Mandatory 3

Håvard Godal, 245609

11th November, 2020

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

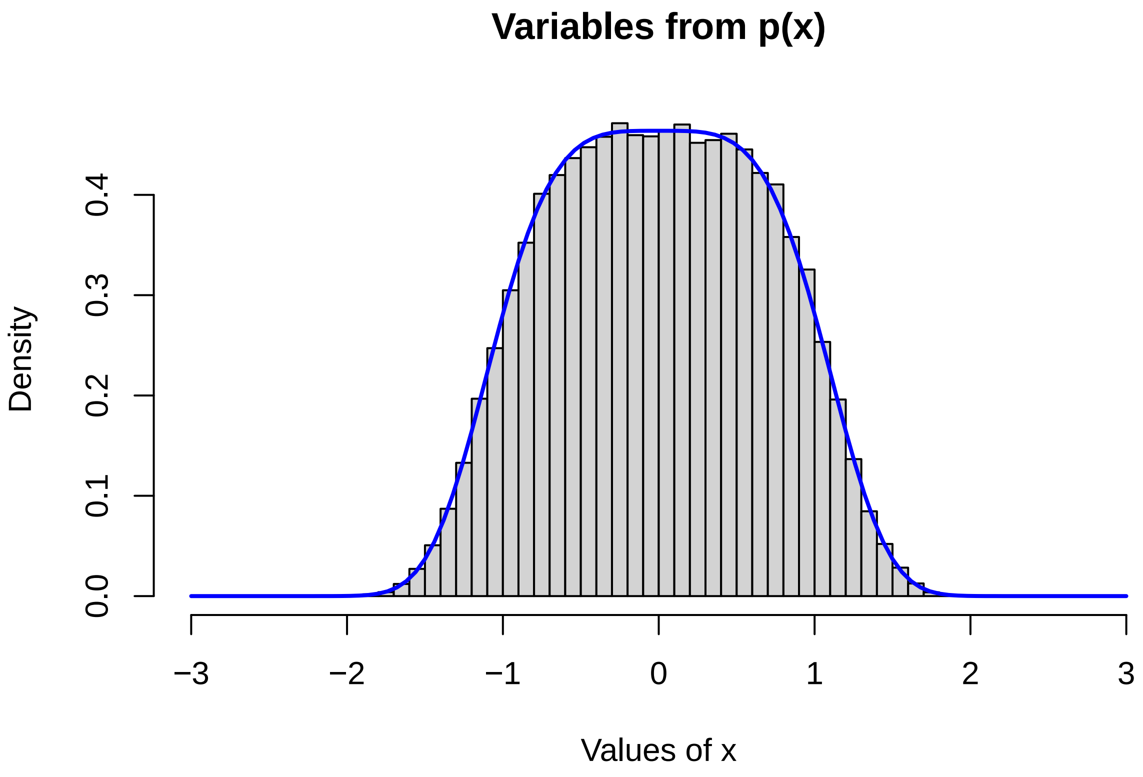
Note: For most of the simulations regarding random walk, burn-in has been used, although it is not necessary for all types of random walk. As long as the effective sample size is sufficient, burn-in is OK to use, which is why it has been used in most of the simulations.

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 $p(x)$:

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

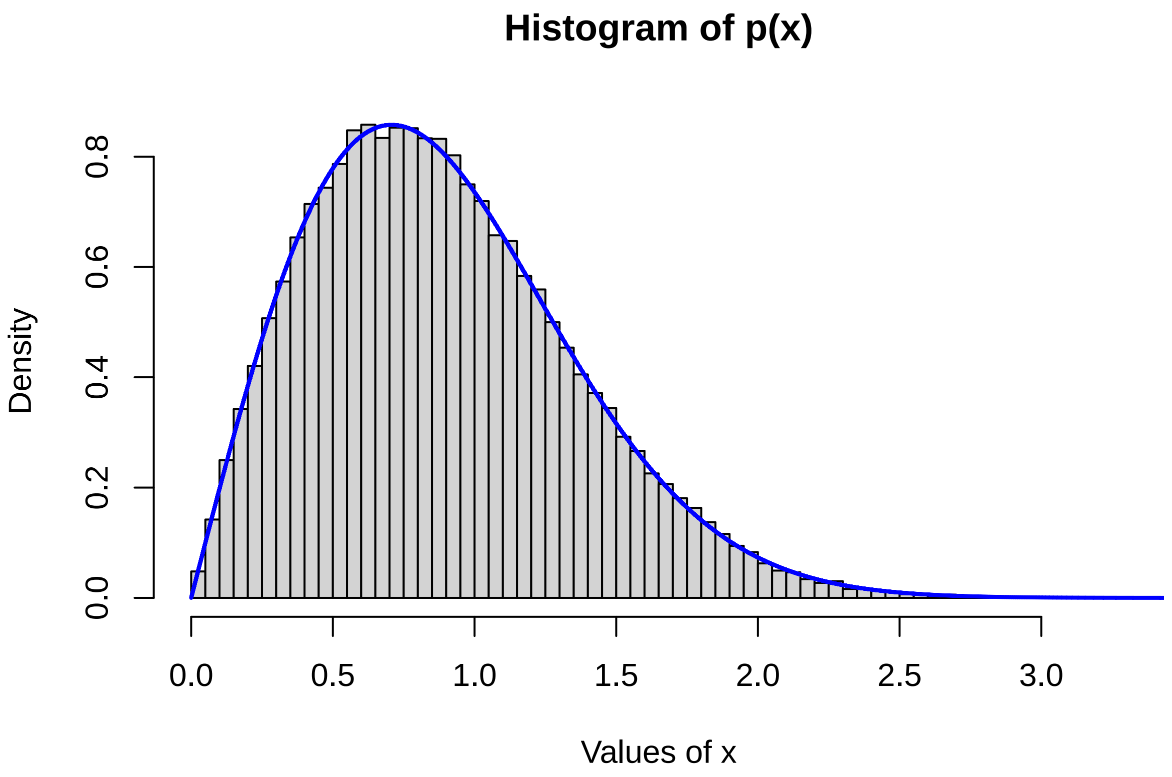
Because $0 \leq F(x) \leq 1$ and $F(x)$ is uniformly distributed, we set $F(X) = U$ and solve for x:

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

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.

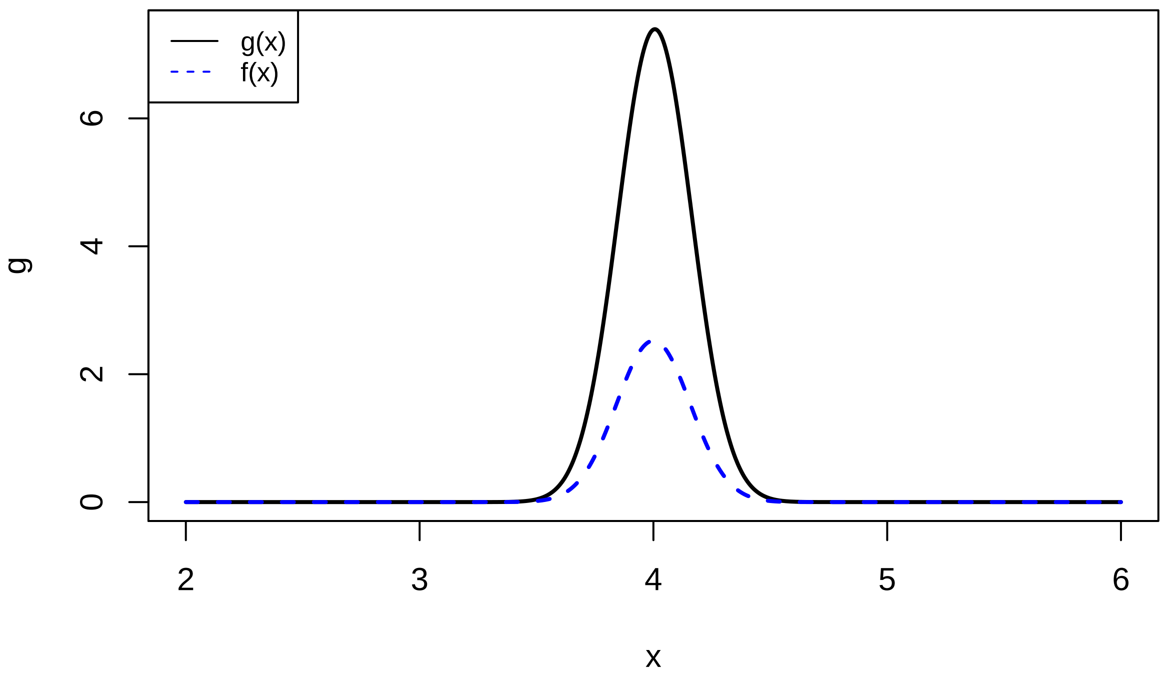
Further, we have that:

