Mandatory 3

Håvard Godal, 245609

November 9, 2020

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

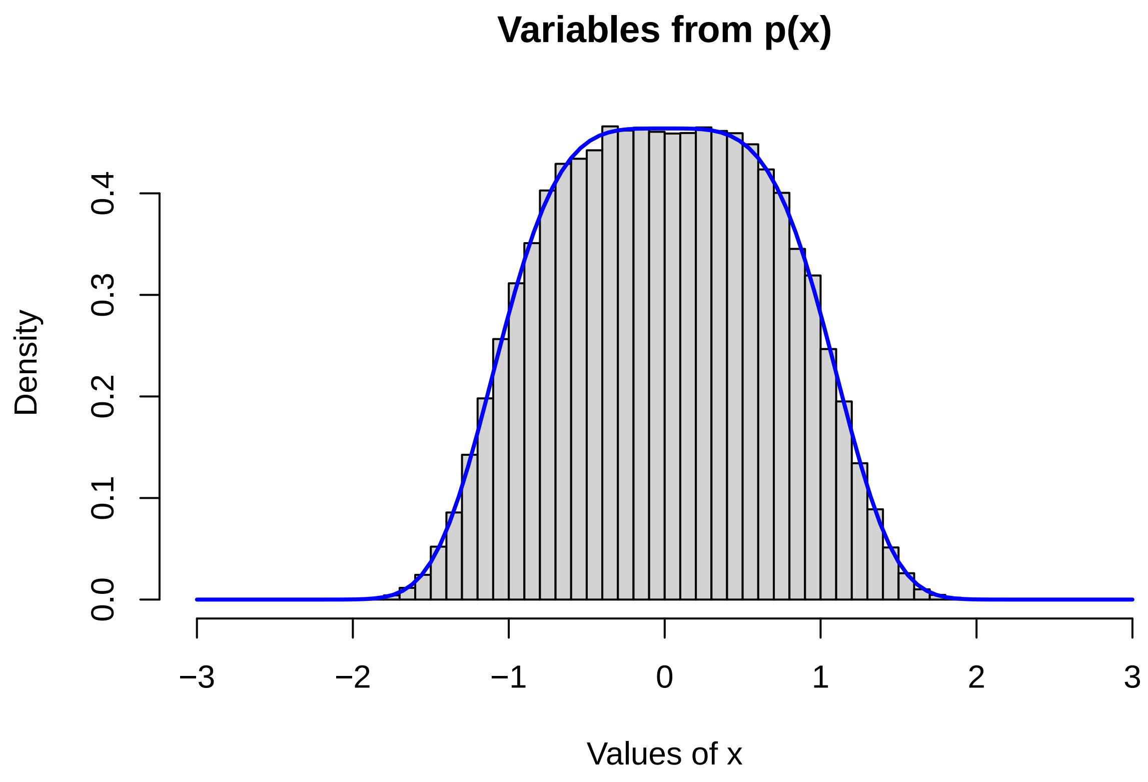
Note: For most of the simulations regarding random walk, burn-in has been used, although it is not necessary for all types of random walk. As long as the effective sample size is sufficient, burn-in is OK to use, which is why it has been used in most of the simulations.

The reason burn-in is used in some random walks is because the first part of the MCMC simulation is spent "locating" the high density region. The process of "locating" the high density is not of interest, and is therefore removed by burn-in.

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
}

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.

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 $p(x)$:

