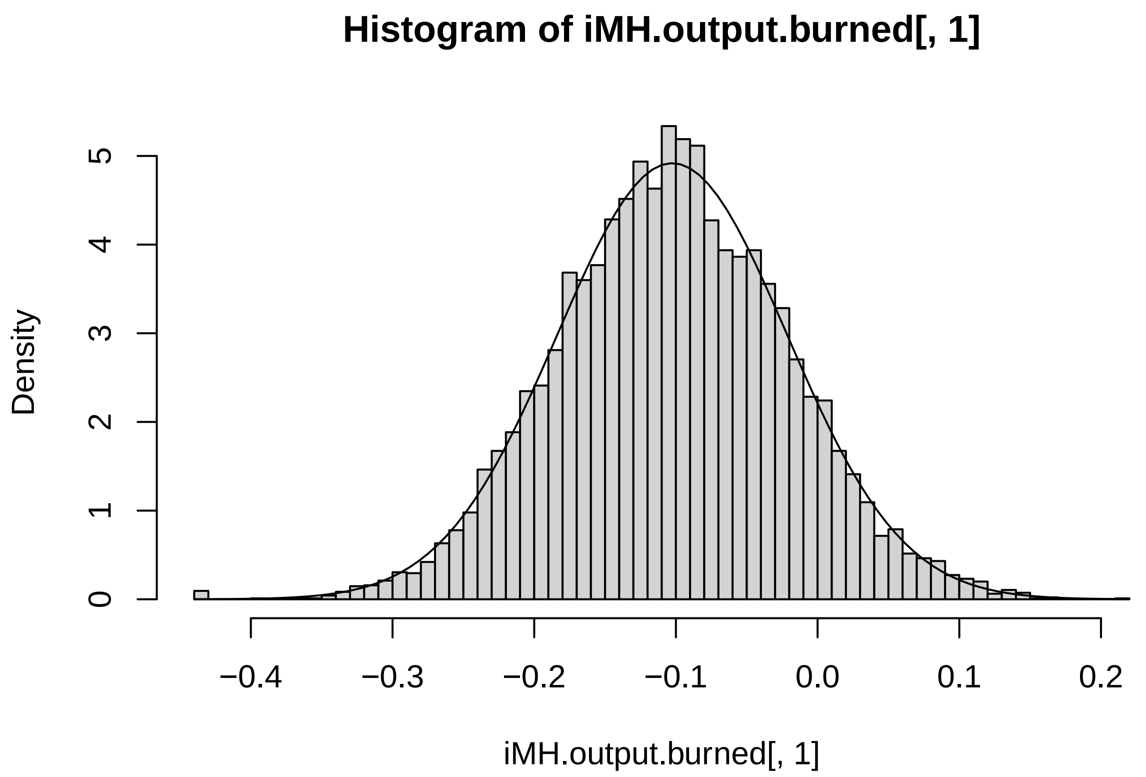
```r
df <- data.frame(read.table("logistic_regression_data.txt"))
x <- df$x
y <- df$y

# exp siden man har log av
logistic.lp <- function(theta){
    alpha <- theta[1]
    beta <- theta[2]

    # log-likelihood
    Eeta <- exp(alpha + beta * x)
    p <- Eeta/(1.0 + Eeta)
    log.like <- sum(dbinom(y, size = 1, prob = p, log = TRUE))
    # priors
    log.prior <- dnorm(alpha, sd =10, log=TRUE) + dnorm(beta, sd = 10, log = TRUE)

    # log-posterior kernel
    return(exp(log.like + log.prior))
}

Nsim <- 10000


iMH <- function(target_prob, theta, sigma, tuning) {
  res <- matrix(0, Nsim, 2)
  # allocate memory
  res[1, ] <- theta
  # old importance weight
  wt.old <- target_prob(res[1, ])/dmvnorm(res[1, ], mean = theta, sigma = sigma)
  Nacc <- 0
  for(i in 2:Nsim){
    # proposal (note, independent of past)
    thetaStar <- rmvnorm(1, mean = theta, sigma = sigma * tuning)
    # new importance weight
    wt.star <- target_prob(thetaStar)/dmvnorm(thetaStar, mean = theta, sigma = sigma)

    # accept probability
    alpha <- min(1.0, wt.star / wt.old)
    # accept/reject
    if(runif(1) < alpha){
      res[i, ] <- thetaStar
      wt.old <- wt.star
      Nacc <- Nacc + 1
    } else {
      res[i, ] <- res[i-1, ]
    }
  }
    print(c("Acceptance rate" = round(Nacc/(Nsim - 1), 3)))
  return(res)
}

sigmas <- matrix(c(0.00653, -0.00058, -0.00058, 0.01689), 2, 2)
thetas <- c(-0.102, 1.993)

iMH.output <- iMH(logistic.lp, theta = thetas, sigma = sigmas, tuning = 1) # 0.6
```

```
## Acceptance rate
##           0.966
```

```r
iMH.output.burned <- iMH.output[501:length(iMH.output[ ,1]), ]
#Burning ikke nødvendig siden det er independence

print(
  c(
    "Theta 1" = round(colMeans(iMH.output.burned)[[1]], 3),
    "Theta 2" = round(colMeans(iMH.output.burned)[[2]], 3)
  )
)
```

```
## Theta 1 Theta 2
##  -0.103   2.000
```

```
hist(iMH.output.burned[ ,1], breaks = 60, freq = FALSE)
curve(dnorm(x, mean(iMH.output.burned[,1]), sd(iMH.output.burned[,1])), add=TRUE)
```