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

```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         2.928062
```

```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.

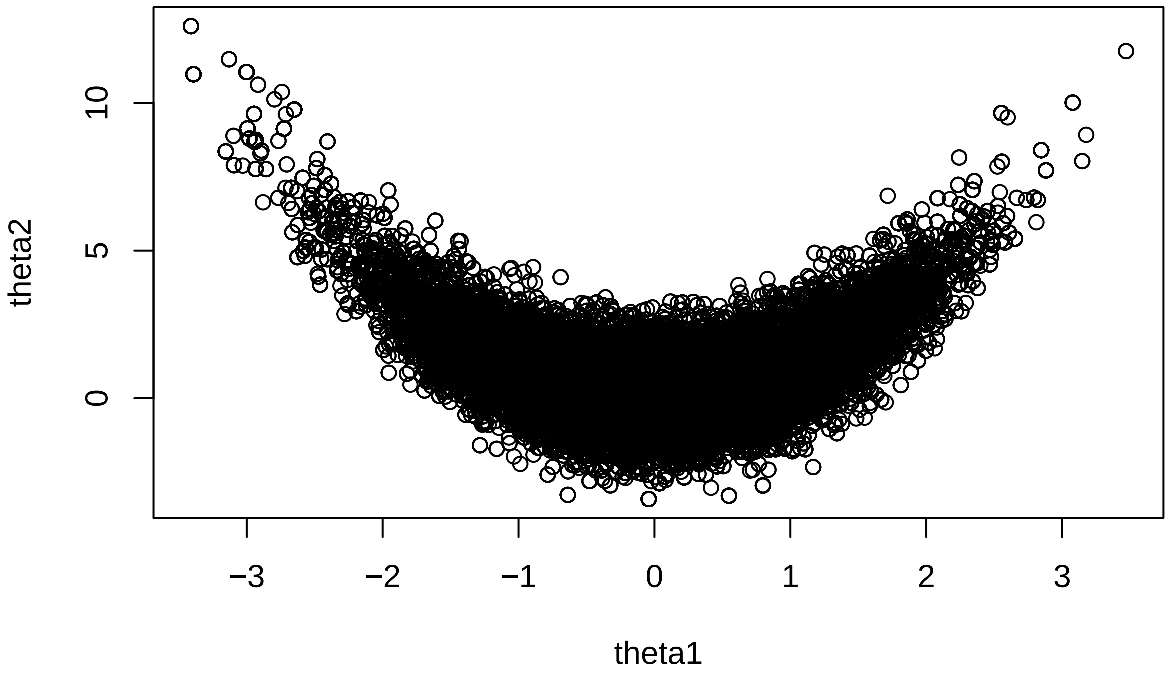
As the problem states, a MCMC algorithm has to be used as direct sampling from the distribution is too complicated to derive.

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
```

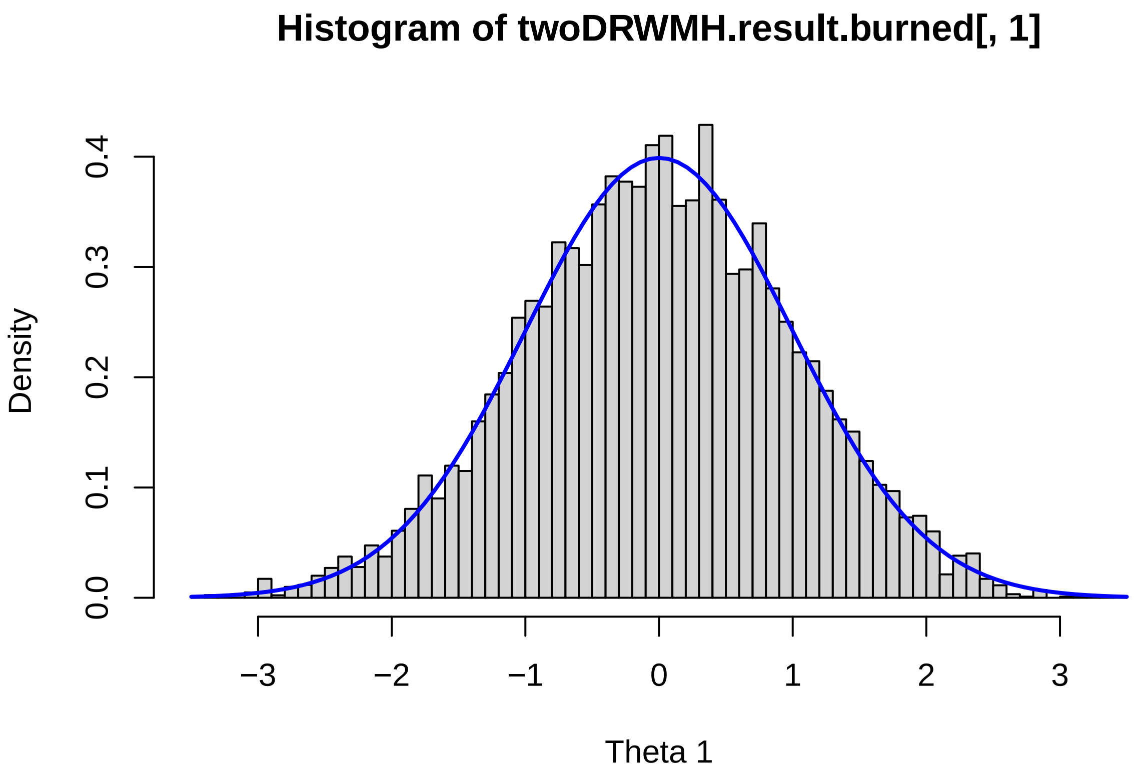
```r
twoDRWMH.result.burned <- twoDRWMH.result[501:length(twoDRWMH.result[ ,1]), ]
colnames(twoDRWMH.result.burned) <- c("theta1","theta2")

plot(twoDRWMH.result.burned)
```



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

```
## ESS Theta 1 ESS Theta 2
##    3886.550    1864.423
```

```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, ]
    }
  }
```

```r
    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)

# 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
##  -0.103   1.997
```

```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
##  8953.685  8096.837
```

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.

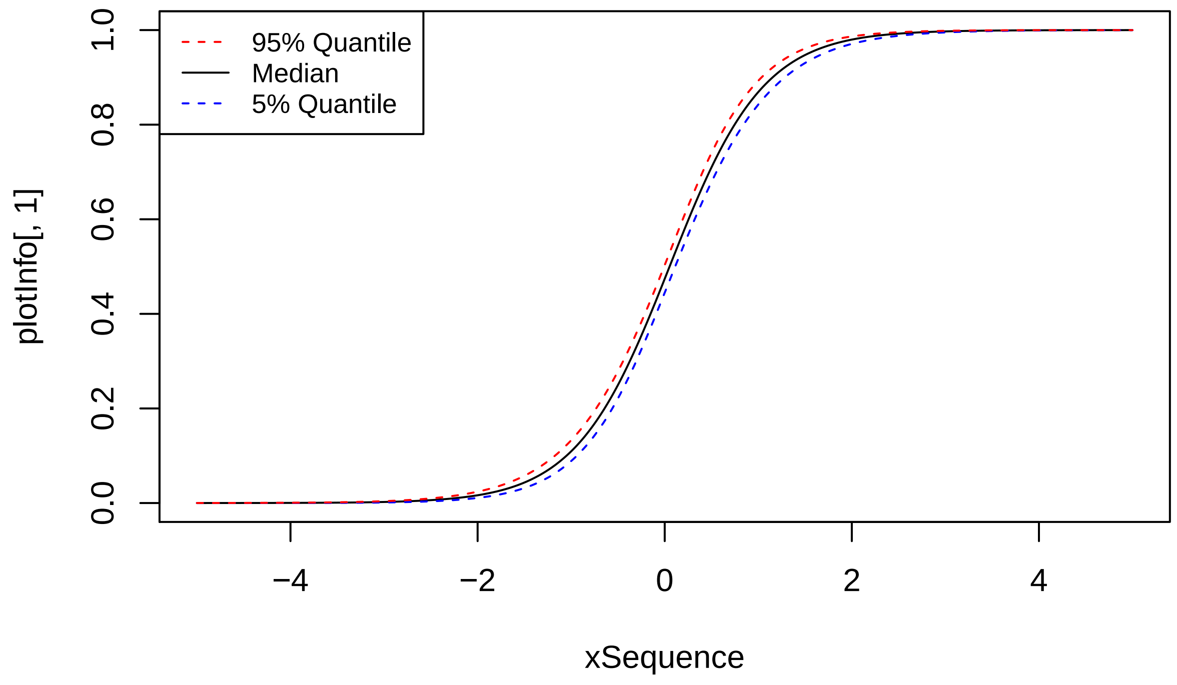
Say something about the acceptance rate and the tuning.