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

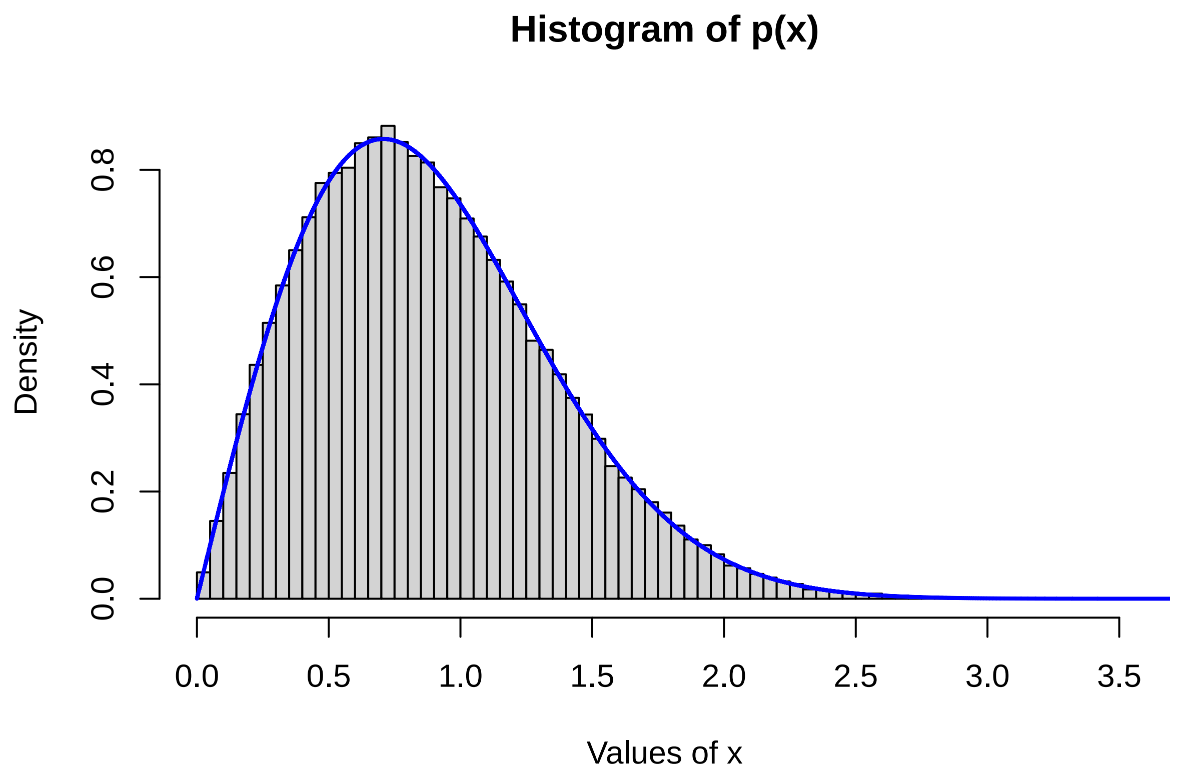
Because $0 \leq F(x) \leq 1$ and $F(x)$ is uniformly distributed, we set $F(X) = U$ and solve for x:

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

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.

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

```r
# 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.

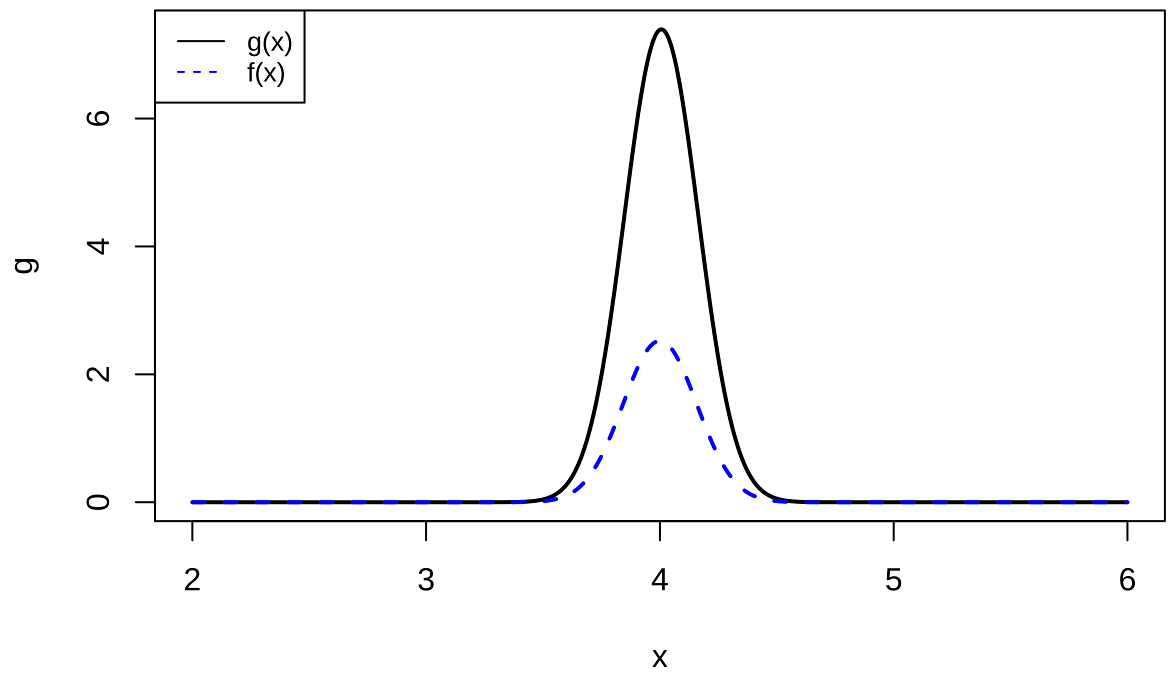
Further, we have that:

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

