Solution to Exercices

F. Guillaume Blanchet and Mathew Talluto

August 17, 2017

1 Rejection Sampling - Solution

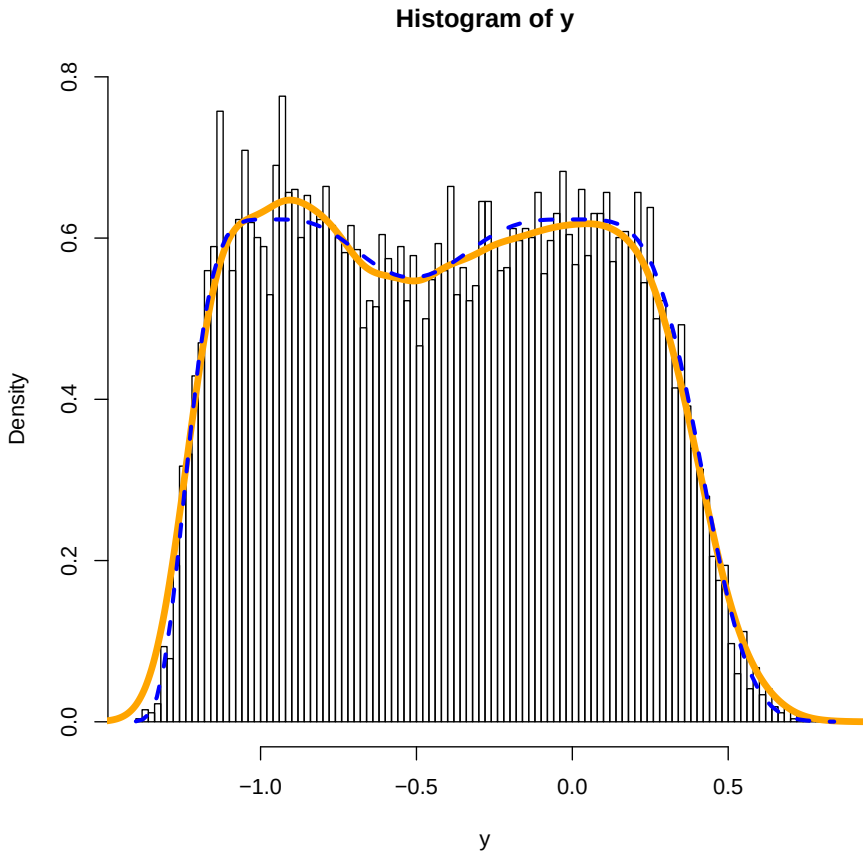
1.1 Code

```
### Result object
y <- numeric()

### number of iterations
niter<-50000

### Define candidate distribution
cand <- function(x) {
  dunif(x,-3,3)
}

### Define target distribution
target <-function(x) {
  exp(-(x^2+sin(2*x))^4-0.473)
}

### Define M
M <- 0.44
M <- target(0)/cand(0)

### Counter
counter <- 1

for(i in 1:niter){
  ### Sample candidate value
  yCand <- runif(1,-3,3)

  ### Calculate acceptance probability
  p <- target(yCand)/(M * cand(yCand))
  ### Sample a value from uniform distribution
```

```
  U <- runif(1)

  if(U < p){
    ### Accept candidate value
    y[counter] <- yCand
    ### Increase counter
    counter <- counter + 1
  }
}
```

1.2 Graphical results

```
### Histogram
hist(y,breaks=100,freq=FALSE)
### Density plot of the true function
lines(density(y),lwd=5,col="orange")
### Sampled density plot
curve(target,col="blue",lwd=3,lty=2,add=TRUE)
```