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.

```r
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)
```

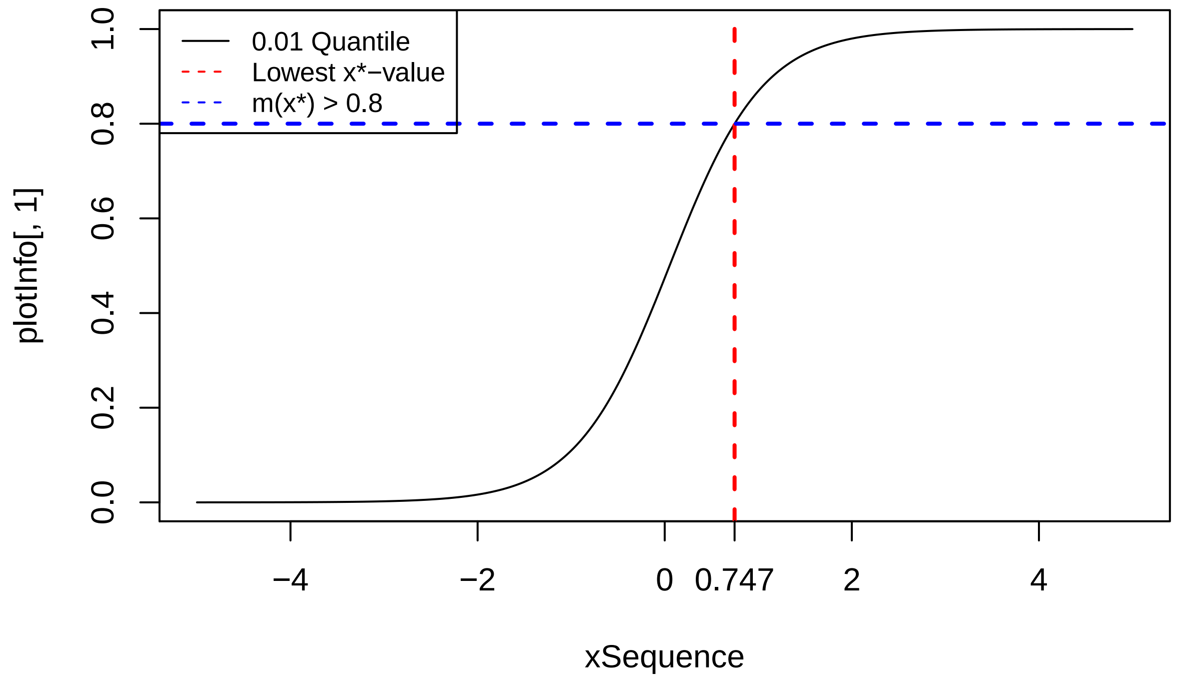


Problem 3C
Find x-values with 99% certainty that $m(x*) > 0.8$.

```
# 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
##         0.7469207
```

```
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 0.747, 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 $p(\boldsymbol{\theta}|\boldsymbol{y})$.

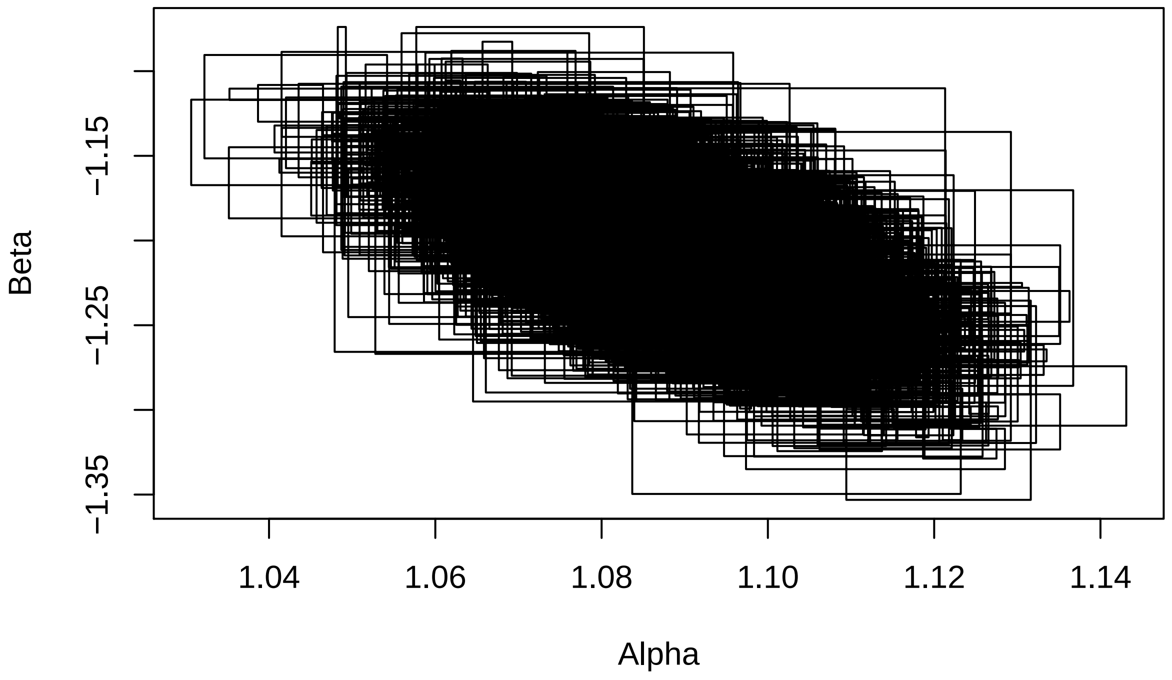
```r
df <- data.frame(read.table(file="linear_regression_data.txt"))
x <- df$x
y <- df$y


#
# 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]*length(y) + 1/100))

      # 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]*sum(x^2) + 1/100))
      # 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) + 1))
      # store state
      res[k, ] <- theta
      k <- k+1
  }
  res
}
```

Problem 4C
Use the Gibbs sampler with 3 blocks.

```r
gibbs3 <- gibbs3b()


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

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

```
##    Alpha     Beta      Tau
##  1.0884 -1.2063  1.0941
```

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



```
# 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
##  5029.130  5076.662  8927.613
```

```
# 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 -0.5109307, 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, 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) + 1))
    #store state
    res[k,] <- theta
    k <- k+1
  }
  res
}

gibbs2 <- gibbs2b()

gibbs2.burned <- gibbs2[501:length(gibbs2[ ,1]), ]
```

```r
colnames(gibbs2.burned) <-  c("Alpha", "Beta", "Tau")

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

```
##   Alpha    Beta     Tau
##  1.0879 -1.2050  1.0749
```

```r
# 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")

# hist(gibbs2.burned[,1])
# hist(gibbs2.burned[,2])

# plot(gibbs2.burned, type = "l")

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

```
## ESS Alpha  ESS Beta   ESS Tau
## 11761.683 11954.437  9436.255
```

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.

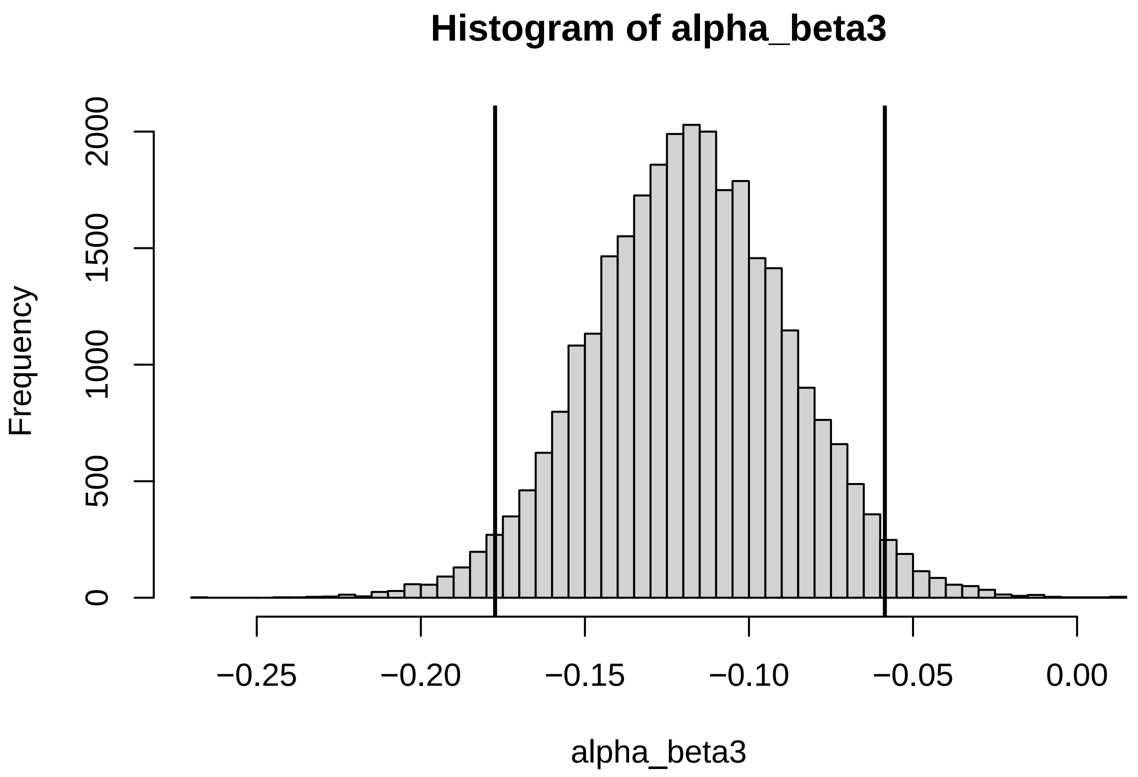
Having fewer blocks and improving independence between the blocks gives better samples, which is clearly showed in the solution of the problem.