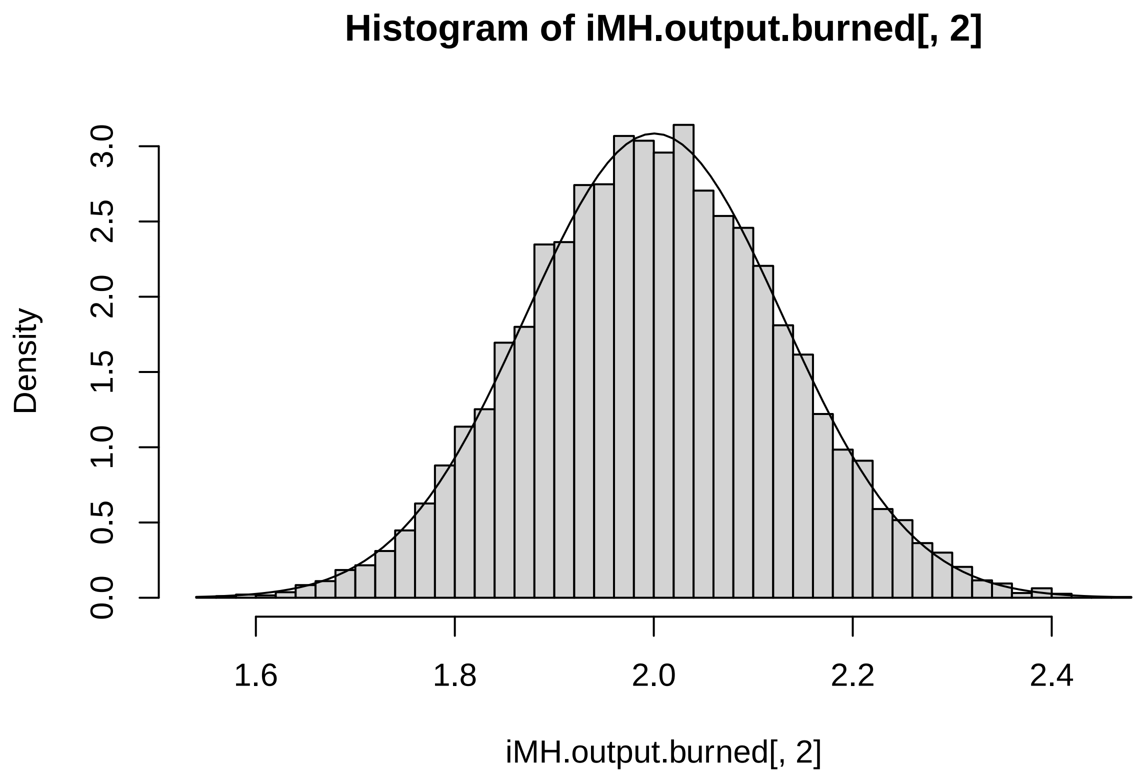
```
print(var(iMH.output.burned[ ,1]))
```

```
## [1] 0.006578421
```

```
print(var(iMH.output.burned[ ,2]))
```

```
## [1] 0.01672494
```

```
hist(iMH.output.burned[ ,2], breaks = 60, freq = FALSE)
curve(dnorm(x, mean(iMH.output.burned[,2]), sd(iMH.output.burned[,2])), add=TRUE)
```



