Exercise set 8

Bjørn Christian Weinbach

30th October, 2020

Clear R environment

```
rm(list = ls())
```

Exercise 8.4

Note: MLE for $\hat{\lambda} = n/\sum_{i=1}^{n} X_i$, where X denote time between failure

Refer to the air-conditioning data set **aircondit** provided in the boot library. The 12 observations are the times in hours between failures of air-conditioning equipment

$$3 \quad 5 \quad 7 \quad 18 \quad 43 \quad 85 \quad 91 \quad 98 \quad 100 \quad 130 \quad 230 \quad 487$$

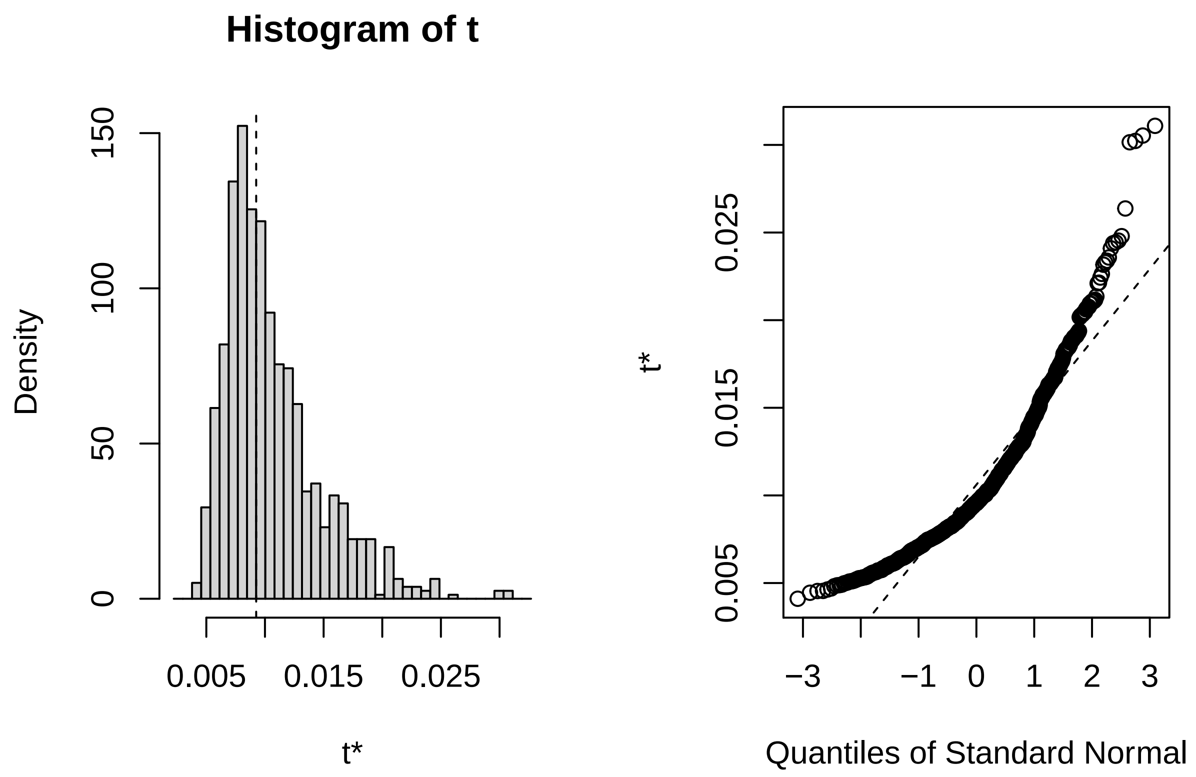
Assume that the times between failures follow an exponential model with rate λ. Obtain the MLE of the hazard rate λ and use bootstrap to estimate the bias and standard error of the estimate.

in R:

```
library(boot)

# MLE for hazard rate of exponential distributed data
mle <- function(data, i) {
  return(length(data[i])/sum(data[i]))
}

# bootstrapping with 1000 replications
results <- boot(data=aircondit$hours, statistic = mle, R=1000)

# view results
results
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = aircondit$hours, statistic = mle, R = 1000)
##
##
## Bootstrap Statistics :
##       original       bias    std. error
## t1* 0.00925212 0.001358761 0.004105667
```

```
plot(results)
```



```r
# get 95% confidence interval
boot.ci(results, type=c("bca", "norm", "perc"))
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = results, type = c("bca", "norm", "perc"))
##
## Intervals :
## Level      Normal              Percentile            BCa
## 95%   (-0.0002,  0.0159 )   ( 0.0053,  0.0209 )   ( 0.0047,  0.0182 )
## Calculations and Intervals on Original Scale
## Some BCa intervals may be unstable
```