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

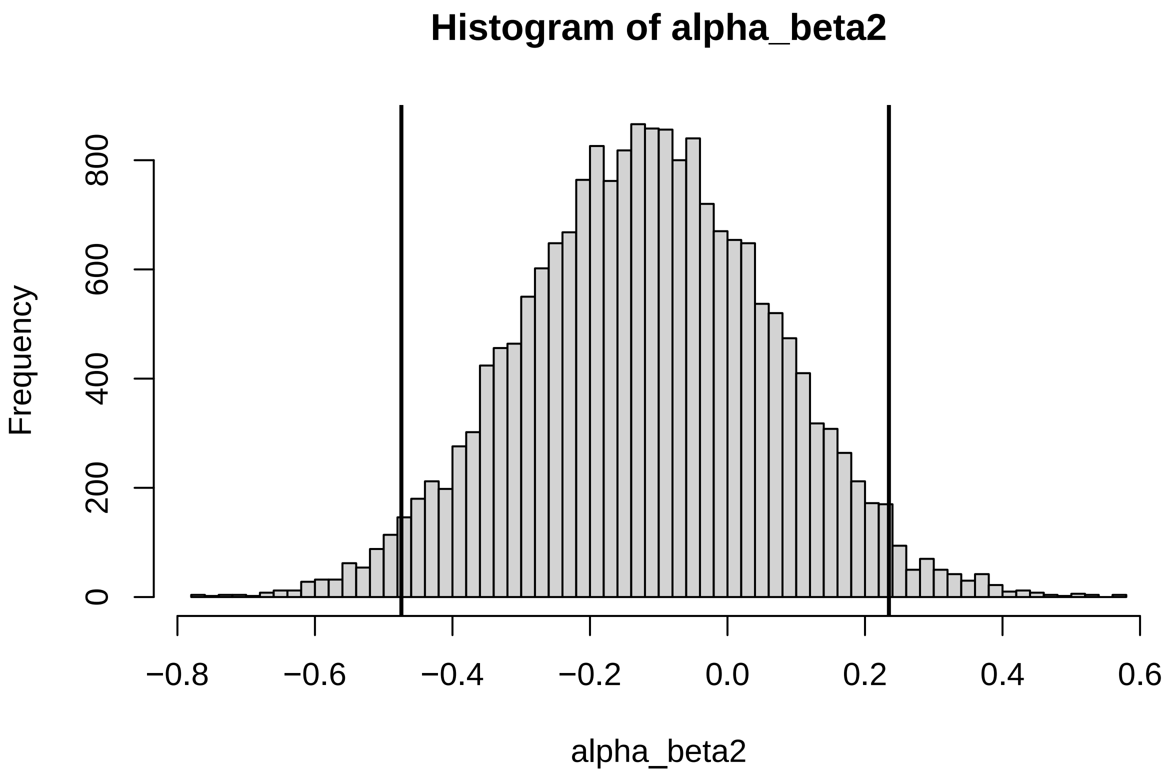
Problem 4E
Verify that $\alpha + \beta = 0$

This can be seen as a hypothesis test, using the 15- and 75-percentile on both Gibbs samplers.

```r
alpha_beta3 <- gibbs3.burned[, 1] + gibbs3.burned[, 2]
hist(alpha_beta3, breaks = 60)
abline(v = quantile(alpha_beta3, 0.975), lwd="2")
abline(v = quantile(alpha_beta3, 1-0.975), lwd="2")
```



```r
alpha_beta2 <- gibbs2.burned[, 1] + gibbs2.burned[, 2]
hist(alpha_beta2, breaks = 60)
abline(v = quantile(alpha_beta2, 0.975), lwd="2")
abline(v = quantile(alpha_beta2, 1-0.975), lwd="2")
```



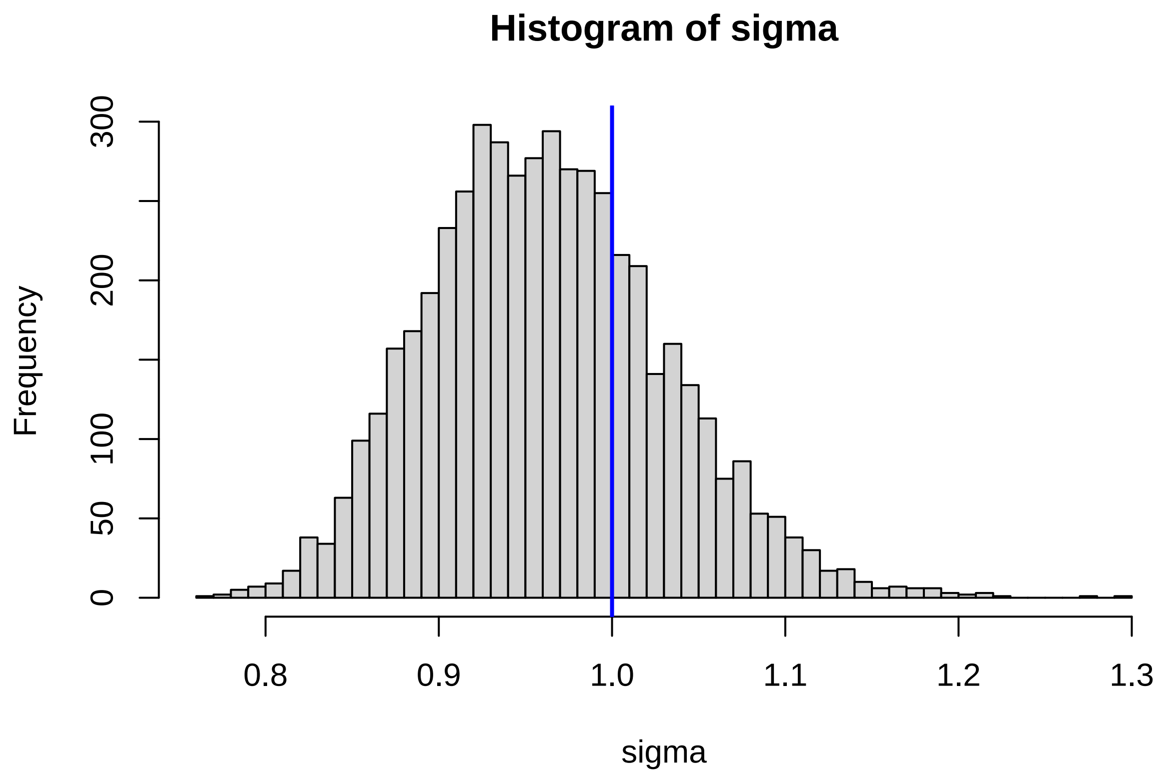
From the Gibbs sampler with three blocks shows that $\alpha + \beta$ equals zero falls outside the quantile limits, and the hypothesis can be rejected.

On the other hand, the Gibbs sampler with two blocks shows that $\alpha + \beta$ equals zero falls inside the quantile limits. This means that $\alpha + \beta = 0$ might hold true. Thus, the hypothesis can not be rejected in this case.