2 Sample a distribution using a Metropolis-Hasting Algorithm

2.1 Distribution

```
### Distribution to sample
distriEx1<-function(x) {exp(-(x+sin(3*x))^2+0.195)/2}
```

2.2 Code

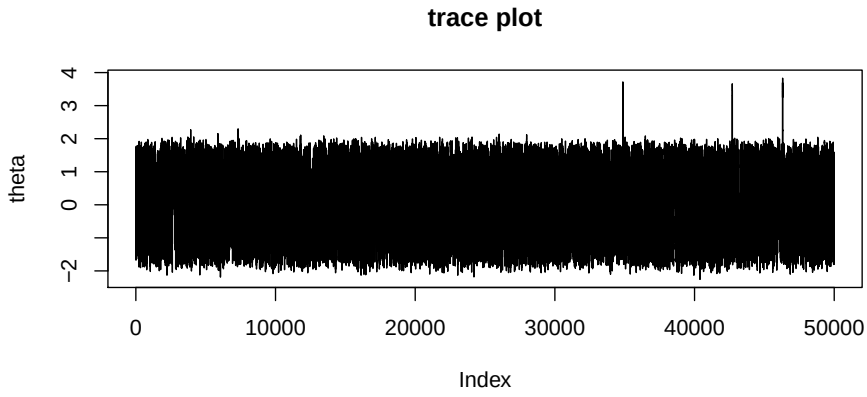
```
### Number of iterations
niter<-50000
```

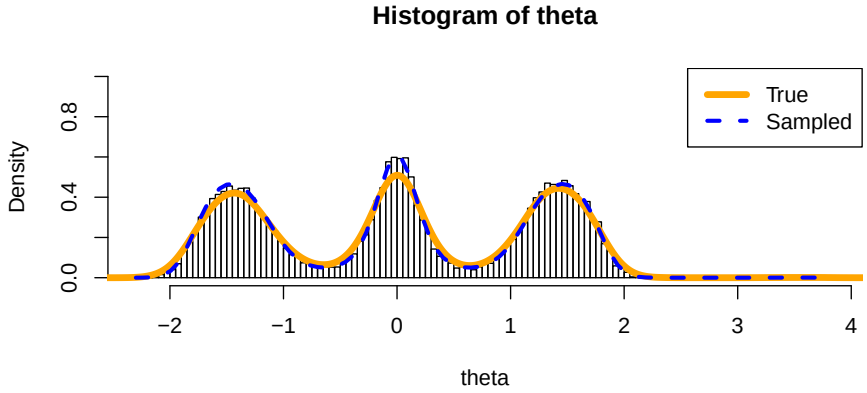
```r
### Starting value
thetaStart <- 0

### Candidate value
theta_c <- 0

### Vector of saved value
theta<-numeric()

### Initiate result vector
theta[1]<-thetaStart

### acceptance probability
r_c<-numeric()

for(i in 2:niter){
  ### Sample candidate value from jumping distribution
  theta_c <- rnorm(1,mean=theta[i-1],sd=1)

  ### Sample from target distribution using candidate value
  distri_c <- distriEx1(theta_c)

  ### Sample from target distribution using reference value
  distri_Ref <- distriEx1(theta[i-1])

  ### Calculate acceptance probability
  r_c <- distri_c/distri_Ref

  ### Sample a value from uniform distribution
  U <- runif(1)
        if(U < r_c){
    ### Accept candidate value
          theta[i]<-theta_c
        }else{
    ### Keep reference value
          theta[i]<-theta[i-1]
        }
}
```

2.3 Graphical results

```r
par(mfrow=c(2,1))
```

```r
### Trace plot
plot(theta,main="trace plot",type="l")

### Histogram
hist(theta,freq=FALSE,breaks=100,ylim=c(0,1))

### Density plot of the true function
lines(density(theta),lwd=5,col="orange")

### Sampled density plot
curve(distriEx1(x),add=TRUE,col="blue",lwd=3,lty=2)

### Legende
legend("topright",legend=c("True","Sampled"),lwd=c(5,3),
       lty=c(1,2),col=c("orange","blue"))
```