```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         2.928545
```

```r
plot(g, xlim = c(2, 6), n= 1000, lwd = 2)
curve(dnorm(x, 4, sqrt(1/40)), 2, 6, n=1000, add= TRUE, col = "blue ", lty = 2, lwd = 2)
legend("topleft", legend=c("g(x)", "f(x)"),
       col=c("black", "blue"), lty = c(1, 2) , cex=0.8)
```



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 done properly.

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

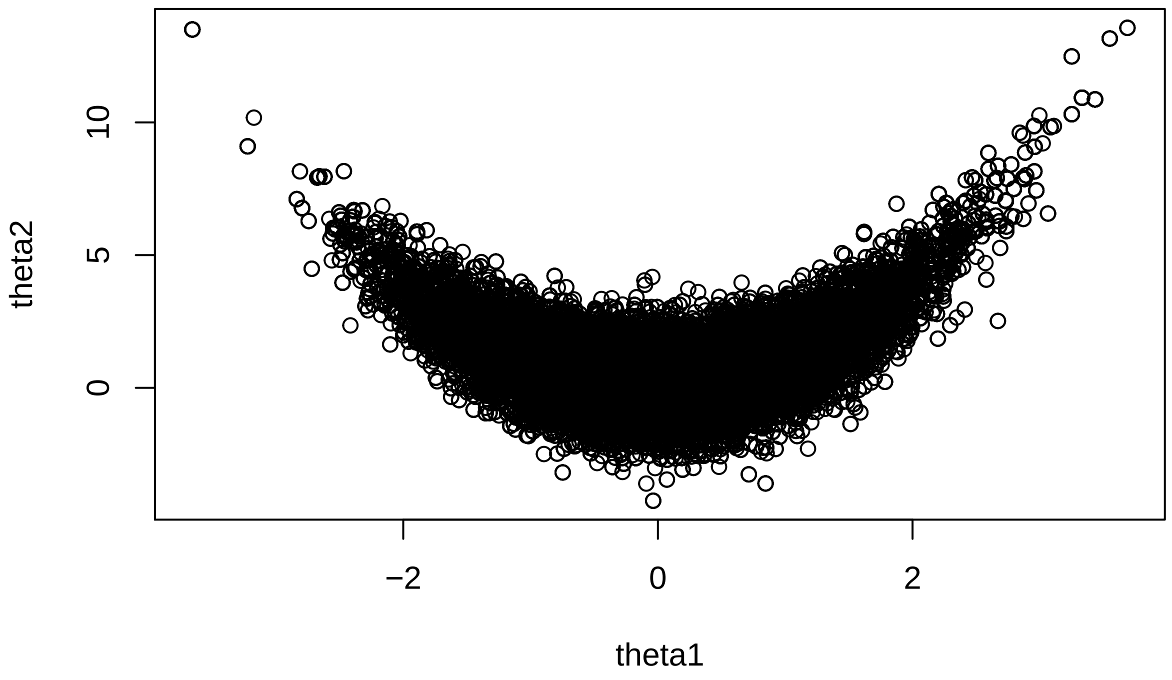
As the problem states, a MCMC algorithm has to be used as direct sampling from the distribution is too complicated to derive.

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.

```r
# Write from scratch!

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

Nsim <- 50000

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
  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), 4)))
  return(out)
}