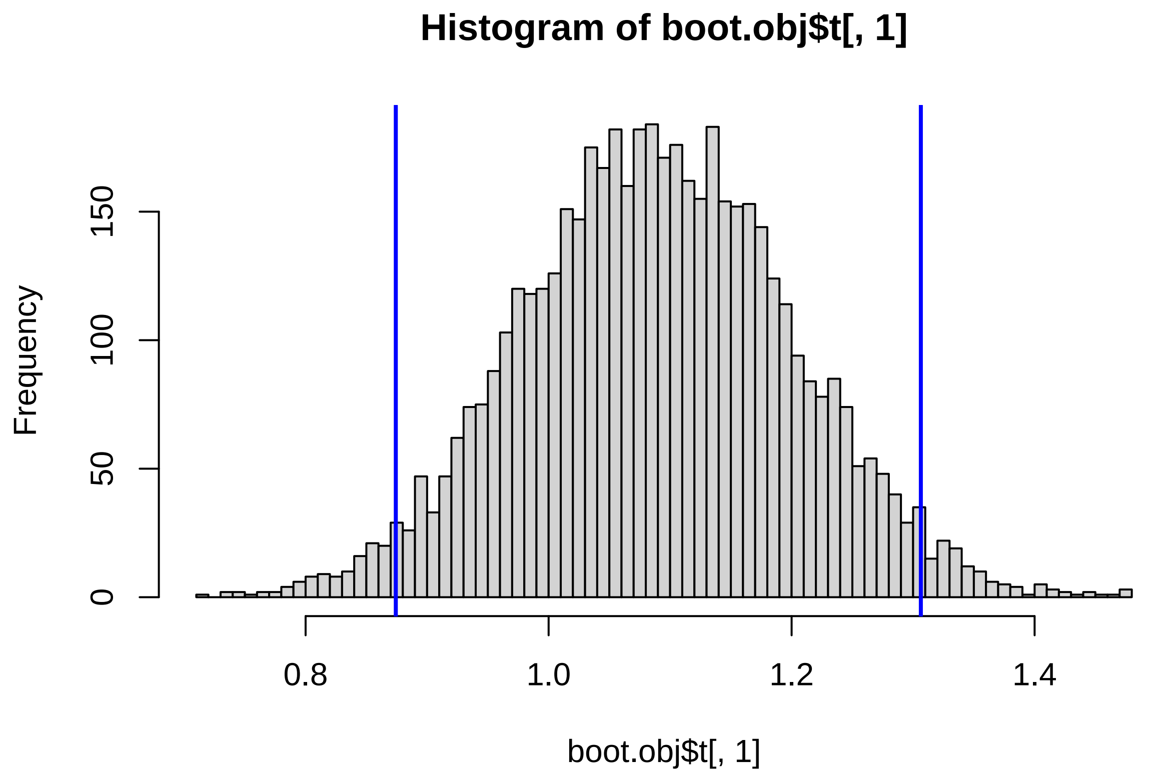
Problem 5: Bootstrapping
Problem 5A
Check if the claims given in the problem holds true.

Claim 1: $\sigma = \frac{1}{\sqrt{\tau}}$ is strictly smaller than 1.0.
Claim 2: α is equal to 0.
Claim 3: $\alpha + \beta$ is equal to 0.

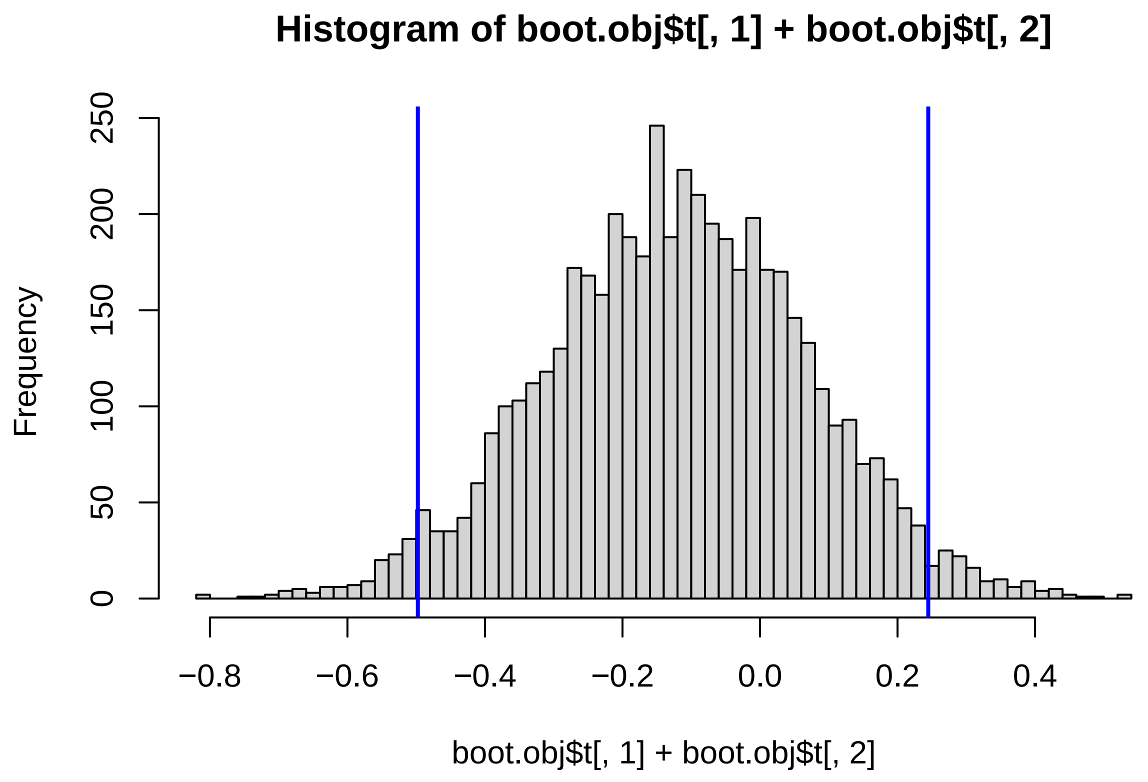
```r
df <- data.frame(read.table(file = "linear_regression_data.txt"))
x <- df$x
y <- df$y

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)

# 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
hist(sigma, breaks = 60)
abline(v = 1, lwd = 2, col = "blue")
```



```r
# Claim 2
hist(boot.obj$t[ ,1], breaks = 60)
abline(v = quantile(boot.obj$t[ ,1], 0.975), lwd = 2, col = "blue")
abline(v = quantile(boot.obj$t[ ,1], 1 - 0.975), lwd = 2, col = "blue")
```



```r
# Claim 3
hist(boot.obj$t[ ,1] + boot.obj$t[ ,2], breaks = 60)
abline(v = quantile(boot.obj$t[ ,1] + boot.obj$t[ ,2], 0.975), lwd = 2, col = "blue")
abline(v = quantile(boot.obj$t[ ,1] + boot.obj$t[ ,2], 1-0.975), lwd = 2, col = "blue")
```



The first histogram clearly shows that σ is not strictly smaller than 1.0. Because of this, claim 1 is rejected.

Claim 2 is also rejected as the second histogram shows that $\alpha = 0$ falls outside the 95% confidence interval.

As $\alpha + \beta = 0$ falls inside the 95% confidence interval, claim 3 can not be rejected.

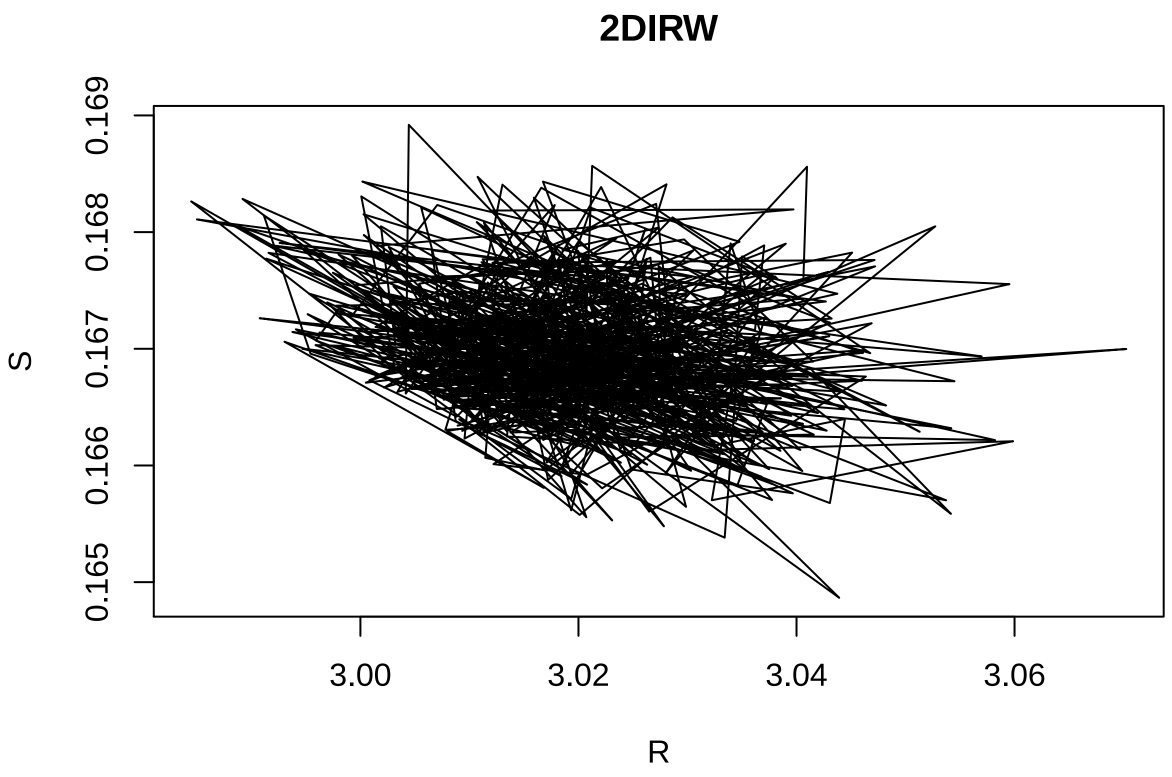
Problem 6: Bayesian inference for the SEIR model
Problem 6A
Obtain MCMC samples targeting $p(\theta|\boldsymbol{y})$