3 Sample a distribution using an adaptative Metropolis-Hasting Algorithm

3.1 Distribution

```
### Distribution to sample
distriEx2<-function(x) {exp(-(x^2+sin(3*x))^4-0.344)}
```

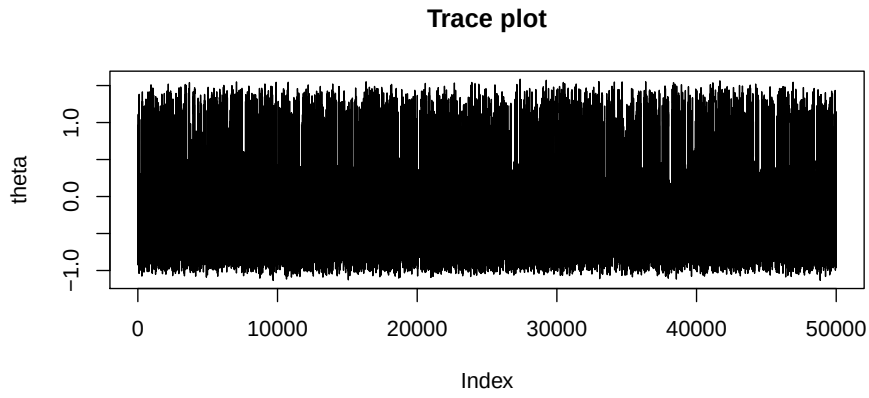
3.2 Code

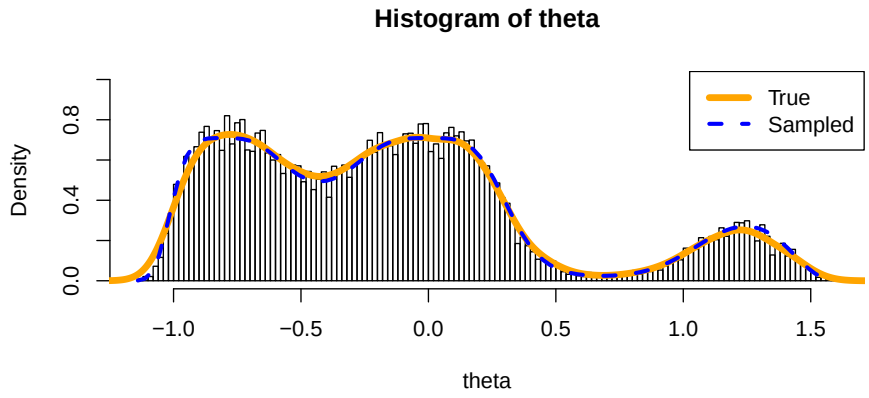
```
### Number of iterations
niter <- 50000

### Number of burnin iterations
burnin <- 20000

### Starting value
thetaStart <- 0

### Candidate value
theta_c <- 0

### Vector of saved value
theta<-numeric()

### Initiate result vector
theta[1]<-thetaStart

### acceptance probability
r_c<-numeric()

### Adatptation value
A <- 1

for(i in 2:niter){
  ### Sample candidate value from jumping distribution
  theta_c <- rnorm(1,mean=theta[i-1],sd=A)

  ### Sample from target distribution using candidate value
  distri_c <- distriEx2(theta_c)

  ### Sample from target distribution using reference value
  distri_Ref <- distriEx2(theta[i-1])
```

```
  ### Calculate acceptance probability
  r_c <- distri_c/distri_Ref

  ### Sample a value from uniform distribution
  U <- runif(1)
  if(U < r_c){
    ### Accept candidate value
    theta[i]<-theta_c
    if(burnin < i){
      ### Modify adaptation value
      A = A * 1.01
    }
  }else{
    ### Keep reference value
    theta[i]<-theta[i-1]
    if(burnin < i){
      ### Modify adaptation value
      A = A / 1.01
    }
  }
}
```

3.3 Graphical results

```
par(mfrow=c(2,1))

### Trace plot
plot(theta,main="Trace plot",type="l")

### Histogram
hist(theta,freq=FALSE,breaks=100,ylim=c(0,1))

### Density plot of the true function
lines(density(theta),lwd=5,col="orange")
### Sampled density plot
curve(distriEx2(x),add=TRUE,col="blue",lwd=3,lty=2)

### Legend
legend("topright",legend=c("True","Sampled"),lwd=c(5,3),
       lty=c(1,2),col=c("orange","blue"))
```





4 Sample a distribution using a single component adaptive Metropolis-Hasting Algorithm

4.1 Distribution

```
### Define target distribution (bivariate normal distribution)
library(mvtnorm)

### Mean of the bivariate normal distribution
mu <- c(10,30)

### Covariance of the bivariate normal distribution
sigma <- matrix(c(1,0.7,0.7,2),2,2)
```

```r
### Distribution
distriEx3<-function(x) {
  dmvnorm(x,mean=mu,sigma=sigma)
}
```