twoDRWMH.result <- twoDRWMH(lprob = lp.log , sigma = 4.3)
```

```
## Acceptance rate
##          0.2387
```

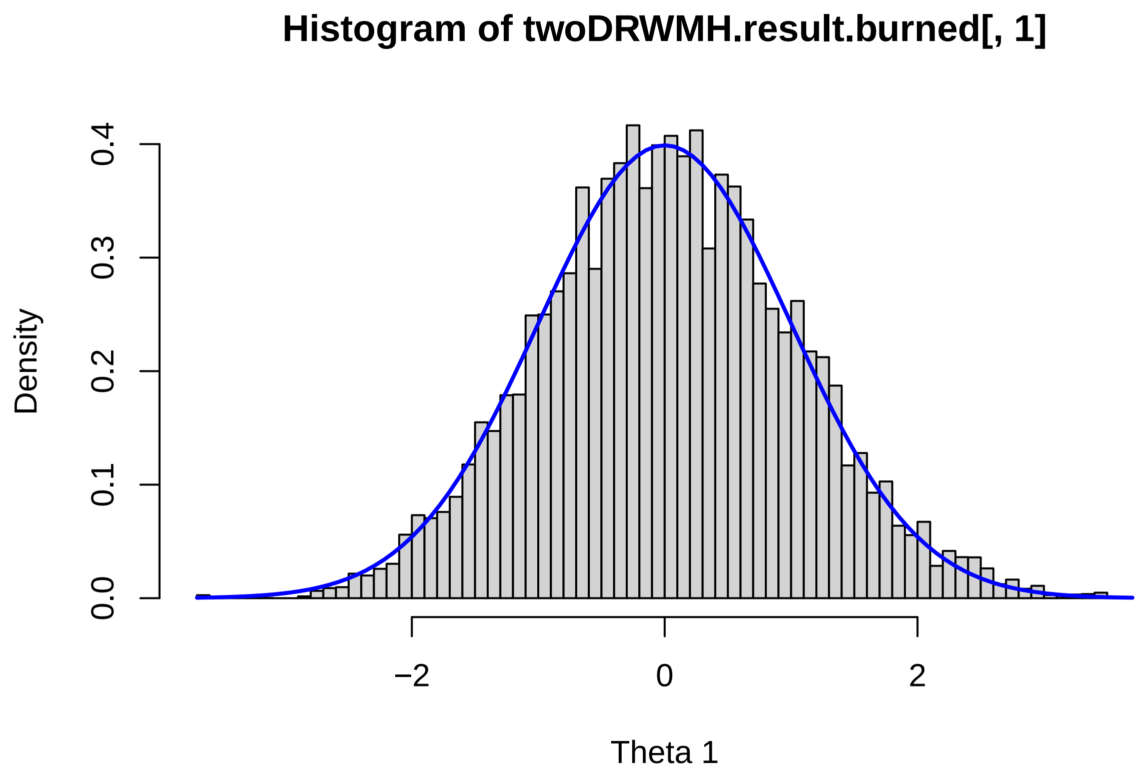
```
twoDRWMH.result.burned <- twoDRWMH.result[501:length(twoDRWMH.result[ ,1]), ]
colnames(twoDRWMH.result.burned) <- c("theta1","theta2")

plot(twoDRWMH.result.burned)
```



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

```
## ESS Theta 1 ESS Theta 2
##    3037.253    1622.077
```

```r
hist(twoDRWMH.result.burned[,1], xlab="Theta 1", freq = FALSE, breaks = 60)
curve(dnorm(x, mean = 0, sd = 1), col = "blue", add = TRUE, lwd = 2)
```



Tuning the algorithm to make it as efficient as possible is done by tweaking the sigma used, which defines how large the jumps are. If the jumps are "too large" the chain sticks, and if the jumps are "too small" the chain explores the parameter space very slowly. According to Sherlock and Roberts, the acceptance rate should be 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, $sigma = 4.3$ is used to make the algorithm the most efficient, and the acceptance rate close to 0.234.

The histogram of theta 1 clearly illustrates that the theta1-marginal distribution of is still normally distributed with parameters $\mu = 0, \sigma^2 = 1$.

After the burn-in, the output shows that the effective sample size, both for alpha and beta, are over 1000.

-> Add reference to the document linked at https://www.cambridge.org/core/journals/journal-of-applied-probability/article/optimal-scaling-of-the-random-walk-metropolis-general-criteria-for-the-0234-acceptance-rule/52670E8E3E61E8D920DC98E79D3E2CE5

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

```r
library(mvtnorm)

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(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 <- exp(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 <- exp(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)
}
```

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

# Not necessarry to use burn-in, but a good rule of thumb
iMH.output <- iMH(logistic.lp, theta = thetas, sigma = sigmas, tuning = 0.8)
```

```
## Acceptance rate
##           0.974
```

```r
iMH.output.burned <- iMH.output[501:length(iMH.output[ ,1]), ]

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

```
## Theta 1 Theta 2
##  -0.102   1.998
```

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

# 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 Alpha" = ess[[1]], "ESS Beta" = ess[[2]])
```

```
## ESS Alpha  ESS Beta
##  8897.975  8240.060
```

As the function *logistics.lp* returns the log-posterior kernel, taking $e^{logistics.lp}$ allows the independent MH sampler to operate with "normal" values instead of log-values.