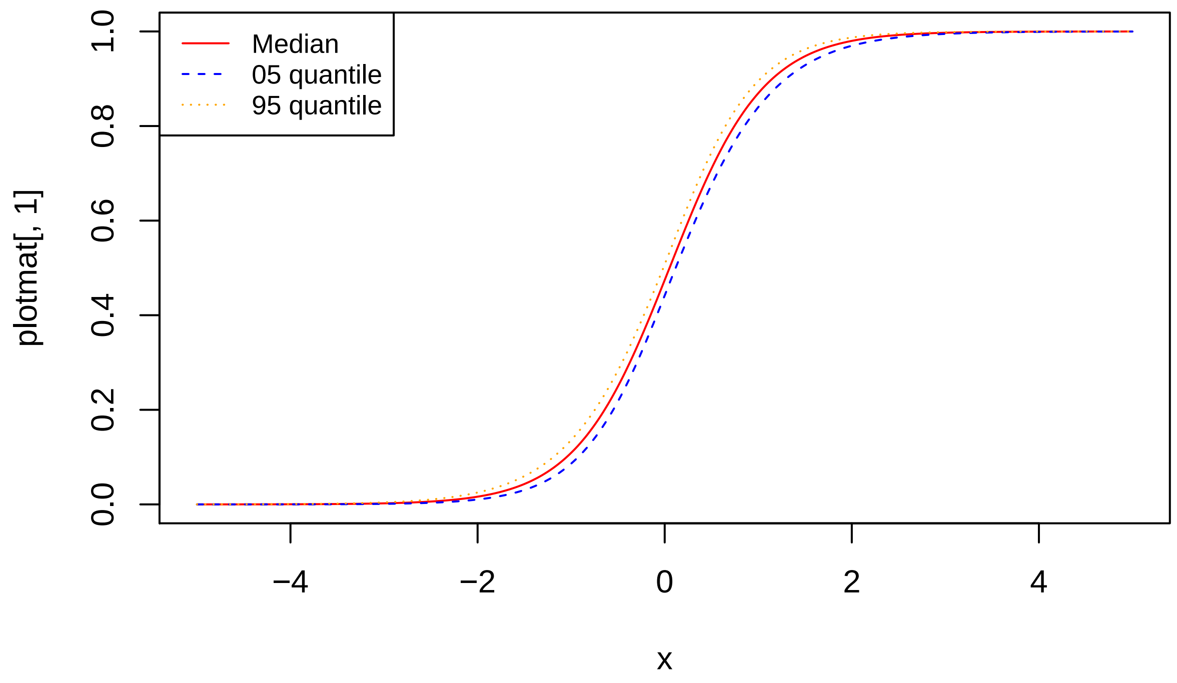
```
# plot(iMH.output.burned,pch=20,cex=0.1,xlab="MCMC iteration #")

# cov(iMH.output.burned)

ess <- ESS(iMH.output.burned)
c("ESS Theta 1" = ess[[1]], "ESS Theta 2" = ess[[2]])
```

```
## ESS Theta 1 ESS Theta 2
##    7939.885    7944.012
```

Problem 3B

Problem 3B

```
plotdist <- function (x) {
  mstar <- exp(iMH.output.burned[,1] + iMH.output.burned[,2]*x) / (1 + exp(iMH.output.burned[,1] + iMH.
  med <- median(mstar)
  quant05 <- quantile(mstar, 0.05)
  quant95 <- quantile(mstar, 0.95)
  # Used in problem 3c)
  quant01 <- quantile(mstar, 0.01)
  return(c(med, quant05, quant95, quant01))
}

x <- seq(-5, 5, length.out = length(iMH.output.burned[ ,1]))
plotmat <- matrix(0, length(x), 4)
i <- 0
for (val in x) {
  i <- i+1
  plotmat[i,] <- plotdist(val)
}

plot(x, plotmat[, 1], type="l", col="red") #, xlim = c(0.6, 1), ylim = c(0.6, 0.9))
lines(x, plotmat[, 2], type="l", col="blue", lty=2)
lines(x, plotmat[, 3], type="l", col="orange", lty=3)
legend("topleft", legend=c("Median", "05 quantile", "95 quantile"),
       col=c("red", "blue", "orange"), lty=1:4, cex=0.8)
```