Exercise 8.5

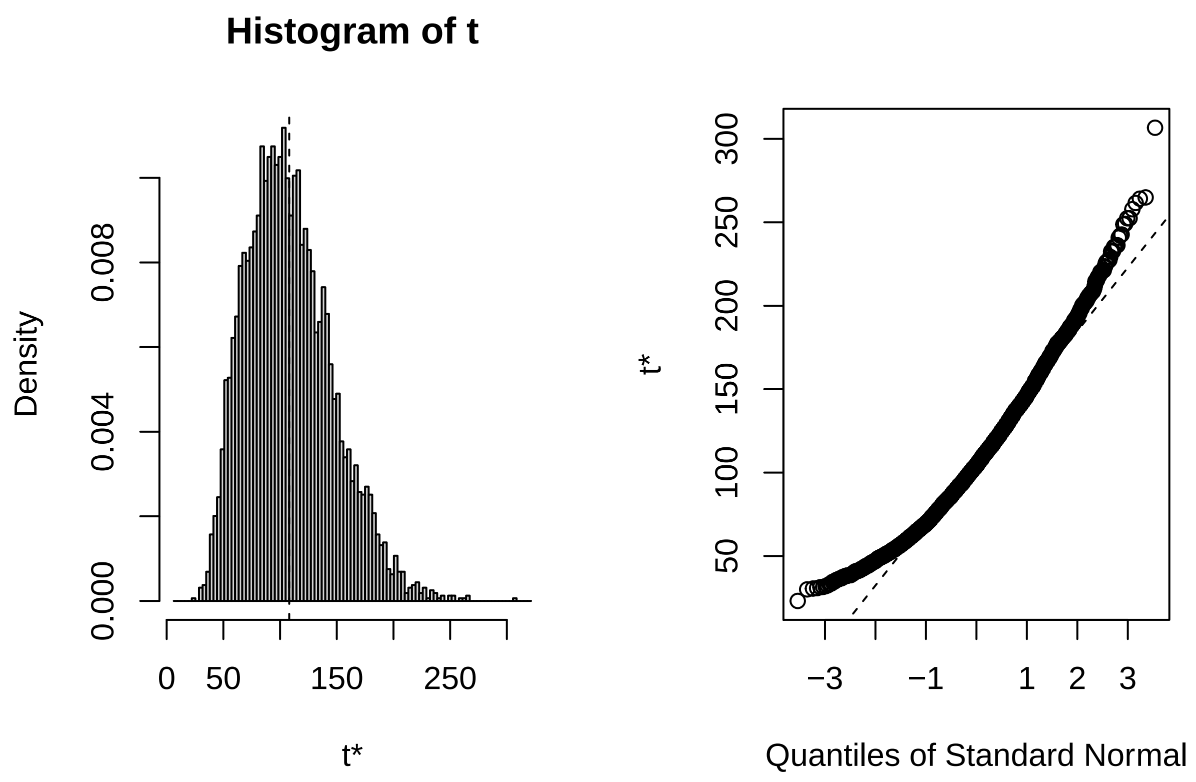
Refer to exercise 8.4. Compute 95% confidence interval for the mean time between failures by the standard normal, basic, percentile and BCa methods.

```r
# MLE for hazard rate of exponential distributed data
meantimeest <- function(data, i) {
  rate <- length(data[i])/sum(data[i])
  return(1/rate)
}
```

```r
# bootstrapping with 1000 replications
results <- boot(data=aircondit$hours, statistic = meantimeest, R=5000)

# view results
results
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = aircondit$hours, statistic = meantimeest, R = 5000)
##
##
## Bootstrap Statistics :
##     original      bias    std. error
## t1* 108.0833 0.4705167     38.09992
```

```r
plot(results)
```



```r
# get 95% confidence interval
boot.ci(results, type=c("norm", "basic", "perc", "bca"))
```

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

```
## boot.ci(boot.out = results, type = c("norm", "basic", "perc",
##     "bca"))
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 32.9, 182.3 )   ( 24.3, 168.1 )
##
## Level     Percentile            BCa
## 95%   ( 48.1, 191.8 )   ( 56.9, 227.3 )
## Calculations and Intervals on Original Scale
```

Exercise 11.3

Use MALA sampler to generate random variables from a standard Cauchyy distribution. Discard the first 1000 of the chain, and compare the deciles of the generated observations with the deciles of the standard Cauchy distribution. Recall that a Cauchy(θ, η) has density

$$f(x) = \frac{1}{\theta\pi(1 + [(x-\eta)/\theta]^2)}, \quad -\infty < x < \infty, \quad \theta > 0. \tag{1}$$

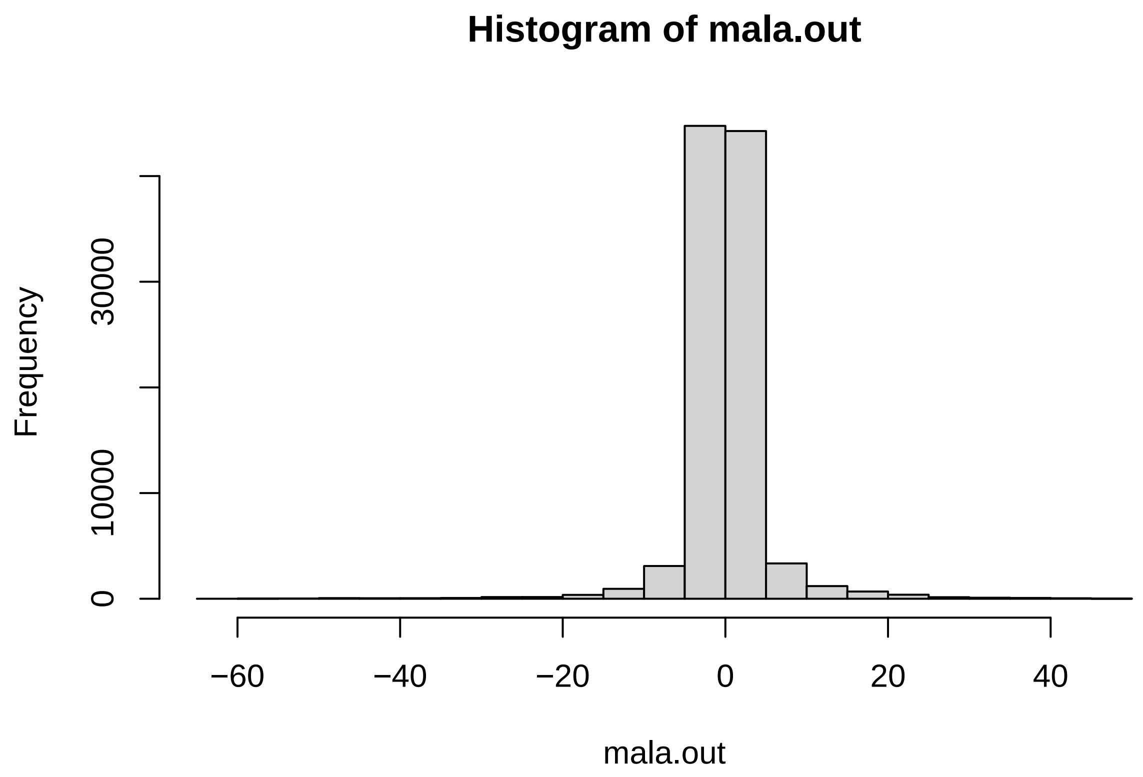
In R:

```
# 1-d MALA method from the lectures

oneDMALA <- function(lp_grad,
                     theta1 = 0.0,
                     Delta = 0.5,
                     n.iter=10000){

  # storage
  out <- numeric(n.iter)
  out[1] <- theta1
  old.lpg <- lp_grad(theta1)

  Nacc <- 0

  for( i in 2:n.iter){
    # proposal
    thetaStar <- out[i-1] + 0.5*Delta*old.lpg$grad +
      sqrt(Delta)*rnorm(1)
    # new eval at proposal
    new.lpg <- lp_grad(thetaStar)

    # proposal density forward
    lqf <- dnorm(thetaStar,mean = out[i-1] + 0.5*Delta*old.lpg$grad,
                 sd = sqrt(Delta),log=TRUE)
    # proposal density backward
    lqb <- dnorm(out[i-1],mean = thetaStar + 0.5*Delta*new.lpg$grad,
                 sd = sqrt(Delta),log=TRUE)

    # accept probability
    alpha <- exp(min(0.0,
                     new.lpg$lp + lqb # numerator
                     -old.lpg$lp - lqf # denominator
```