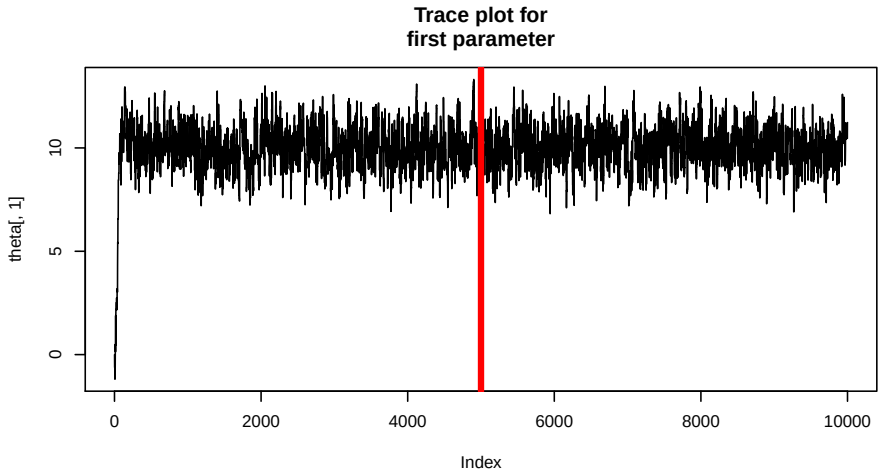
4.2 Code

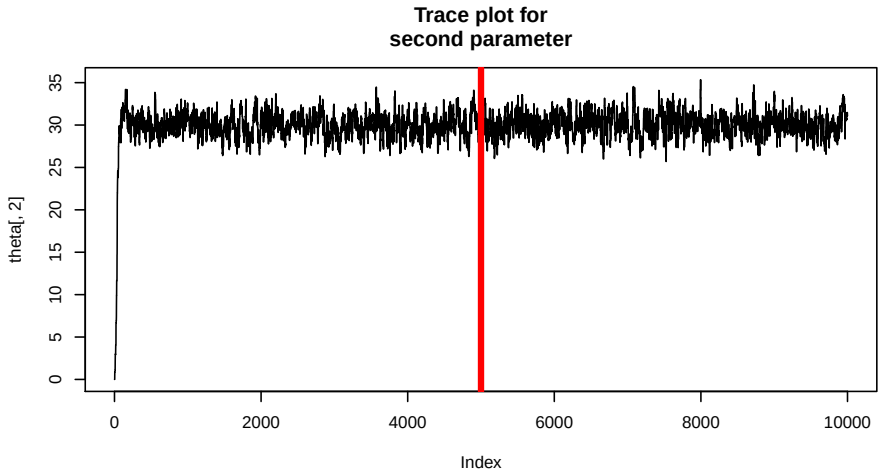
```r
### Number of iterations
niter <- 10000

### Number of burnin iterations
burnin <- 5000

### Vector of starting value
thetaStart <- c(0,0)

### Vector of candidate values
theta_c <- thetaStart

### Matrix of results value
theta <- matrix(NA,ncol=2,nrow=niter)

### Initiate result matrix
theta[1,] <- thetaStart

### Number of parameters
nparam <- length(mu)

### Acceptance probability
r_c <- numeric()

### Adatptation values
A <- c(1,1)

for(i in 2:niter){
  for(j in 1:nparam){
    ### Sample candidate value from jumping distribution
    theta_c[j] <- rnorm(1,mean=theta[i-1,j],sd=A)

    ### Sample from target distribution using candidate value
    distri_c <- distriEx3(theta_c)

    ### Sample from target distribution using reference value
```