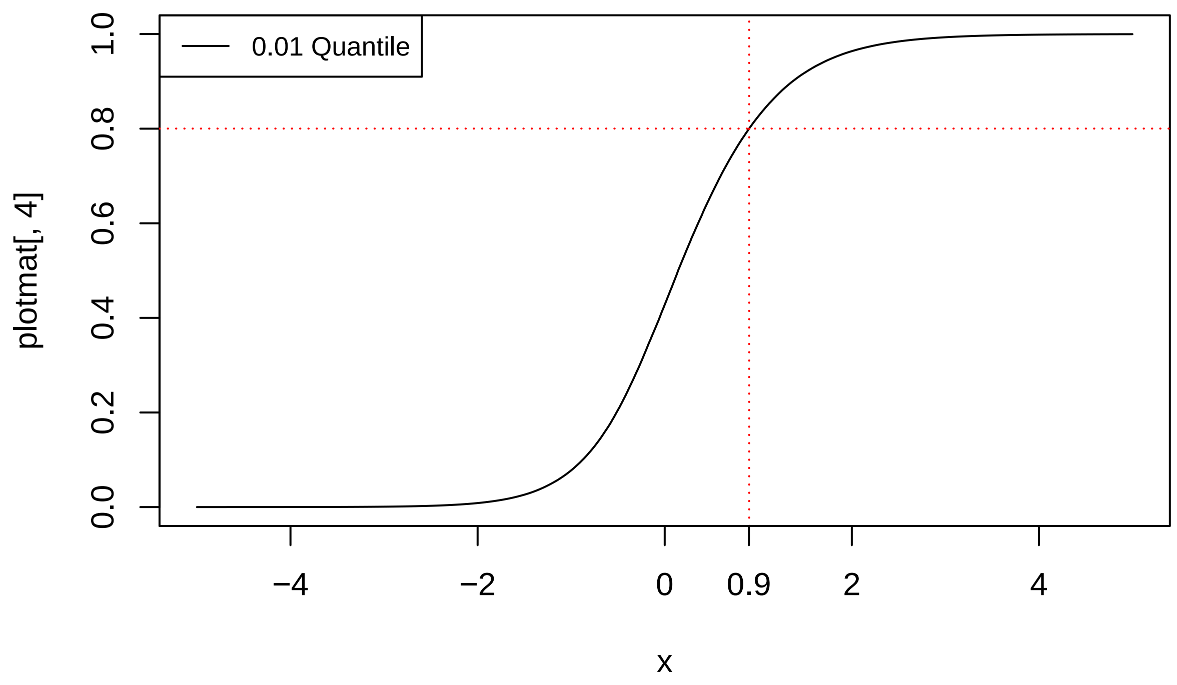


Problem 3C

```
lowestValue <- plotmat[, 4][plotmat[, 4] > 0.8][1]

# First x-value of m-value with 99% probability of being larger than 0.8
firstOccurence <- x[Position(function(x) x > 0.8, plotmat[, 4])]
firstOccurence
```

```
## [1] 0.9027266
```

```
firstOccurence.rounded <- round(firstOccurence, 2)

plot(x, plotmat[, 4], type="l", col="black") #, xlim = c(0.6, 1), ylim = c(0.6, 0.9))
abline(v = firstOccurence, col="red", lty = 3)
abline(h = lowestValue, col="red", lty = 3)
axis(1, at=firstOccurence.rounded, labels=firstOccurence.rounded)
legend("topleft", legend=c("0.01 Quantile"),
       col=c("black"), lty = 1 , cex=0.8)
```