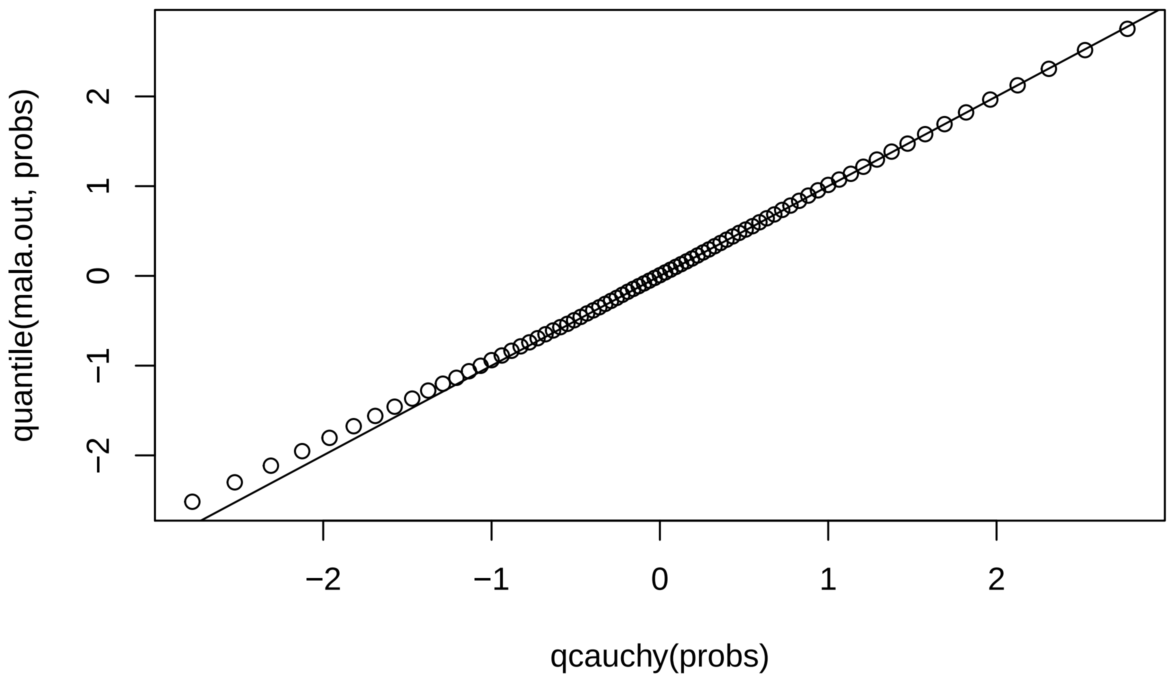
```r
    ))

    if(runif(1)<alpha && is.finite(alpha)){
      out[i] <- thetaStar
      old.lpg <- new.lpg
      Nacc <- Nacc + 1
    } else {
      out[i] <- out[i-1]
    }

  } # main iteration loop
  print(paste0("MALA done, accept rate :",Nacc/(n.iter-1)))
  return(out)
} # function


# target function
lpg <- function(x){
  return(list(lp=-log(1.0+x^2), #no need for normalizing constant
              grad=-2.0*x/(1.0+x^2)))
}

# try algorithm (change initial condition and sigma for illustration)
mala.out <- oneDMALA(lp_grad=lpg,theta1=rcauchy(1),Delta=4.0,n.iter = 100000)
```

```
## [1] "MALA done, accept rate :0.703847038470385"
```

```r
# Histogram
hist(mala.out)
```



```r
# plot mcmc quantiles against theoretical deciles
par(mfrow=c(1,1))
probs <- 0.01*(11:89)
plot(qcauchy(probs),quantile(mala.out,probs))
abline(0,1)
```



```r
# looks quite good, note, tails very difficult to get properly represented
```

This implementation uses standard normal proposals. We can see from the plot that the random walk stops suddenly on values far from the center of the distribution and skews the samples. Let's perform some diagnostics:

```r
###--- Problem 11.3 (Rizzo)
library(coda) # mcmc utilites
# now run a few more chains to be able to run proper tests

mcmc1 <- mcmc(mala.out)
varnames(mcmc1)<- 'x'
mala.out <- oneDMALA(lp_grad=lpg,theta1=rcauchy(1),Delta=4.0,n.iter = 100000)
```

```
## [1] "MALA done, accept rate :0.702617026170262"
```

```r
mcmc2 <- mcmc(mala.out)
varnames(mcmc2)<- 'x'
mala.out <- oneDMALA(lp_grad=lpg,theta1=rcauchy(1),Delta=4.0,n.iter = 100000)
```

```
## [1] "MALA done, accept rate :0.706957069570696"
```