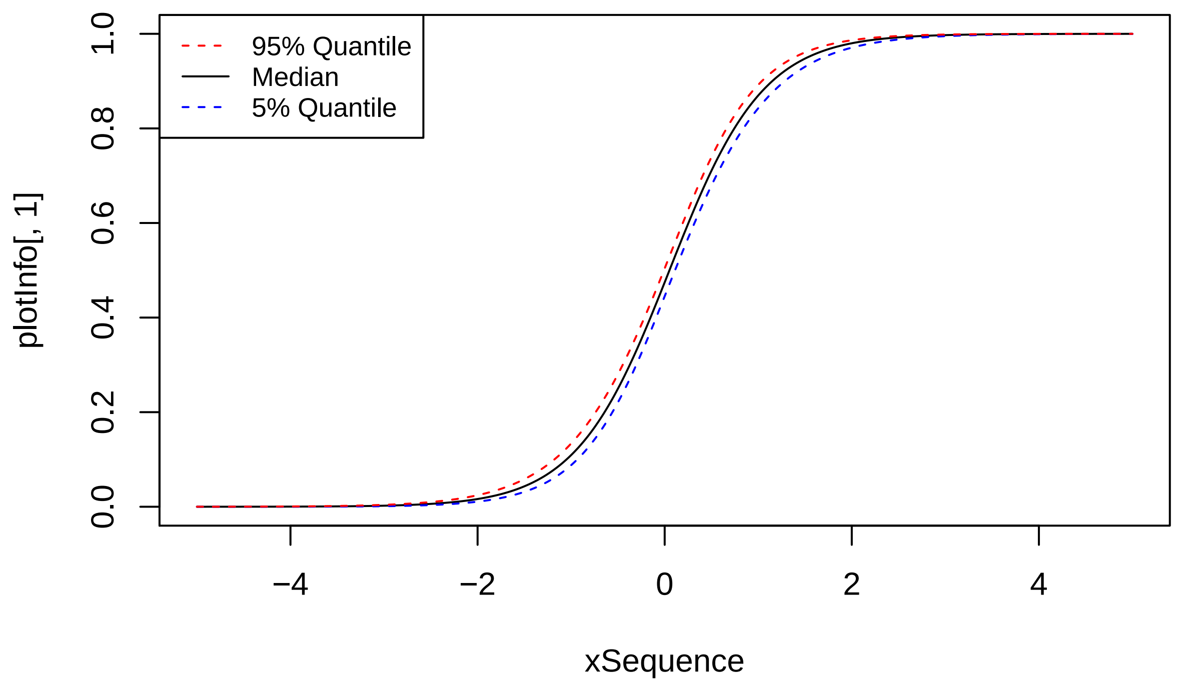
Say something about the acceptance rate and the tuning.

The mean from the independent MH sampler is approximately the same as the point estimate from the initial maximum likelihood-based classical analysis.

The effective sample size of both alpha and beta are well over 1000.

Problem 3B
Plot the median, 5% quantile, and the 95% quantile.

```
mFunc <- function(x_i) {
  exp(alpha + beta * x_i) / (1 + exp(alpha + beta * x_i))
}

stepInfo <- function(x_i) {
  mstar <- exp(iMH.output.burned[ , 1] + iMH.output.burned[ , 2] * x_i) / (1 + exp(iMH.output.burned[ ,
  med <- median(mstar)
  # Quant01 is used in problem 3c)
  quant01 <- quantile(mstar, 0.01)
  quant05 <- quantile(mstar, 0.05)
  quant95 <- quantile(mstar, 0.95)
  return(c(med, quant01, quant05, quant95))
}

xSequence <- seq(-5, 5, length.out = length(iMH.output.burned[ ,1]))
plotInfo <- matrix(0, length(xSequence), 4)
i <- 0


for (i in 1:length(xSequence)) {
  plotInfo[i,] <- stepInfo(xSequence[i])
}

plot(xSequence, plotInfo[, 1], type="l", col="black")
lines(xSequence, plotInfo[, 3], type="l", col="blue", lty=2)
lines(xSequence, plotInfo[, 4], type="l", col="red", lty=2)
legend("topleft", legend=c("95% Quantile","Median", "5% Quantile"),
       col=c("red", "black", "blue"), lty=c(2, 1, 2), cex=0.8)
```