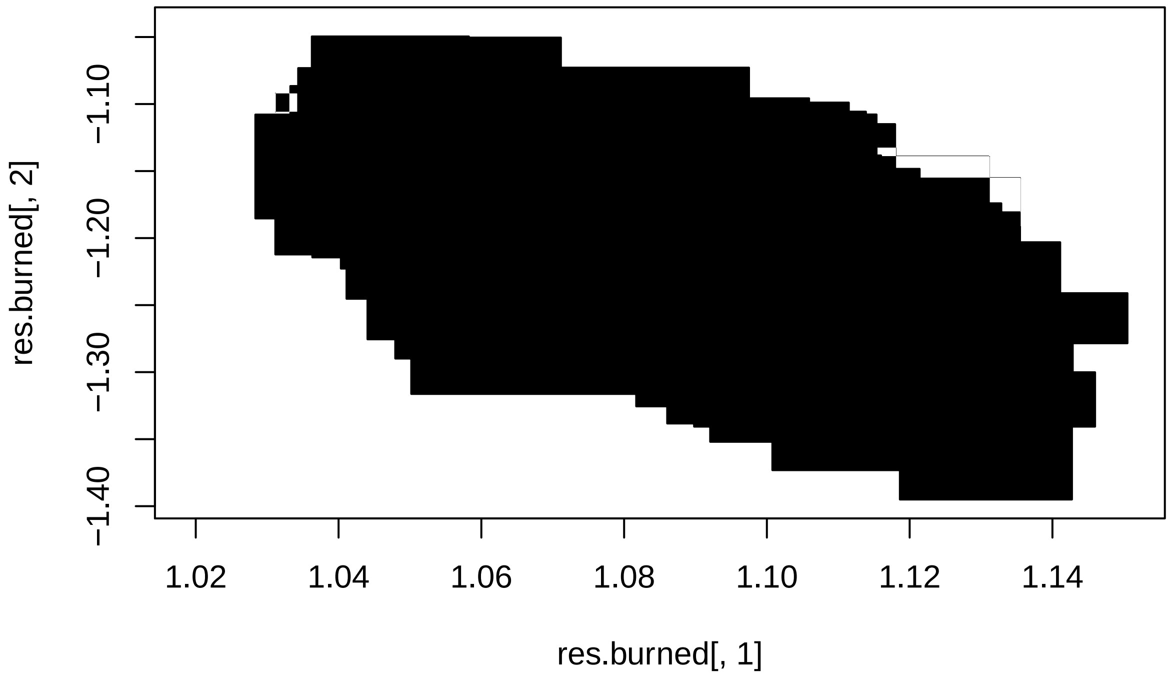


Problem 4A Problem 4B

```
df <- data.frame(read.table(file="linear_regression_data.txt"))
x <- df$x
y <- df$y


#
# Gibbsampler for bivariate normal target
#
n <- 100000 # number of Gibbs iterations
theta <- c(1, 1, 1) # variable used as state throughout
res <- matrix(0.0, 3*n+1, 3) # results, note stores twice per iteration
res[1, ] <- theta
k <- 2 # store counter

for(i in 1:n){
    # update first block
    theta[1] <- rnorm(1, theta[3]*sum(y-theta[2]*x) / (theta[3]*length(y) + 1/100), 1 / (theta[3]*lengt

    # store state
    res[k, ] <- theta
    k <- k+1

    # update second block
    theta[2] <- rnorm(1, theta[3]*sum(x*y-x*theta[1]) / (theta[3]*sum(x^2) + 1/100), 1 / (theta[3]*sum(
    # store state
    res[k, ] <- theta
    k <- k+1

     # update third block
    theta[3] <- rgamma(1, shape = length(y)/2 + 1, scale = 1/(1/2 * sum((y-theta[1]-theta[2]*x)^2) + 1)
    # store state
    res[k, ] <- theta
    k <- k+1
}

res.burned <- res[501:length(res[ ,1]), ]

# cov(res.burned)
round(colMeans(res.burned), 5)
```

```
## [1]  1.08840 -1.20637  1.09634
```

```
# hist(res.burned[,1], breaks = 100, freq = FALSE)
# hist(res.burned[,2], breaks = 100, freq = FALSE)
# hist(res.burned[,3], breaks = 100, freq = FALSE)

plot(res.burned[,1], res.burned[,2], type="l")
```