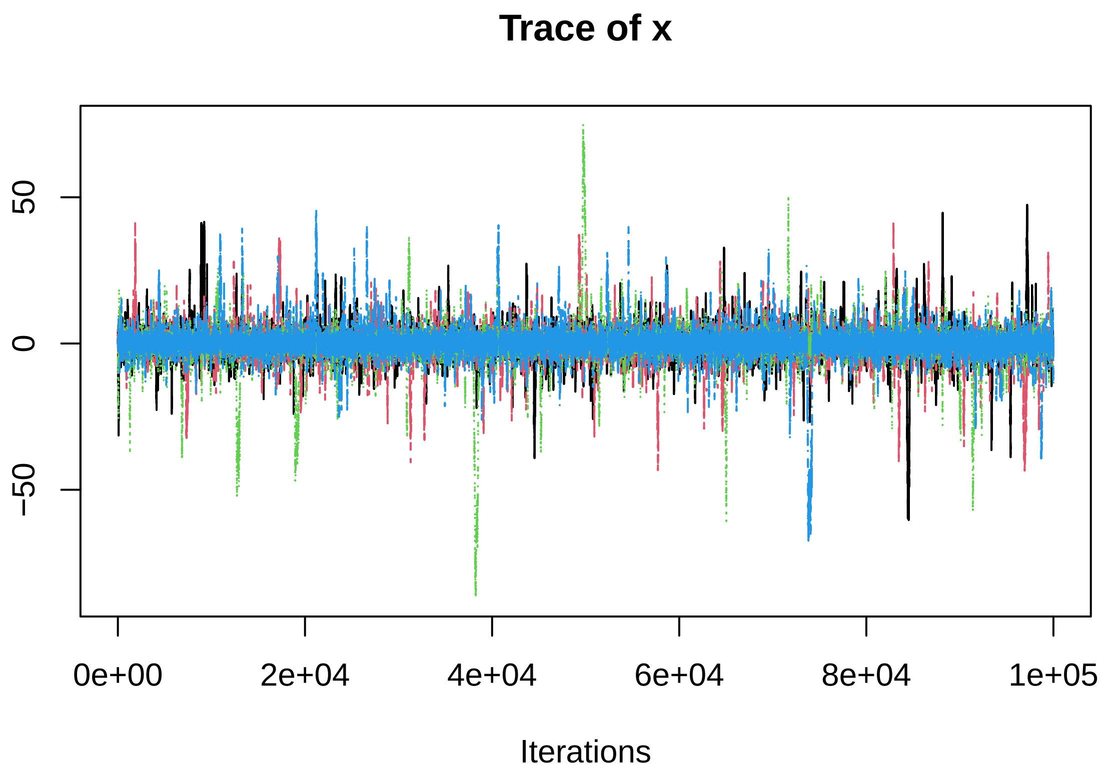
```r
mcmc3 <- mcmc(mala.out)
varnames(mcmc3)<- 'x'
mala.out <- oneDMALA(lp_grad=lpg,theta1=rcauchy(1),Delta=4.0,n.iter = 100000)
```

```
## [1] "MALA done, accept rate :0.70919709197092"
```

```r
mcmc4 <- mcmc(mala.out)
varnames(mcmc4)<- 'x'

ml <- mcmc.list(mcmc1,mcmc2,mcmc3,mcmc4)

effectiveSize(ml)
```

```
##       x
## 2770.58
```

```r
# not super-good due to the heavy tails
traceplot(ml)
```



```r
# look at trace plot
geweke.diag(ml)
```

```
## [[1]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##     x
## 1.586
##
##
## [[2]]
##
```

```
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##     x
## 0.655
##
##
## [[3]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##      x
## -1.005
##
##
## [[4]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##    x
## 1.21
```

```
# borderline OK
gelman.diag(ml)
```