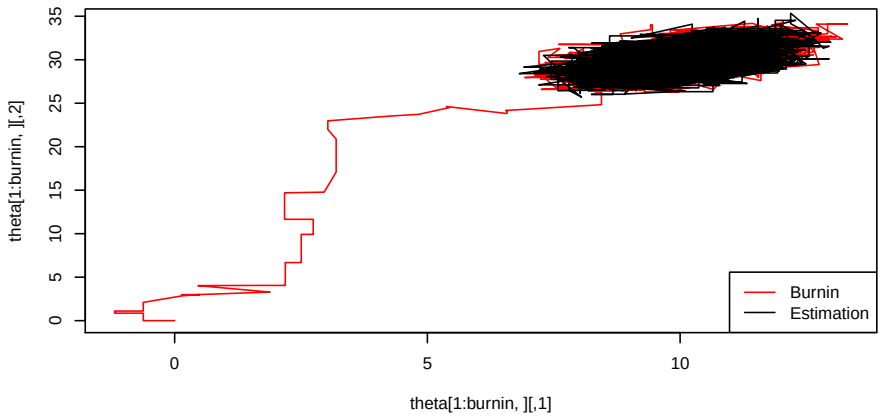
```
    distri_Ref <- distriEx3(theta[i-1,])

    ### Calculate acceptance probability
    r_c <- distri_c/distri_Ref

    ### Sample value from a uniform distribution
    U <- runif(1)
    if(U < r_c){
      ### Accept candidate value
      theta[i,j]<-theta_c[j]
      if(burnin < i){
        ### Modify adaptation value
        A[j] = A[j] * 1.01
      }
    }else{
      ### Keep reference value
      theta[i,j]<-theta[i-1,j]
      if(burnin < i){
        ### Modify adaptation value
        A[j] = A[j] / 1.01
      }
    }
  }
}
```

4.3 Graphical results

```
par(mfrow=c(3,1))

### Trace plot of the first parameter
plot(theta[,1],main="Trace plot for\nfirst parameter",type="l")
abline(v=burnin,col="red",lwd=4)

### Trace plot of the second parameter
plot(theta[,2],main="Trace plot for\nsecond parameter",type="l")
abline(v=burnin,col="red",lwd=4)

### Distribution of both parameters
plot(theta[1:burnin,],type="l",col="red")
lines(theta[burnin:niter,],type="l")
legend("bottomright",legend=c("Burnin","Estimation"),lty=c(1,1),col=c("red","black"))
```


Trace plot for
first parameter
theta[, 1]
0
5
10
0
2000
4000
6000
8000
10000
Index


Trace plot for
second parameter
theta[, 2]
0
5
10
15
20
25
30
35
0
2000
4000
6000
8000
10000
Index


theta[1:burnin,][,2]
0
5
10
15
20
25
30
35
0
5
10
theta[1:burnin,][,1]
Burnin
Estimation

5 Gibbs sampler using conjugate priors

5.1 Data

```
Y <- c(15,19.59,15.06,15.71,14.65,21.4,17.64,18.31,15.12,14.40)
n <- length(Y)
```

5.2 Define hyperparameter

```
hyperParamMu <- c("mu0" = 16, "sigma0" = 0.4)
hyperParamSigma <- c("alpha0" = 1, "beta0" = 3)
```