Problem 4C

```
lm.out <- lm(y~x,data=df)
c("Alpha" = lm.out$coefficients[[1]], "Beta" = lm.out$coefficients[[2]])
```

```
##    Alpha     Beta
##  1.08871 -1.20703
```

```
# Se etter a3chain for geweke
```

Problem 4D

```
n <- 10000 # number of Gibbs iterations
theta <- c(1, -1.2, 1.0) # variable used as state throughout
res <- matrix(0.0, 2*n+1, 3) # results, note stores twice per iteration
res[1, ] <- theta
k <- 2 # store counter
```

```
for(i in 1:n){
  cVal = matrix(c(-length(y)*theta[2]-0.01, -theta[2]*sum(x), -theta[2]*sum(x), -theta[2]*sum(x^2)-0.01
  bVal = c(theta[2]*sum(y), theta[2]*sum(y*x))

  # update first block
  theta[1:2] <- rmvnorm(1, -solve(cVal) %*% bVal, -solve(cVal))

  # store state
  res[k,] <- theta
  k <- k+1


  # update the third block
  theta[2] <- rgamma(1, shape = length(y)/2 + 1, scale = 1/(1/2 * sum((y-rnorm(1, 0, 10) - rnorm(1, 0,
  #store state
  res[k,] <- theta
  k <- k+1
}
```

```
## Warning in rmvnorm(1, -solve(cVal) %*% bVal, -solve(cVal)): sigma is numerically
## not positive semidefinite
```

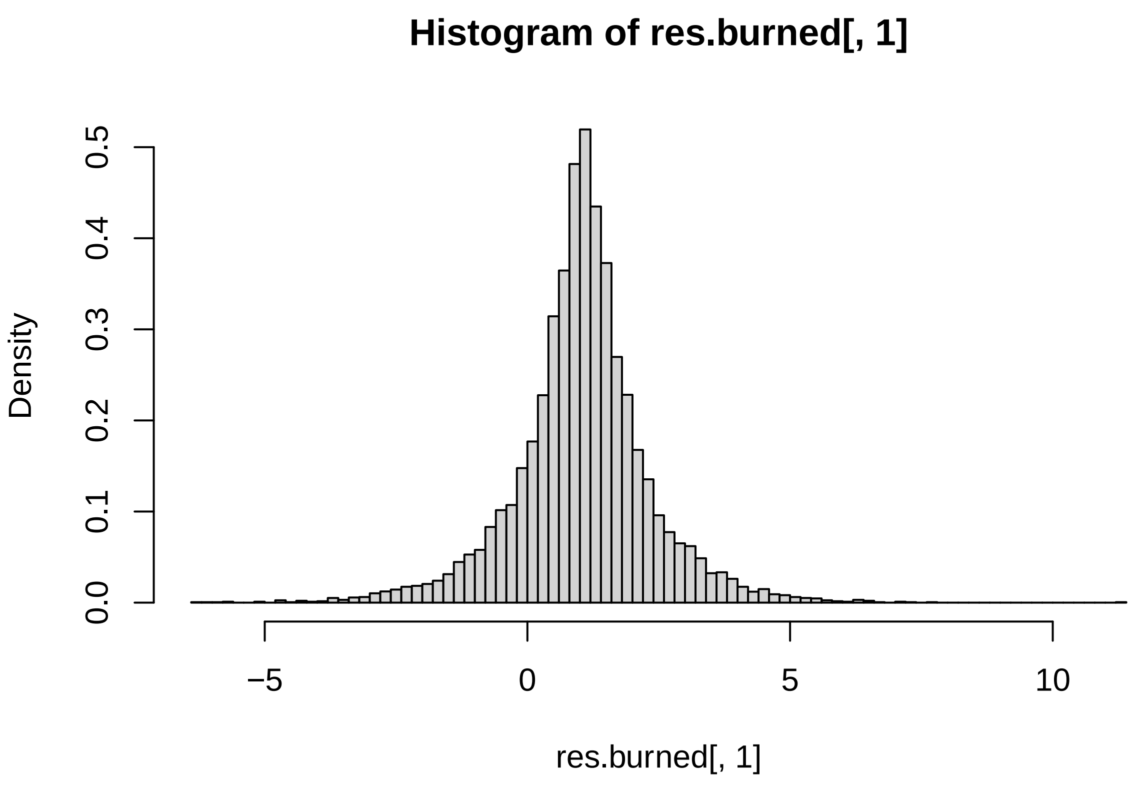
```
res.burned <- res[501:length(res[ ,1]), ]


cov(res.burned)
```

```
##            [,1]      [,2] [,3]
## [1,]  1.643269 -0.760817    0
## [2,] -0.760817  3.024782    0
## [3,]  0.000000  0.000000    0
```