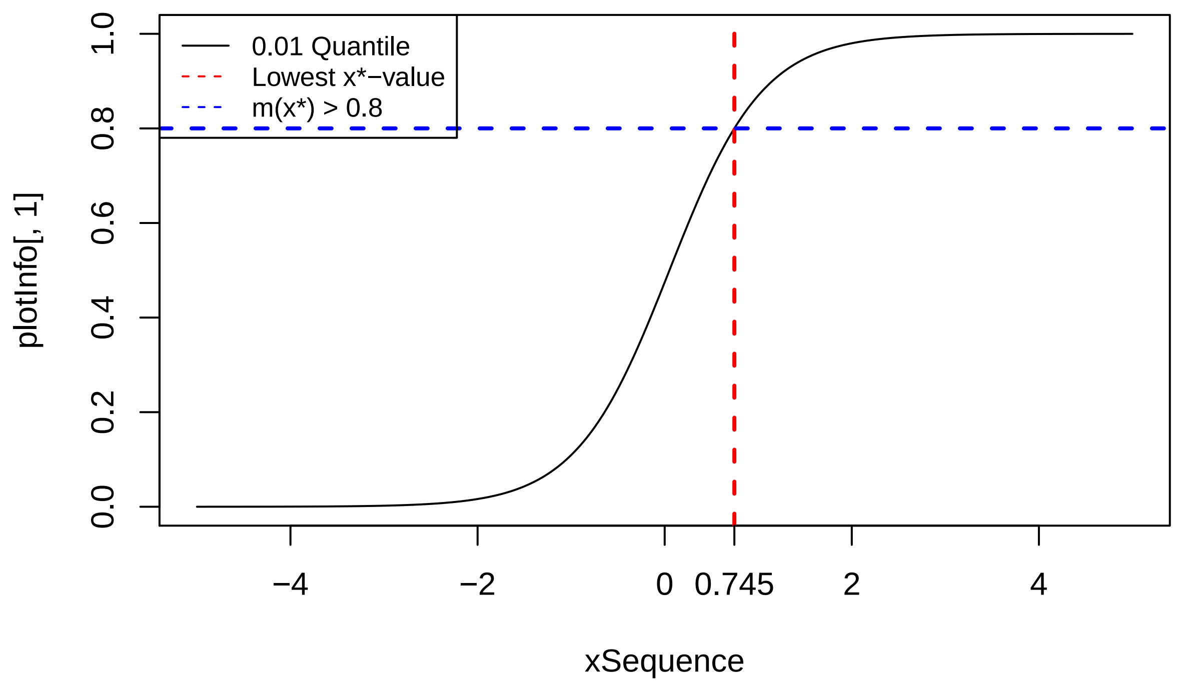


Problem 3C
Find x-values with 99% certainty that $m(x*) > 0.8$.

```r
# First value larger than 0.8. 1% quantile
lowestValue <- plotInfo[, 1][plotInfo[, 1] > 0.8][1]

# First x-value of m-value with 99% probability of being larger than 0.8
firstOccurence <- xSequence[Position(function(x) x > 0.8, plotInfo[, 1])]

firstOccurence.rounded <- round(firstOccurence, 3)

print(c("Smallest x*-value" = firstOccurence))
```

```
## Smallest x*-value
##         0.7448152
```

```r
plot(xSequence, plotInfo[, 1], type="l", col="black")
abline(v = firstOccurence, col="red", lty = 2, lwd = 2)
abline(h = lowestValue, col="blue", lty = 2, lwd = 2)
axis(1, at=firstOccurence.rounded, labels=firstOccurence.rounded)
legend("topleft", legend=c("0.01 Quantile", "Lowest x*-value", "m(x*) > 0.8"),
       col=c("black", "red", "blue"), lty = c(1, 2, 2) , cex=0.8)
```



As the plot from problem 3b illustrates, $m(x*)$ is a strictly increasing function. This means that if $a < b$, $m(a) < m(b)$. The 1% quantile provides x-values with 99% certainty that $m(x*) > 0.8$. This means that there is a lower $x*$, and all $x*$-values above satisfy the condition $m(x*) > 0.8$.

This means that for any $x*$-values larger than or equal to 0.745, the condition $m(x*) > 0.8$ with a certainty of 99% is fulfilled.

Problem 4: Gibbs sampler for simple linear regression model Problem 4A
Find the distribution of $\alpha|\beta, \tau, \boldsymbol{y}$ and $\beta|\alpha, \tau, \boldsymbol{y}$

Given that the joint posterior of $\boldsymbol{\theta} = (\alpha, \beta, \tau)$ has the log-density kernel:

$$\log g(\boldsymbol{\theta}) = \frac{n}{2}\log(\tau) - \frac{\tau}{2}\sum_{i=1}^{n}(y_i - \alpha - \beta x_i)^2 - \frac{\alpha^2 + \beta^2}{200} - \tau$$

Finding the conditional α and β distributions is/are done by collecting all parts of the equation not containing the parameter in focus into a constant and (in this case) writing the expressions in the form $a + bx + cx^2 \Rightarrow x \sim N\left(-\frac{b}{2c}, -\frac{1}{2c}\right)$.

By first expanding the squared sum and factorizing each part of the expression, the expression becomes:

For α:

$$\alpha|\beta,\tau,\boldsymbol{y} = \alpha\sum_{i=1}^{n}(y_i\tau) + \alpha^2\sum_{i=1}^{n}(-\frac{\tau}{2}) + \alpha\sum_{i=1}^{n}(-\beta x_i\tau) + \alpha^2(-\frac{1}{200}) + C$$

$$\alpha|\beta,\tau,\boldsymbol{y} = C + \alpha(\tau\sum_{i=1}^{n}(y_i - \beta x_i)) + \alpha^2(-\frac{\tau n}{2} - \frac{1}{200})$$

$$\alpha|\beta,\tau,\boldsymbol{y} \sim N\left(\frac{\tau\sum_{i=1}^{n}(y_i - \beta x_i)}{\tau n + 1/100}, \frac{1}{\tau n + 1/100}\right)$$