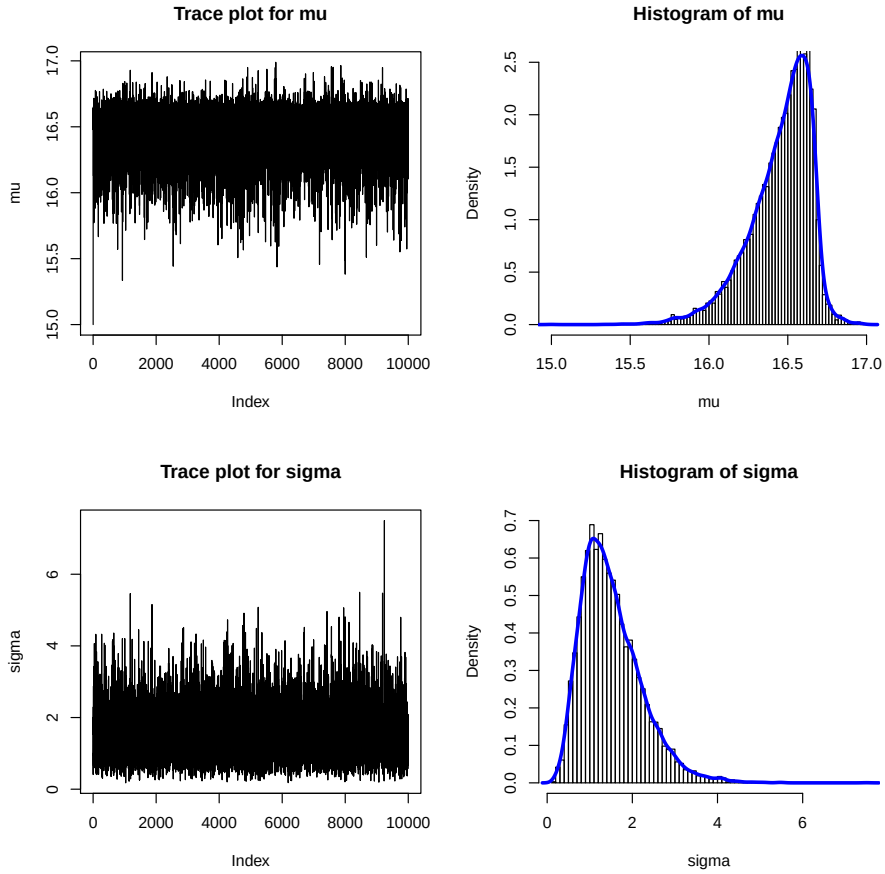
5.3 Code

```
### Number of iterations
niter <- 10000

### Vector of starting value
mu <- 15
sigma <- 1

### Sample each parameters
for(i in 1:niter){
  #=-=-=-=-=-=-
  ### Sample mu
  #=-=-=-=-=-=-
  ### Conjugate prior for the mean of sigma
  ConjSigmaMean <- (1/hyperParamMu[2] + n/sigma[i]^2)^(-1)

  ### Conjugate prior for the mean of mu
  ConjMuMean <- (hyperParamMu[1]/hyperParamMu[2] + sum(Y)/sigma[i]^2)/ConjSigmaMean^(-1)

  ### Sample mu
  muSample <- rnorm(1,ConjMuMean, ConjSigmaMean)

  ### Save value
  mu <- c(mu,muSample)
  #=-=-=-=-=-=-=-=
  ### Sample sigma
  #=-=-=-=-=-=-=-=
  ### Conjugate prior for the mean of sigma
```

```
  ConjAlphaSigma <- (hyperParamSigma[1]+n/2)

  ### Conjugate prior for the mean of mu
  ConjBetaSigma <- (hyperParamSigma[2]+sum(Y-mu[i])/2)

  ### Sample the sigma from the conjugate prior
  sigmaSample <- rgamma(1,ConjAlphaSigma, ConjBetaSigma)

  ### Save value
  sigma <- c(sigma, sigmaSample)
}
```

5.4 Graphical results

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

### Trace plot of the first parameter
plot(mu,main="Trace plot for mu",type="l")

### Histogram and density plot
hist(mu,freq=FALSE,breaks=100,ylim=c(0,2.5))
lines(density(mu),col="blue",lwd=3)

### Trace plot of the first parameter
plot(sigma,main="Trace plot for sigma",type="l")

### Histogram and density plot
hist(sigma,freq=FALSE,breaks=100,ylim=c(0,0.7))
lines(density(sigma),col="blue",lwd=3)
```



6 Gibbs sampler using Metropolis algorithm

6.1 Data

```
Y <- c(15,19.59,15.06,15.71,14.65,21.4,17.64,18.31,15.12,14.40)
```

6.2 Define important functions

```
### Function to sample the candidate
cand <- function(params, T, k)
{
        params[k] <- rnorm(1, params[k], T[k])
        return(params)
```

```
}

### Function to calculate the log-likelihood
loglik <- function(params, data)
{
        if(params[2] <= 0) return(NA)
        sig <- sqrt(1/params[2])
        sum(dnorm(data, params[1], sig, log=TRUE))
}