Using a bivariate RWMH does not yield effective sample sizes of R and S of 200 or more. To fix this, an independent RWMH is implemented. Because the independent random walk returns independent values of R and S, the effective sample size will increase.

```r
source("seir_mcmc_funs.R")

Nsim <- 1000

# Independent sampler
ind.2DRW <- function(l_target_prob, theta, sigma, tuning){
  # allocate memory
  res <- matrix(0, Nsim, 2)
  res[1, ] <- theta
   # old importance weight
  wt.old <- l_target_prob(theta) - dmvnorm(theta, theta, sigma, log=TRUE)
  Nacc <- 0
  for (i in 2:Nsim){
    # proposal (note, independent of past)
    # Sample theta from multivariate normal with scaled sigma
    thetaStar <- rmvnorm(1, theta, tuning*sigma)
    # new importance weight
    wt.star <- l_target_prob(thetaStar) - dmvnorm(thetaStar, theta, sigma, log=TRUE)
    # accept probability
    alpha <- exp(min(0.0, wt.star - wt.old))
    # accept/reject
    if(runif(1)<alpha && is.finite(alpha)){
      res[i,] <- thetaStar
      wt.old <- wt.star
      Nacc <- Nacc+1
    } else {
      res[i,] <- res[(i-1),]
    }
  }
  print(paste0("accept rate : ",Nacc/Nsim))
  return(res)
}


ind.result <- ind.2DRW(seir.lp,
             theta=c(3.019, 0.167),
             sigma=matrix(c(0.0003,0,0,0.0000005),2,2),
             tuning=0.8)
```

```
## [1] "accept rate : 0.601"
```

```r
plot(ind.result[,1], ind.result[,2], type="l", main="2DIRW", xlab="R", ylab="S")
```



```r
ESS(ind.result[,1])
```

```
## [1] 395.2775
```

```r
ESS(ind.result[,2])
```

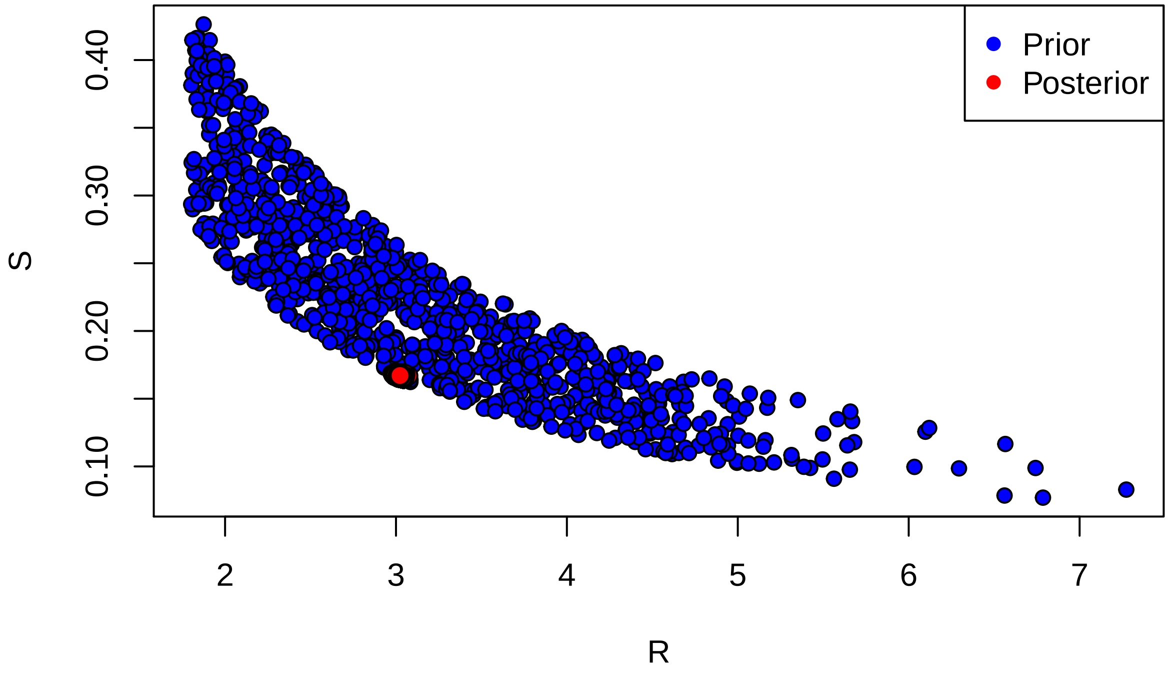
```
## [1] 177.4678
```

The effective sample size for R and S are both over the required 200.
# Omformulere? The plot shows that the sampler explores the distribution quite well, avoiding the chain to stick, while also avoiding exploring the parameter space too slowly.

Problem 6B Compare the posterior distribution and the prior distribution.

```r
Nsim <- 1000

# inverse transform method (from mandatory 2)
randNum <- function(l, mu, sigma) {
    qnorm(
      pnorm(l, mu, sigma) + runif(Nsim, 0, 1)*(1 - pnorm(l, mu, sigma)),
          mu, sigma
      )
}

# generate random R- and S-variables (from mandatory 2)
prior <- function() {
  rNum <- randNum(l = 1.8, mu = 2.2, sigma = 1.5)
  sNum <- runif(Nsim, 0.5/rNum, 0.8/rNum)
  cbind(rNum, sNum)
}
```

```r
prior.values <- prior()

plot(prior.values[ ,1], prior.values[ ,2],
     xlab = "R", ylab = "S",
     pch = 21, bg = "blue")
points(ind.result[ ,1], ind.result[ ,2], pch = 21, bg = "red", cex = 1.3)
legend("topright", legend = c("Prior", "Posterior"),
       pch = c(16, 16), col = c("blue", "red"))
```