For β:

$$\beta|\alpha,\tau,\boldsymbol{y} = \beta\sum_{i=1}^{n}(y_i x_i\tau) + \beta\sum_{i=1}^{n}(-\alpha x_i\tau) + \beta^2\sum_{i=1}^{n}(-\frac{\tau}{2}x_i^2) + \beta^2(-\frac{1}{200}) + C$$

$$\beta|\alpha,\tau,\boldsymbol{y} = C + \beta(\tau\sum_{i=1}^{n}(y_i x_i - \alpha x_i)) + \beta^2(-\frac{\tau}{2}\sum_{i=1}^{n}(x_i^2) - \frac{1}{200})$$

$$\beta|\alpha,\tau,\boldsymbol{y} \sim N\left(\frac{\tau\sum_{i=1}^{n}(y_i x_i - \alpha x_i)}{\tau\sum_{i=1}^{n}(x_i^2) + \frac{1}{100}}, \frac{1}{\tau\sum_{i=1}^{n}(x_i^2) + \frac{1}{100}}\right)$$

Also, $\tau|\alpha,\beta,\boldsymbol{y}$ is given as:

$$\tau|\alpha,\beta,\boldsymbol{y} \sim Gamma\left(\frac{n}{2} + 1, \frac{1}{\frac{1}{2}\sum_{i=1}^{n}(y_i - \alpha - \beta x_i)^2 + 1}\right)$$

Problem 4B + 4C
Gibbs sampler with 3 blocks targeting $p(\boldsymbol{\theta}|\boldsymbol{y})$.

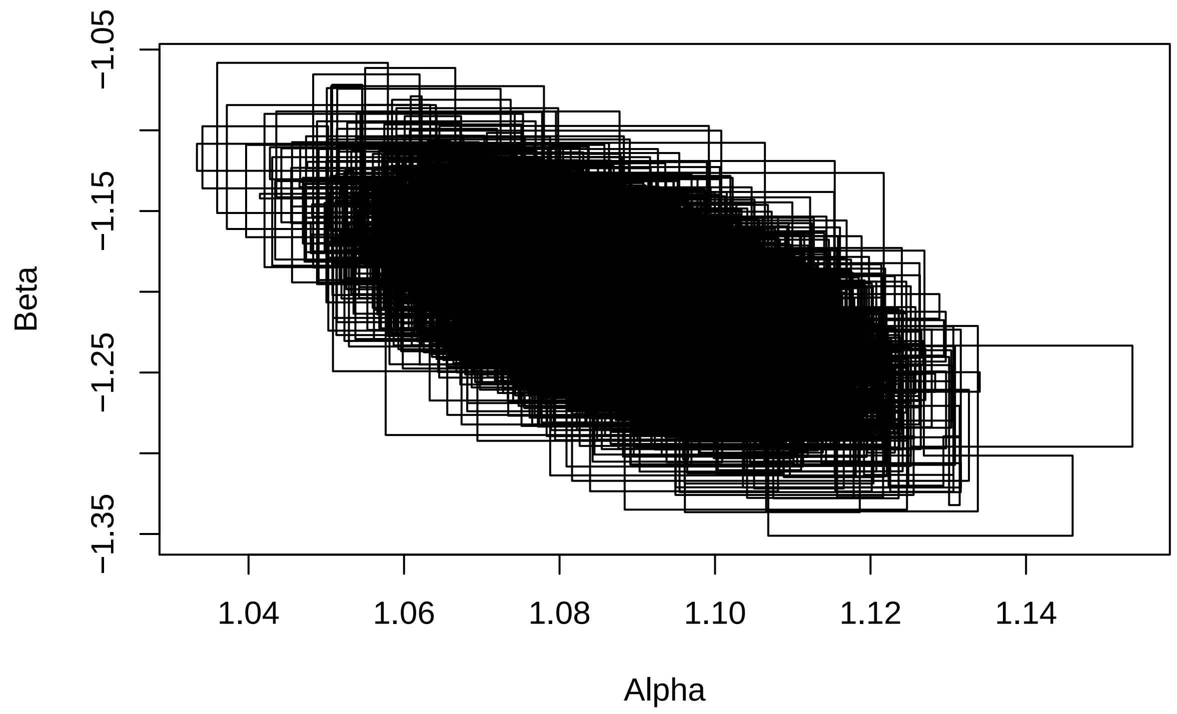
```
df <- data.frame(read.table(file="linear_regression_data.txt"))
x <- df$x
y <- df$y


#
# Gibbsampler for bivariate normal target
#

gibbs3b <- function() {
  n <- 10000 # 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 (alpha)
      theta[1] <- rnorm(1, theta[3]*sum(y-theta[2]*x) / (theta[3]*length(y) + 1/100), 1 / (theta[3]*leng
      # store state
      res[k, ] <- theta
      k <- k+1

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

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

Problem 4C
Use the Gibbs sampler with 3 blocks.