### Function to define the log-priors
logprior <- list(
        function(mu) dnorm(mu, 16, 0.4),
        function(tau) dgamma(tau, 1,3)
)
```

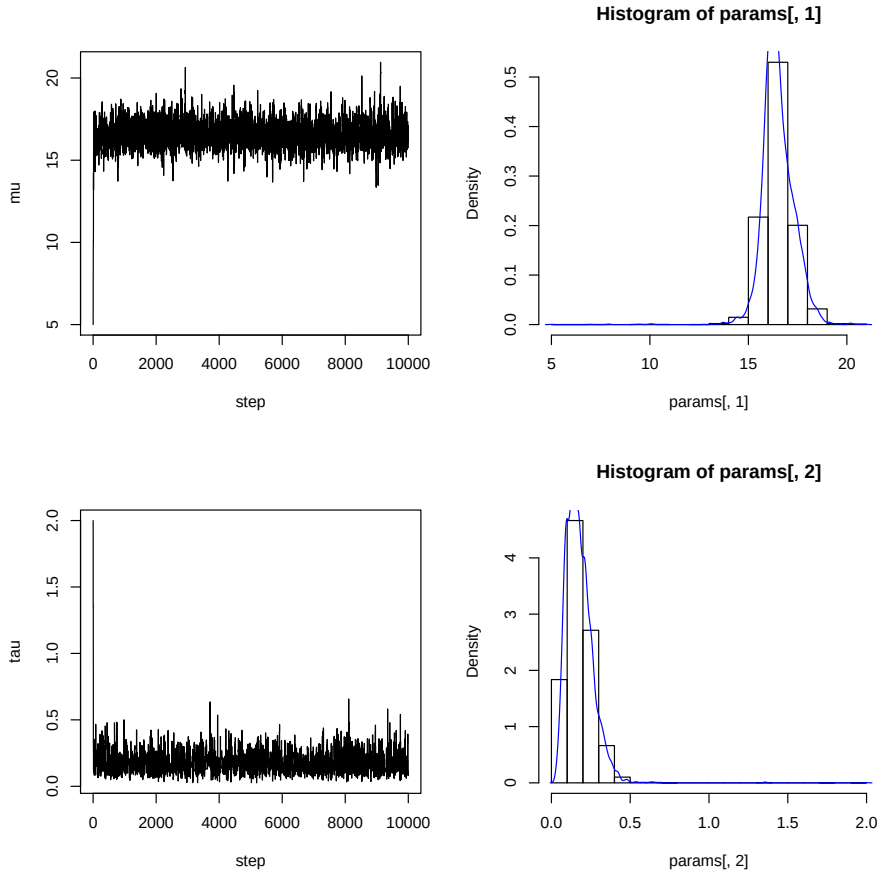
6.3 Code

```
### Basic objects
nsteps <- 10000
nparam <- 2
tuning <- c(1.5, 0.4)
accepts <- 0

### Result matrix
params <- matrix(nrow=nsteps, ncol=nparam, dimnames = list(1:nsteps, c('mu', 'tau')))

### Starting value
params[1,] <- c(5, 2)

### Sample each parameters
for(t in 2:nsteps){
        current <- params[t-1,]
        for(k in 1:nparam){
                # propose a new value based on the current values and candidate distributio
                # returns a parameter vector but only changes param k
                proposal <- cand(current, tuning, k)

                # compute log posterior probabilities
                # this is the target distribution
                lp_proposal <- loglik(proposal, Y) + logprior[[k]](proposal[k])
                lp_previous <- loglik(current, Y) + logprior[[k]](current[k])

                # in case of a numerical error, such as from a negative variance, we automa
```

```
                if(! is.finite(lp_proposal)){
                        next()
                }

                accept_prob <- exp(lp_proposal - lp_previous)
                U <- runif(1,0,1)

                # if we accept, change the value, otherwise keep the previous
                if(U < accept_prob) {
                        current[k] <- proposal[k]
                        accepts <- accepts + 1
                }
        }
        params[t,] <- current
}
```

6.4 Results

```
print(paste("Acceptance Rate:", accepts/((nsteps-1)*nparam)))

## [1] "Acceptance Rate: 0.343684368436844"

par(mfrow=c(2,2))

### Trace plot of the first parameter
plot(1:nsteps, params[,1], type='l', xlab='step', ylab='mu')

### Histogram and density plot
hist(params[,1], freq=FALSE)
lines(density(params[,1]), col='blue')

### Trace plot of the second parameter
plot(1:nsteps, params[,2], type='l', xlab='step', ylab='tau')

### Histogram and density plot
hist(params[,2], freq=FALSE)
lines(density(params[,2]), col='blue')
```


mu
step
Histogram of params[, 1]
Density
params[, 1]
tau
step
Histogram of params[, 2]
Density
params[, 2]