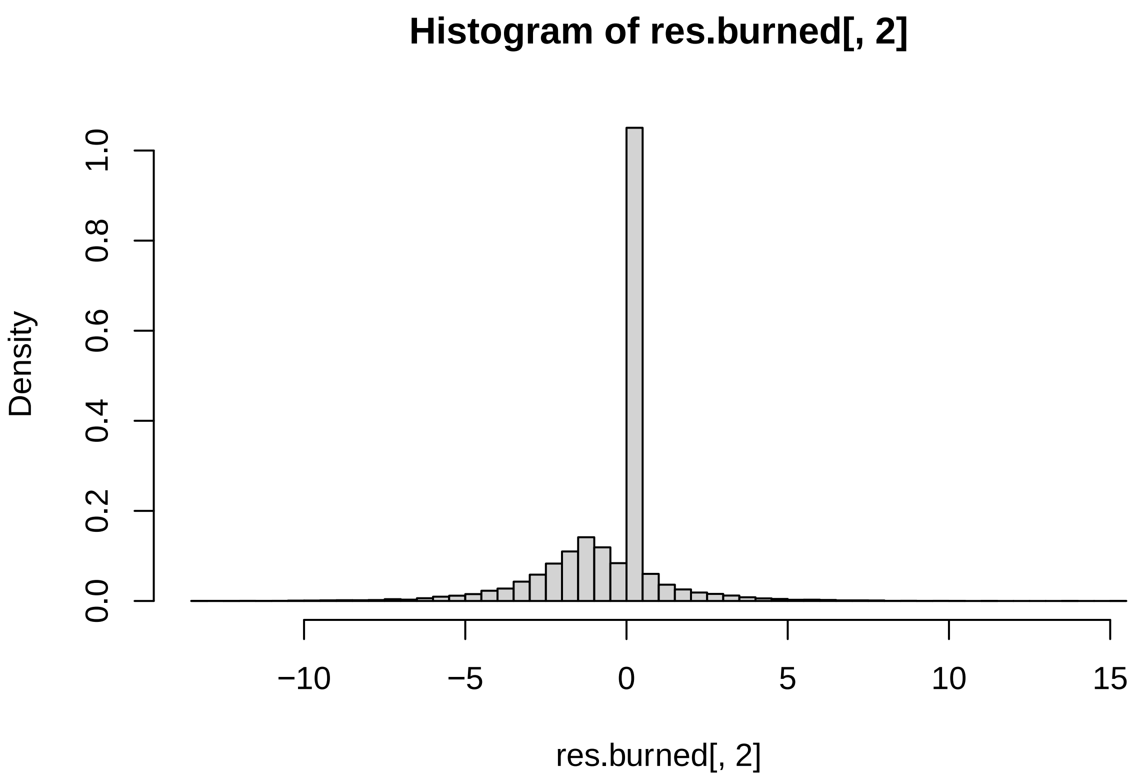
```
round(colMeans(res.burned), 5)
```

```
## [1]  1.04265 -0.52813  1.00000
```