```
gibbs3 <- gibbs3b()


gibbs3.burned <- gibbs3[501:length(gibbs3[ ,1]), ]
colnames(gibbs3.burned) <-  c("Alpha", "Beta", "Tau")

round(colMeans(gibbs3.burned), 4)
```

```
##   Alpha    Beta     Tau
##  1.0888 -1.2071  1.0983
```

```
plot(gibbs3.burned[,1], gibbs3.burned[,2], type="l", xlab = "Alpha", ylab = "Beta")
```



```r
# plot(gibbs3.burned[,1], gibbs3.burned[,3], type="l", xlab = "Alpha", ylab = "Tau")
# plot(gibbs3.burned[,2], gibbs3.burned[,3], type="l", xlab = "Beta", ylab = "Tau")

ess <- ESS(gibbs3.burned)
c("ESS Alpha" = ess[[1]], "ESS Beta" = ess[[2]], "ESS Tau" = ess[[3]])
```

```
## ESS Alpha  ESS Beta   ESS Tau
##  5172.043  5354.978 11436.639
```

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

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

After burn-in, the effective sample size of all α, β and τ are above the required size of 1000.

It can be seen from the plots that α and β has a correlation of -0.5084097, while the other plots show a correlation close to zero. Thus, α and β are dependent on each other, and the Gibbs sampler would be improved by combining α and β, and reducing the number of blocks from three to two.

The last output can be used to compare the posterior means with a classical linear regression, which gives approximately the same α- and β-values.

Problem 4D
Gibbs sampler with 2 blocks.

From the text, we have that $(\alpha, \beta)|\tau, \boldsymbol{y} \sim N(-\boldsymbol{c}^{-1}\boldsymbol{b}, -\boldsymbol{c}^{-1})$, where:

$$\boldsymbol{c} = \begin{pmatrix} -n\tau & 0.01 \\ -\tau\sum_{i=1}^{n} x_i & -\tau\sum_{i=1}^{n} x_i^2 - 0.01 \end{pmatrix}$$

$$\boldsymbol{b} = \begin{pmatrix} \tau\sum_{i=1}^{n} y_i \\ \tau\sum_{i=1}^{n} y_i x_i \end{pmatrix}$$

Using $(\alpha, \beta)|\tau, \boldsymbol{y}$ and $\tau|\alpha, \beta, \boldsymbol{y}$ gives the following Gibbs sampler with two blocks:

```r
gibbs2b <- function() {
  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

  # Calculating some constants for performance:
  sumx <- sum(x)
  # Sum of x^2
  sumx2 <- sum(x^2)
  # Sum of y
  sumy <- sum(y)
  # Sum of xy
  sumyx <- sum(y*x)
  # Length of both y (or x)
  dataLength <- length(y)


  for(i in 1:n){
    cVal = matrix(c(-dataLength*theta[3]-0.01, -theta[3]*sumx, -theta[3]*sumx, -theta[3]*sumx2-0.01), 2
    bVal = c(theta[3]*sumy, theta[3]*sumyx)

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

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


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

```
gibbs2 <- gibbs2b()

gibbs2.burned <- gibbs2[501:length(gibbs2[ ,1]), ]
colnames(gibbs2.burned) <-  c("Alpha", "Beta", "Tau")

round(colMeans(gibbs2.burned), 4)
```

```
##   Alpha    Beta     Tau
##  1.0894 -1.2079  1.0743
```

```
# plot(gibbs2.burned[,1], gibbs2.burned[,2], type="l", xlab = "Alpha", ylab = "Beta")
# plot(gibbs2.burned[,1], gibbs2.burned[,3], type="l", xlab = "Alpha", ylab = "Tau")
# plot(gibbs2.burned[,2], gibbs2.burned[,3], type="l", xlab = "Beta", ylab = "Tau")

ess <- ESS(gibbs2.burned)
c("ESS Alpha" = ess[[1]], "ESS Beta" = ess[[2]], "ESS Tau" = ess[[3]])
```

```
## ESS Alpha  ESS Beta   ESS Tau
## 10037.420 10183.503  8992.425
```

The new effective sample sizes clearly shows that fewer simulations are needed to get the same ESS as in problem 3C, meaning that the performance of the Gibbs sampling procedure has been improved.

As in problem 3C, the mean of α, β and τ are still the same, meaning that the sampling still works as intended.

Problem 4E
:D

Problem 5A

Problem 6
Problem 6A
Obtain MCMC samples targeting p(theta|y)

Problem 6C