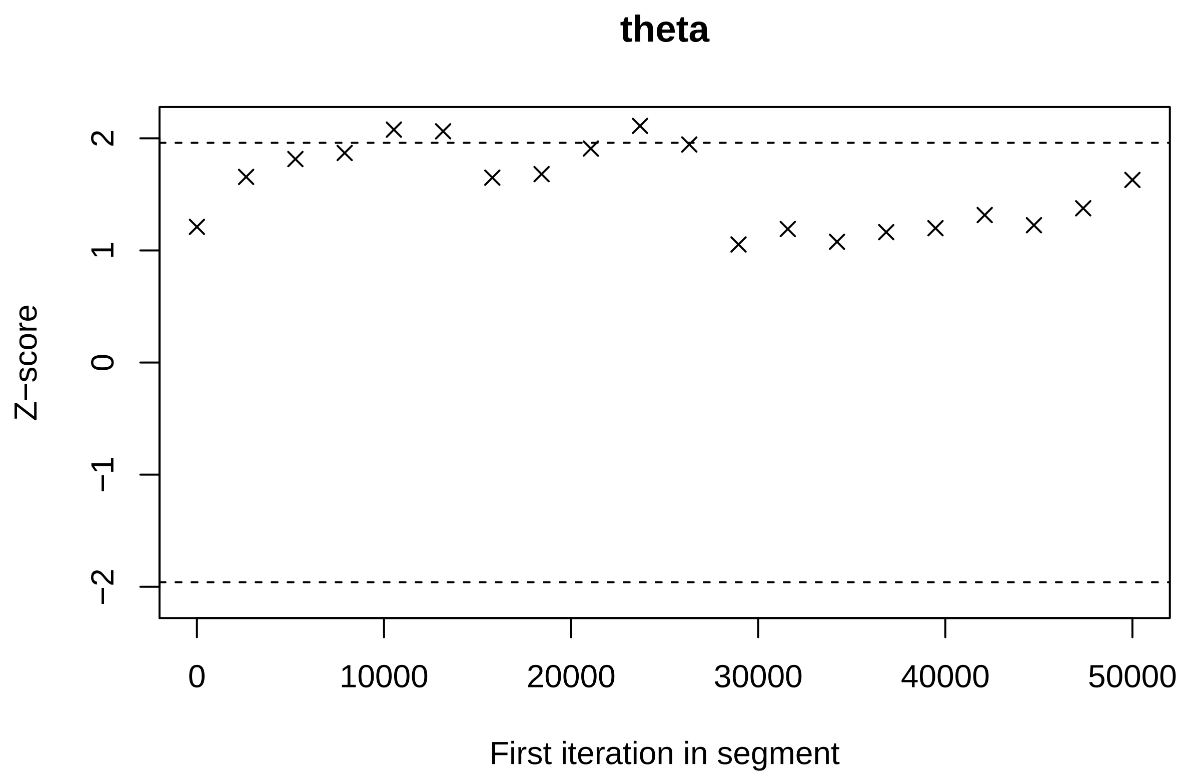
```
## Potential scale reduction factors:
##
##   Point est. Upper C.I.
## x       1.01       1.01
```

```
# borderline OK
```

```
# Geweke (single chain test)
#
# make an mcmc object used by the coda package
chain.mcmc <- mcmc(data=mala.out)
varnames(chain.mcmc) <- "theta"
# returns z-score (to be compared two-sided with standard normal)
geweke.diag(chain.mcmc)
```

```
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
## theta
##  1.21
```

```
# check if we should discard more burn-in
geweke.plot(chain.mcmc)
```


theta
Z−score
2
1
0
−1
−2
0
10000
20000
30000
40000
50000
First iteration in segment

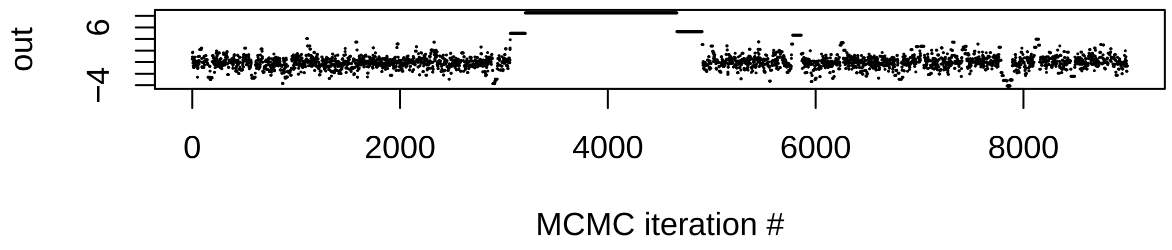
Exercise 11.6

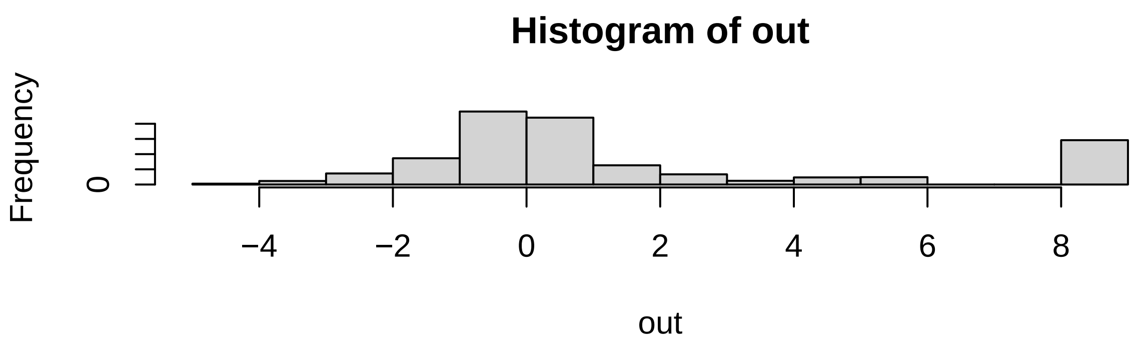
Implement a random walk Metropolis sampler for generating the standard Laplace distribution. For the increment, simulate from a normal distribution. Compare the chains generated when different variances are used for the proposal distribution. Also, compute the acceptance rates of each chain.

```r
# general 1d Gaussian proposal random walk MH
oneD.RWMH <- function(prob,
                     sigma=1.3,
                     theta1=0.0,
                     n.iter=10000){
  # space for output
  output <- numeric(n.iter)
  # first iterate given
  output[1] <- theta1
  # Calculate probability of theta 1
  p.old <- prob(theta1)
  # main iteration loop
  for(t in 2:n.iter){
    # proposal
    thetaStar <- output[t-1] + rnorm(1,sd=sigma)
    # accept probability, for numerical stability we compute
    p.star <- prob(thetaStar)
    # the log-accept prob, and then take exp
    alpha <- min(1,p.star/p.old)
    # accept/reject step
    if(runif(1)<alpha && is.finite(alpha)){
      output[t] <- thetaStar
    } else {
      output[t] <- output[t-1]
    }
  }
print(paste0("RWMH done, accept prob : ",mean(abs(diff(output))>1.0e-14)))
  return(output[1000:length(output)])
}

laplace <- function(x){return(0.5*exp(-abs(x)))}

# try algorithm (change initial condition and sigma for illustration)
out <- oneD.RWMH(laplace, theta1 = 0.0, sigma=1.3)
```

```
## [1] "RWMH done, accept prob : 0.298029802980298"
```

```r
par(mfrow=c(2,1))
plot(1:length(out),out,pch=20,cex=0.1,xlab="MCMC iteration #")
hist(out)
```