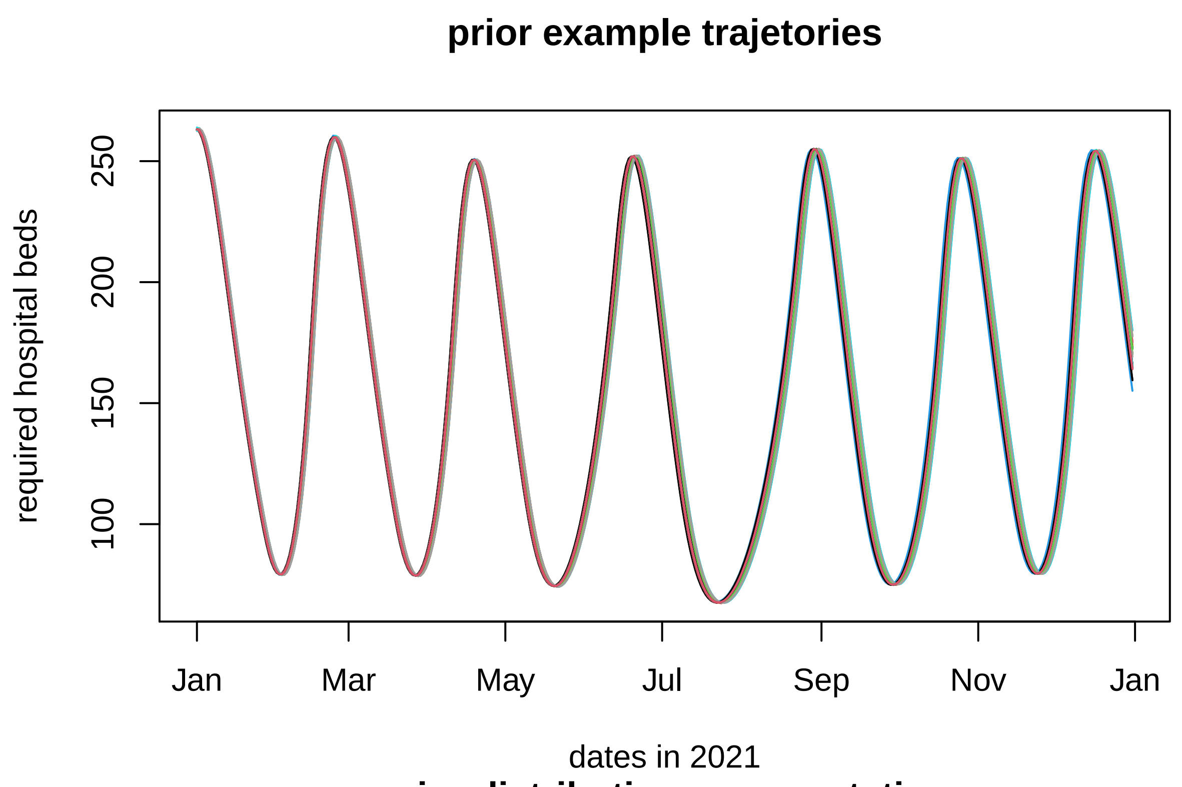


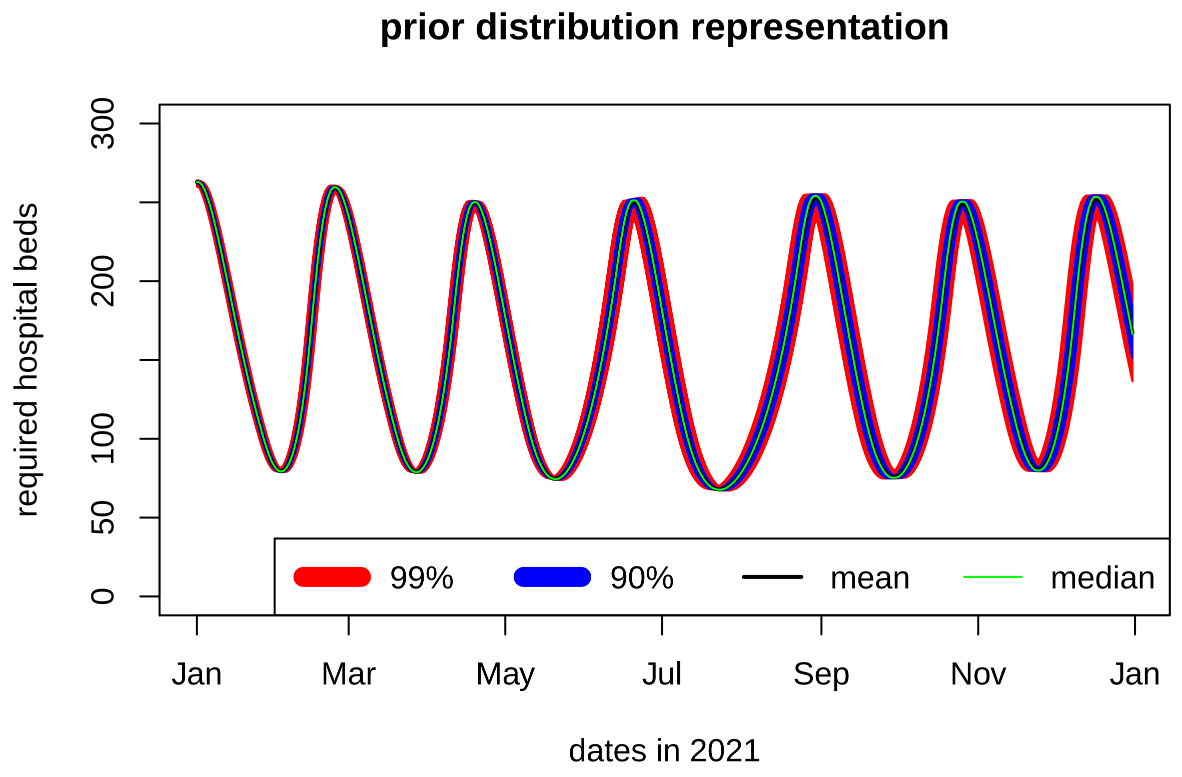
The hospital data provides a lot of information of the R- and S-values, which can clearly be seen by comparing the posterior distribution with the prior distribution through the plot.

Problem 6C
Re-do Problem 4F) of mandatory assignment 2

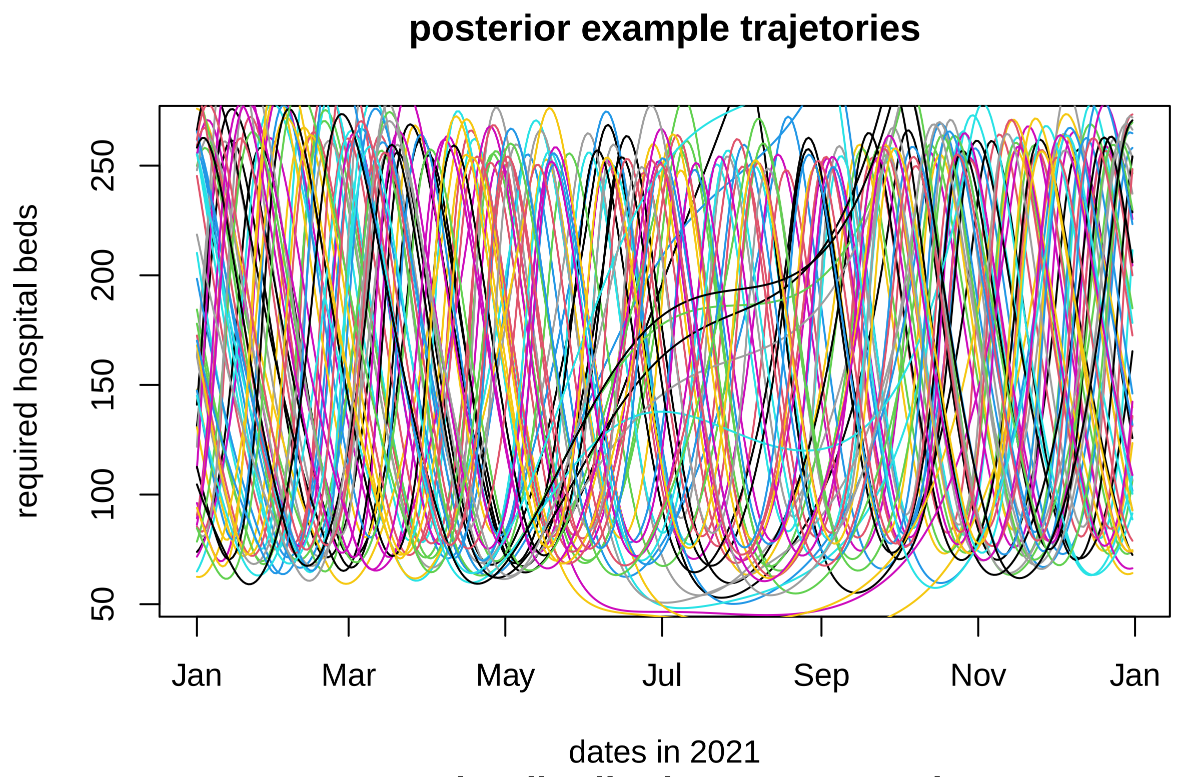
```r
Nsim <- 500
source("seir_model.R")

uncertainty <- function(dataInfo) {
  # run simulations
  M <- numeric(Nsim)
  H <- matrix(0.0,Nsim,365) # prepare for next point
  for(i in 1:Nsim){
    #R[i] -> dataInfo[i, 1], S[i] -> dataInfo[i, 2]
    ff <- seir_sim(R0max = dataInfo[i, 1],soc_dist_Rfac = dataInfo[i, 2])
    t1 <- which(ff$dates=="2021-01-01")
    t2 <- which(ff$dates=="2021-12-31")
    H[i,] <- rowSums(1000000*ff$ode[t1:t2,c("HH","HC","CC")])
    M[i] <- max(H[i,])
  }

  # make a data frame to facilitate plotting with dates
  df <- data.frame(t(H),dates=ff$dates[t1:t2])

  plot1Name <- ifelse(plotNum == 1, "prior example trajetories", "posterior example trajetories")

  # in the upper plot, simply show some of the trajectories:
  plot(df$dates,df$X1,type="l",
       xlab="dates in 2021",
       ylab="required hospital beds",
       main = plot1Name)
  for(i in 2:50) lines(df$dates,df[,i],col=i)

  # compute daily statistics for plotting
  daily.stats <- matrix(0.0,365,6)
  for(t in 1:365){
    daily.stats[t,1] <- mean(H[,t])
    daily.stats[t,2] <- median(H[,t])
    daily.stats[t,3:6] <- quantile(H[,t],probs=c(0.005,0.1,0.9,0.995))
  }

  df2 <- data.frame(daily.stats,ff$dates[t1:t2])
  colnames(df2) <- c("mean","median","q05","q10","q90","q995","dates")

  plot2Name <- ifelse(plotNum == 1, "prior distribution representation", "posterior distribution repres

  plot(df2$dates,df2$mean,ylim=c(0,300),col=0,
       xlab="dates in 2021",
       ylab="required hospital beds",
       main=plot2Name)
  polygon(c(df2$dates,rev(df2$dates)),c(df2$q05,rev(df2$q995)),
          col="red",density=100)
  polygon(c(df2$dates,rev(df2$dates)),c(df2$q10,rev(df2$q90)),
          col="blue",density=100)
  lines(df2$dates,df2$mean,lwd=2)
  lines(df2$dates,df2$median,col="green")
  legend("bottomright",lty=c(1,1,1,1),lwd=c(10,10,2,1),
         legend=c("99%","90%","mean","median"),
         col=c("red","blue","black","green"),
         horiz = TRUE)
}
```