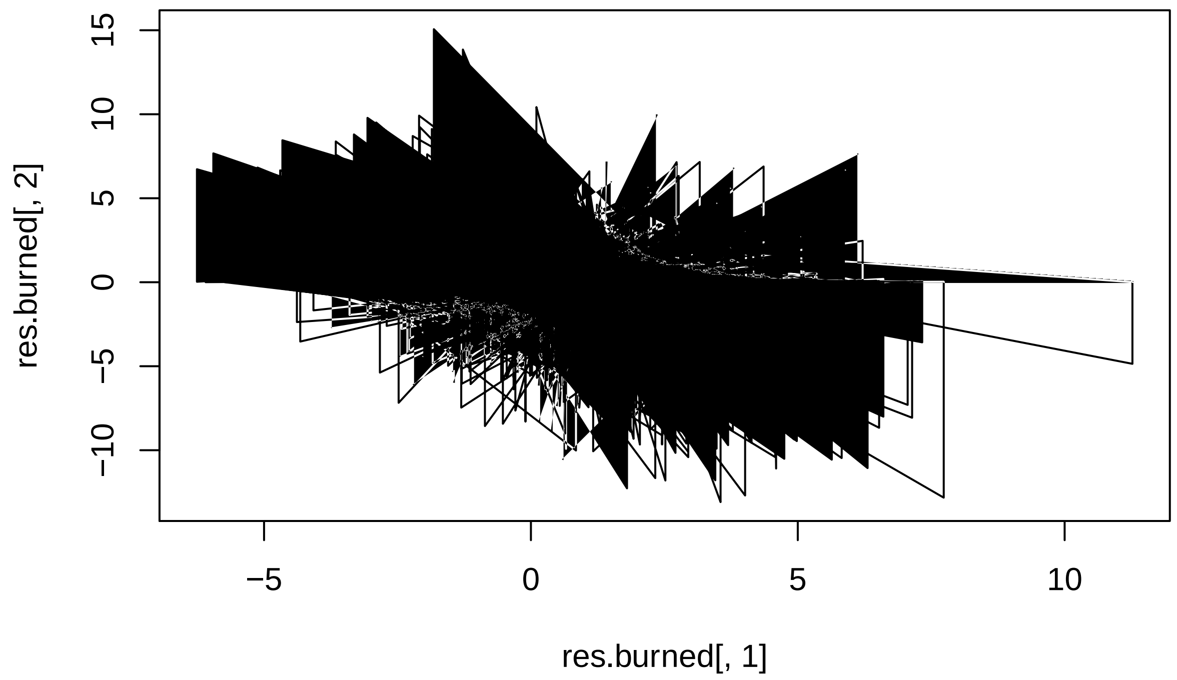
```
hist(res.burned[,1], breaks = 100, freq = FALSE)
```



```
hist(res.burned[,2], breaks = 100, freq = FALSE)
```



```
plot(res.burned[,1], res.burned[,2], type="l")
```



Problem 4E :D

Problem 5A

```
df <- data.frame(read.table(file="linear_regression_data.txt"))
x <- df$x
y <- df$y

library(boot)
sdestfunc <- function(data,i) {
  lm(data[i, 1]~data[i, 2],data = df)$coefficients
}

boot.obj <- boot(data=df, statistic = sdestfunc, R=5000)
boot.obj
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = df, statistic = sdestfunc, R = 5000)
##
##
## Bootstrap Statistics :
##      original        bias    std. error
## t1*  1.08871 -0.001405444   0.1094006
## t2* -1.20703 -0.002278254   0.2182345
```

```
boot.ci(boot.obj,type=c("norm","basic","perc","bca"))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 5000 bootstrap replicates
##
## CALL :
```

```
## boot.ci(boot.out = boot.obj, type = c("norm", "basic", "perc",
##     "bca"))
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 0.876,  1.305 )   ( 0.879,  1.300 )
##
## Level     Percentile            BCa
## 95%   ( 0.877,  1.298 )   ( 0.878,  1.300 )
## Calculations and Intervals on Original Scale
```

```
# plot(boot.obj)

sigma <- 1/sqrt(
  rgamma(5000, shape = length(y)/2 + 1,
  scale = 1/(1/2 * sum((y - mean(boot.obj$t[,1]) - mean(boot.obj$t[,2])*x)^2) + 1))
)

# Claim 1
max(sigma)
```

```
## [1] 1.28166
```

```
# Claim 2
mean(boot.obj$t[,1])
```

```
## [1] 1.087305
```

```
var(boot.obj$t[,1])
```

```
## [1] 0.01196848
```

```
# Claim 3
mean(boot.obj$t[,1] + boot.obj$t[,2])
```

```
## [1] -0.1220042
```

```
# hist(a, breaks=50)
```

Problem 6A