Let's perform some diagnostics

```
ESS <- function(x){ return(as.numeric(coda::effectiveSize(x))) }
ESS(out)
```

```
## [1] 10.16834
```

```
ESS(out)/length(out)
```

```
## [1] 0.00112969
```

```
# run more chains for diagnostics

mcmc1 <- mcmc(oneD.RWMH(laplace))
```

```
## [1] "RWMH done, accept prob : 0.377437743774377"
```

```
mcmc2 <- mcmc(oneD.RWMH(laplace))
```

```
## [1] "RWMH done, accept prob : 0.366736673667367"
```

```
mcmc3 <- mcmc(oneD.RWMH(laplace))
```

```
## [1] "RWMH done, accept prob : 0.350635063506351"
```

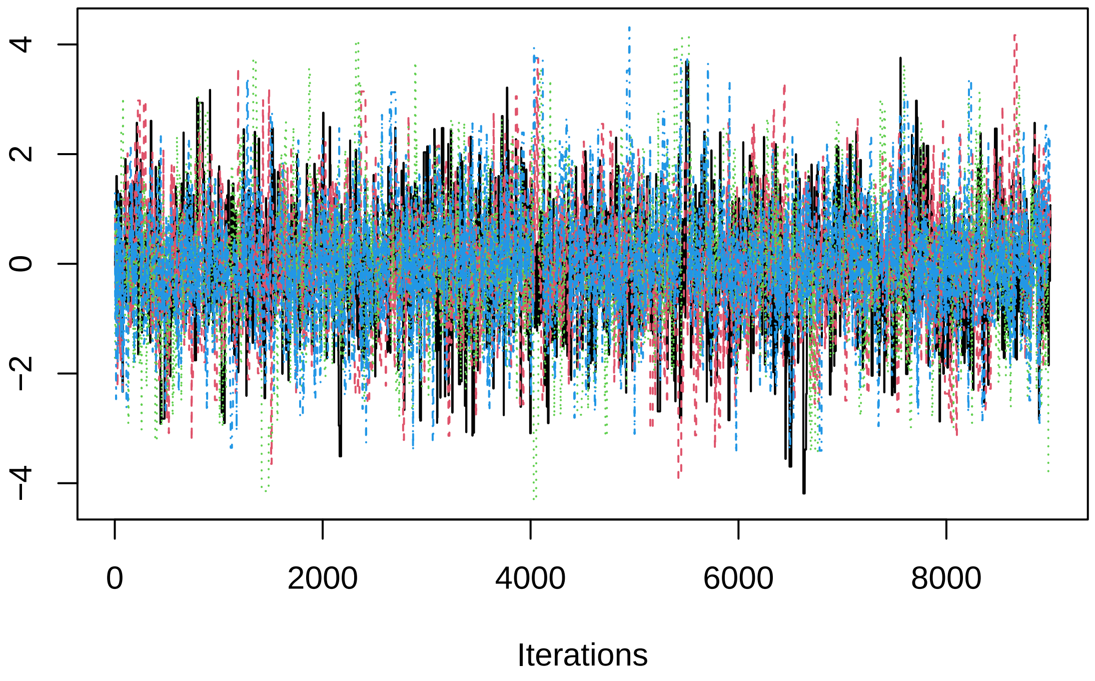
```
mcmc4 <- mcmc(oneD.RWMH(laplace))
```

```
## [1] "RWMH done, accept prob : 0.369036903690369"
```

```
ml <- mcmc.list(mcmc1,mcmc2,mcmc3,mcmc4)

effectiveSize(ml)
```

```
##     var1
## 1867.329
```

```
# quite OK
traceplot(ml)
```



```
# look at trace plot
geweke.diag(ml)
```

```
## [[1]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##   var1
## 0.5535
##
##
## [[2]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##   var1
## 0.5816
##
##
```

```
## [[3]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##    var1
## -0.7795
##
##
## [[4]]
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##  var1
## -1.78
```

```
# OK
gelman.diag(ml)
```