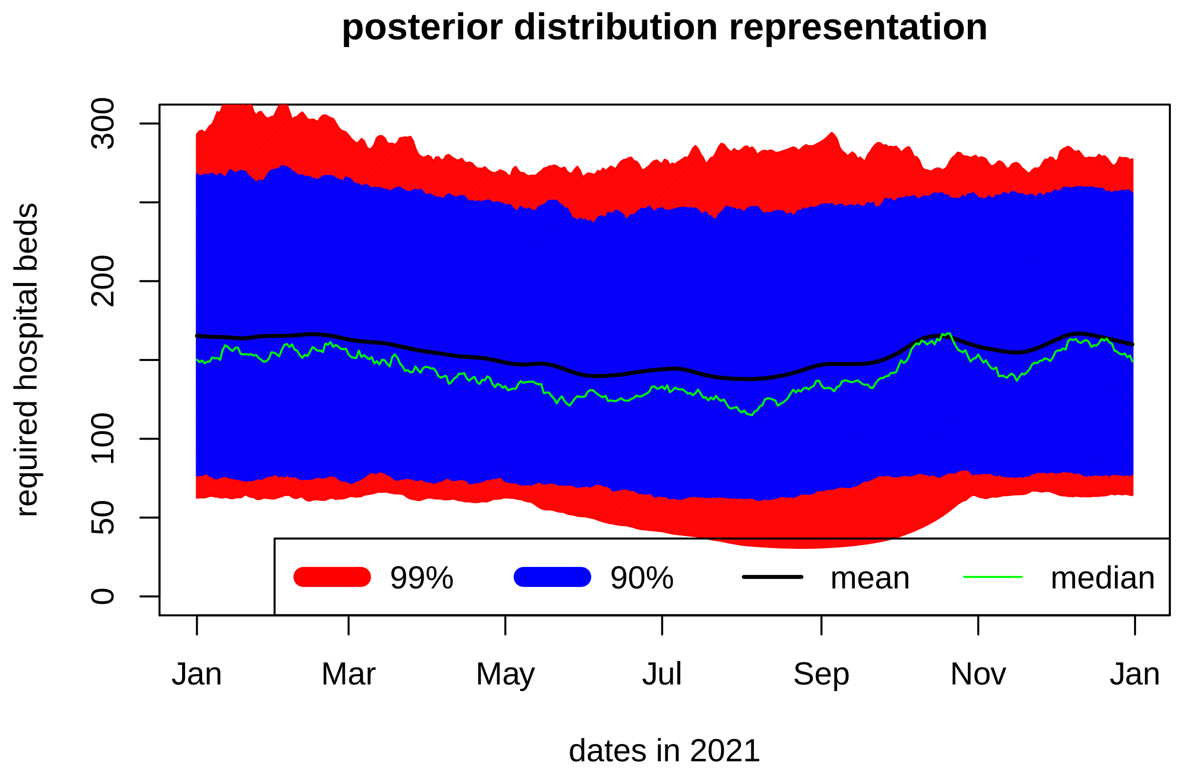
```
plotNum <- 1
uncertainty(ind.result)
```





```
plotNum <- 2
uncertainty(prior.values)
```



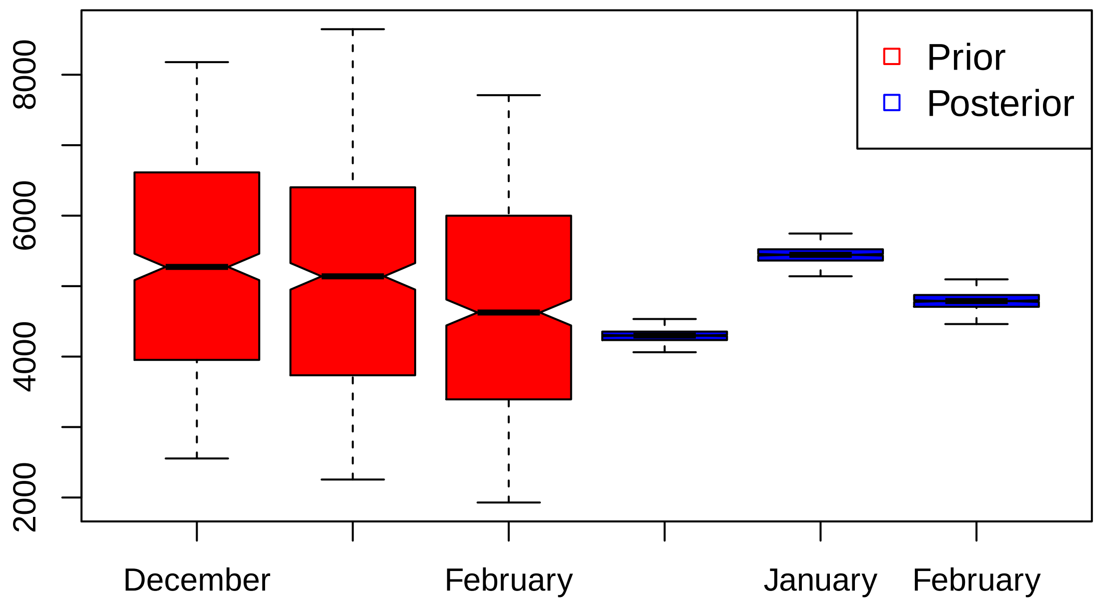


The plots correspond to the data received from problem 6B. When observing the data, the variance of the R- and S-values decreases, which means that the posterior distribution will represent the required hospital beds more accurate than the prior distribution by itself.
The 90% confidence interval of the prior distribution ranges between 80 and 260 beds almost all the time, which from the prior example trajectories appears to be correct as the R- and S-values have a wide variety of

trajectories to “choose from”.
The 90% confidence interval of the posterior distribution has a width of around 10 required hospital beds, and follows the mean closely. Because all of the trajectories of the posterior distribution are close to equal, the result makes sense.

Problem 6D
Distribution of the total number of ”hospital-bed-days” for December 2020, January 2021 and February 2021.

```r
Nsim <- 500

hospBeds <- function(dataInfo) {

  december <- matrix(0.0,Nsim,31)
  january <- matrix(0.0,Nsim,31)
  february <- matrix(0.0,Nsim,28)

  for(i in 1:Nsim){
    f <- seir_sim(R0max = dataInfo[i, 1], soc_dist_Rfac = dataInfo[i, 2])

    days <- which(grepl("2020-12", f$dates))
    december[i, ] <- rowSums(1000000*f$ode[days, c("HH","HC","CC")])

    days <- which(grepl("2021-01", f$dates))
    january[i, ] <- rowSums(1000000*f$ode[days, c("HH","HC","CC")])

    days <- which(grepl("2021-02", f$dates))
    february[i, ] <- rowSums(1000000*f$ode[days, c("HH","HC","CC")])
  }

  cbind("December" = rowSums(december), "January" = rowSums(january), "February" = rowSums(february))
}


priorList <- hospBeds(prior.values)
posteriorList <- hospBeds(ind.result)

boxplot(cbind(priorList, posteriorList),
        col = c("red", "red", "red", "blue", "blue", "blue"),
        notch = TRUE, varwidth = TRUE)
legend("topright", legend = c("Prior", "Posterior"), col = c("red", "blue"), pch = 22, cex = 1.2)
```



The plot explains that without the given dataset of required beds from December, January and February, the uncertainty of how many beds are required increases. In this case, the number of required hospital beds ranges between 7000 and 4000 per month.

By taking the observed data from 21st March, 2020 to 18th July, 2020 into consideration, the uncertainty drastically decreases. This means that if the data is representative for the given situation, the estimation of required hospital beds is much more exact. With a high confidence, it can be seen that the required number of hospital beds for December, January and February is 4300, 5500 and 4800, respectively, with a standard deviation of around 90 hospital beds.