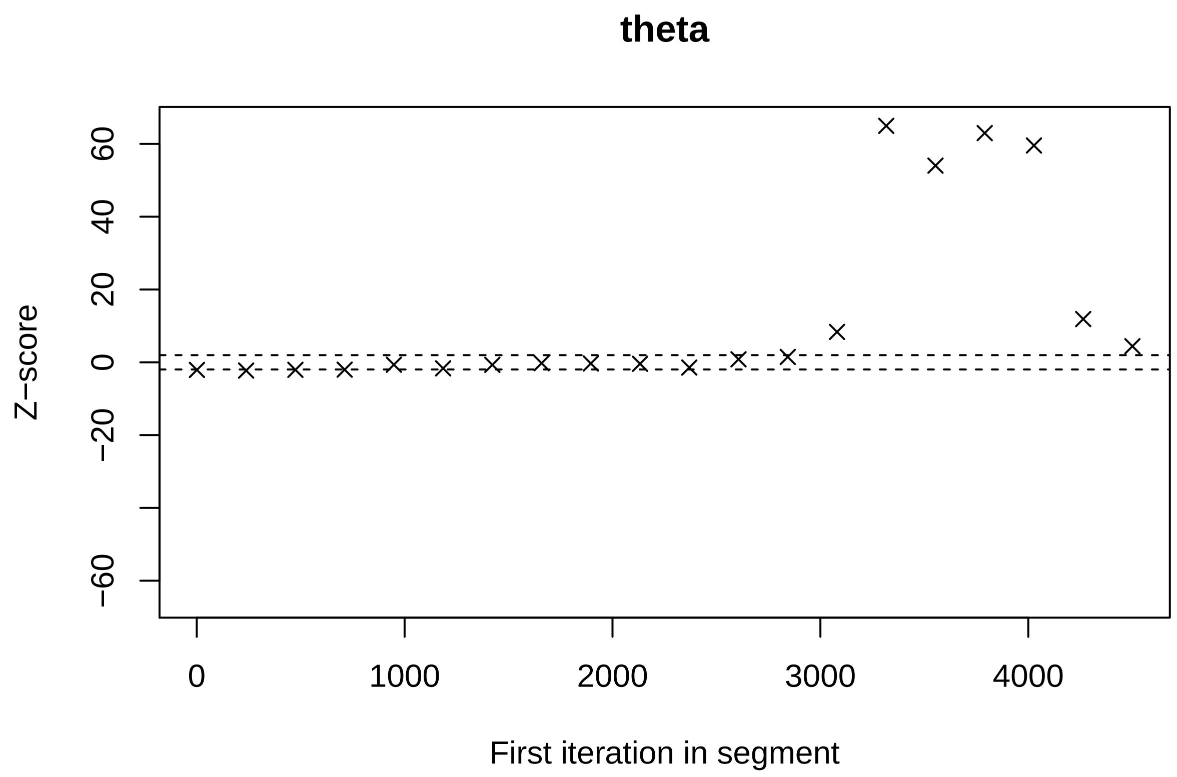
```
## Potential scale reduction factors:
##
##      Point est. Upper C.I.
## [1,]       1.01       1.01
```

```
# OK
```

```
#
# convergence tests
#
library(coda)
#
# Geweke (single chain test)
#
# make an mcmc object used by the coda package
chain.mcmc <- mcmc(data=out)
varnames(chain.mcmc) <- "theta"
# returns z-score (to be compared two-sided with standard normal)
geweke.diag(chain.mcmc)
```

```
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##  theta
## -2.092
```

```
# check if we should discard more burn-in
geweke.plot(chain.mcmc)
```



Problem 1

In this problem we consider a simple special case of a generalized linear model, where the observations y_i, $i = 1, \dots, n$ are modeled as

$$y_i \sim \text{Exponential}(\text{Exp}(\alpha + \beta x_i)), \quad i = 1, \dots, n \tag{2}$$

Where x_i, $i = 1, \dots, n$ is a fixed covariate (e.g y_i is the price of an apartment, and x_i is size (measured in square-meters) of the apartment). Notice in particular that we use the scale parametrization of the exponential distribution, so $E(y_i) = \exp(\alpha + \beta x_i)$. The parameters α and β may take any values, and we let the parameter vector $\boldsymbol{\theta}$ be defined as $\boldsymbol{\theta} = (\alpha, \beta)$

In this problem, we are going to perform a Bayesian analysis of the data set under this model. We assume the priors $\alpha \sim N(0, 10^2)$ and $\beta \sim N(0, 10^2)$

Simulate data in R

```r
##################################################################
###--- Problem 1

# from exercise set
# simulate data
set.seed(123)
n <- 100
x <- runif(n,min=-1)
y <- rexp(n,rate=1.0/exp(0.4+0.5*x))
```

```r
# log-target distribution (theta=(alpha,beta))
lp <- function(theta){
  alpha <- theta[1]
  beta <- theta[2]
  log.like <- sum(dexp(y,rate=1.0/exp(alpha+beta*x),log=TRUE))
  log.prior <- dnorm(alpha,sd=10.0,log=TRUE) + dnorm(beta,sd=10.0,log=TRUE)
  return(log.like+log.prior)
}
# end from exercise set


#
#  point 1,a)
#

# bivariate RWMH method (from exercise set 7)
twoDRWMH <- function(lprob, # log-probability density kernel
                     Sigma=diag(2), # default proposal covariance = identity matrix
                     theta1=c(0.0,0.0), # default initial configuration
                     n.iter=10000){
  # allocate output space
  out <- matrix(0.0,n.iter,2)
  out[1,] <- theta1

  # store old lprob
  lp.old <- lprob(theta1)

  # cholesky factorization of Sigma for fast sampling
  L <- t(chol(Sigma)) #lower triangular factor

  # accept counter
  Nacc <- 0
  # main iteration loop
  for(i in 2:n.iter){
    # proposal
    thetaStar <- out[(i-1),] + L%*%rnorm(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),]
    }
  } # main iteration loop
  print(paste0("RWMH done, accept rate : ",Nacc/(n.iter-1)))
  return(out)
} # function

# initial run to get a reasonable estimate of the covariance
sim <- twoDRWMH(lprob = lp)
```

```
## [1] "RWMH done, accept rate : 0.0366036603660366"
```

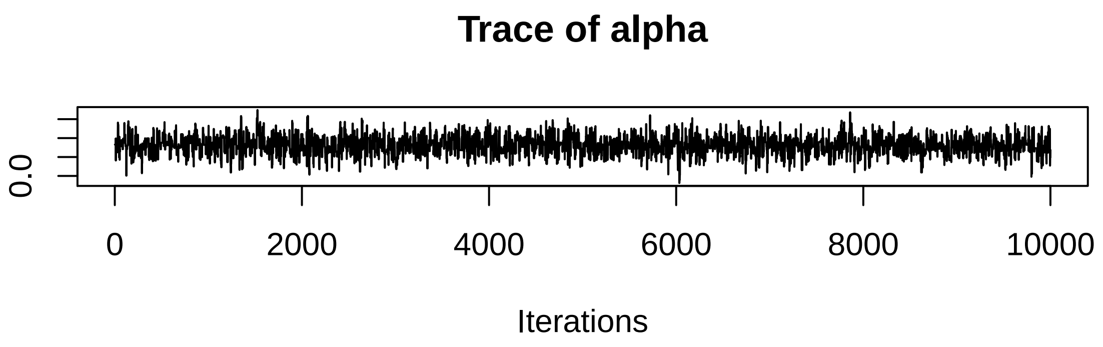
```r
init.theta <- colMeans(sim)
init.cov <- cov(sim)

# main run
#
sim <- twoDRWMH(lprob = lp,Sigma = 4.0*init.cov, theta1=init.theta)
```

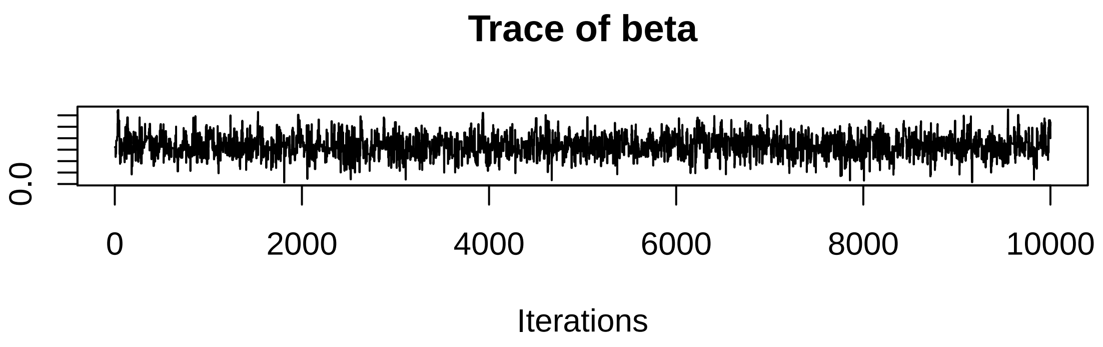
```
## [1] "RWMH done, accept rate : 0.295129512951295"
```

```r
colnames(sim) <- c("alpha","beta")
mcmc1 <- mcmc(sim)
```

b. Let's check our samples

```r
#
# point 1.b)
#

# check trace plots
par(mfrow=c(2,1))
traceplot(mcmc1)
```





```
# removing burn in does not seem neccessary, as we already started
# the last MCMC run close to the posterior mean

# check diagnostics and ESS
effectiveSize(mcmc1)
```

```
##    alpha     beta
## 1331.369 1289.008
```

```
geweke.diag(mcmc1)
```

```
##
## Fraction in 1st window = 0.1
## Fraction in 2nd window = 0.5
##
##   alpha    beta
## 1.29987 0.03013
```

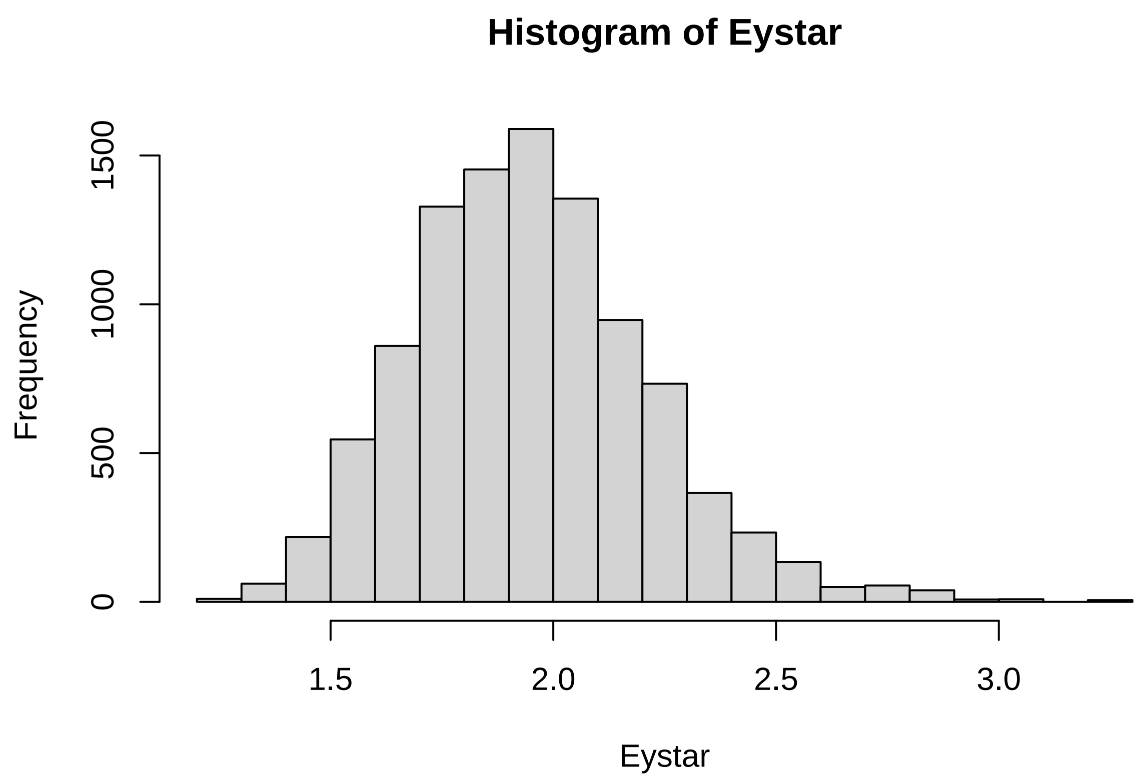
```
# all look OK, with ESS>1000
```

c. Let's predict $y*$. By sampling from the posterior distribution $p(\boldsymbol{\theta}|\boldsymbol{y})$ and transforming it to $\exp(\alpha_t + \beta_t \cdot 0.5)$.

This is also called the predictive distribution $E(y * |x, \boldsymbol{y})$

```
#
# 1.c)
#
```

```r
# predictive mean distribution
Eystar <- exp(mcmc1[,1]+0.5*mcmc1[,2])
par(mfrow=c(1,1))
hist(Eystar)
```



```r
mean(Eystar)
```

```
## [1] 1.948759
```

```r
sd(Eystar)
```

```
## [1] 0.2660259
```

d.)

Bibliography