Exercise set 7

Bjørn Christian Weinbach

28th October, 2020

Clear R environment

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

Problem 0

$N = 10$ measurments of the measurements of the hardness of a new material (alloy) gave the following data:

168	185	164	182	169
181	172	185	172	180

A typical approach is to assume that the hardness measurements follow a $N(\mu; \sigma^2)$ distribution.

Calculate the empirical mean $\hat{\mu}$ and standard deviation

$\hat{\sigma}$.

```
data <- c(168, 185, 164, 182, 169, 181, 172, 186, 172, 180)

print(paste("Empirical mean: ", mean(data)))
```

```
## [1] "Empirical mean:  175.9"
```

```
print(paste("Empirical sd: ", sd(data)))
```

```
## [1] "Empirical sd:  7.79529772790409"
```

Calculate the standard deviation of $\hat{\mu}$

According to the wikipedia article on standard error accessed 22nd October, 2020 (Wikipedia contributors 2020a) For each random variable, the sample mean is a good estimator of the population mean, where a "good" estimator is defined as being efficient and unbiased. Of course the estimator will likely not be the true value of the population mean since different samples drawn from the same distribution will give different sample means and hence different estimates of the true mean. Thus the sample mean is a random variable, not a constant, and consequently has its own distribution.

We can therefore calculate the standard deviation of $\hat{\mu}$, also called the standard error of the mean as follows:

$$\sigma_{\bar{X}} = \frac{\sigma}{\sqrt{N}} \tag{1}$$

(Wikipedia contributors 2020b)

Since the population mean is seldom known. We can estimate this like so:

$$\sigma_{\bar{X}} \approx \frac{s}{\sqrt{N}} \tag{2}$$

Where s is the sample standard deviation.

In R:

```
print(paste("Standard error: ", sd(data)/sqrt(length(data))))
```

```
## [1] "Standard error:  2.46508958593124"
```

Calculate 95% confidence intervals for μ and σ

Since we have a unknown μ and σ for the population and the sample size is small, we will use a T values for our confidence interval for the population mean.

The confidence interval is

$$[\bar{x} + t_{n-1,\alpha/2} \cdot \frac{s}{\sqrt{n}}, \bar{x} + t_{n-1,1-\alpha/2} \cdot \frac{s}{\sqrt{n}}] \tag{3}$$

in R:

```
# Calculate sample mean, sd and no of observations
xbar <- mean(data)
s <- sd(data)
n <- length(data)

# Alpha
a <- 0.05

# Calculate confidence interval. qt is T-distribution
left <- xbar + qt(a/2, n-1) * s/sqrt(n)
right <- xbar + qt(1 - (a/2), n-1) * s/sqrt(n)
ci <- c(left, right)

# Display confidence interval
ci
```

```
## [1] 170.3236 181.4764
```

To calculate the confidence interval for σ we use the formula in the text of the exercise set.

```
left <- sqrt((n-1)*s^2 / qchisq(1 - a/2, df=n-1))
right <- sqrt((n-1)*s^2 / qchisq(a/2, df=n-1))
ci <- c(left, right)
ci
```

```
## [1]  5.36188 14.23117
```

Bootstrapping $\hat{\mu}$

Now we resort to bootstrapping to estimate the calculations we have above from our data. We first do this for the statistic $\hat{\mu}$

```
library(boot)

# Function for statistic - sample mean
meanestfunc <- function(data,i)
  mean(data[i])

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

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = data, statistic = meanestfunc, R = 5000)
##
##
## Bootstrap Statistics :
##     original  bias    std. error
## t1*    175.9 0.02166     2.32338
```

```
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%   (171.3, 180.4 )   (171.5, 180.4 )
##
## Level     Percentile            BCa
## 95%   (171.4, 180.3 )   (171.1, 180.2 )
## Calculations and Intervals on Original Scale
```

We see that our calculations for the standard error is close to the estimated standard error by bootstrapping. The confidence interval is also close to our interval. Though the calculated one used t-values and is therefore slightly wider.

Bootstrapping $\hat{\sigma}$

```
# Function for statistic - sample mean
stdfunc <- function(data,i)
  sd(data[i])

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

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = data, statistic = stdfunc, R = 5000)
##
##
## Bootstrap Statistics :
##     original     bias    std. error
## t1* 7.795298 -0.4438451    1.052887
```

```
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%   ( 6.176, 10.303 )   ( 6.295, 10.396 )
##
## Level     Percentile            BCa
## 95%   ( 5.195,  9.295 )   ( 6.223, 10.152 )
## Calculations and Intervals on Original Scale
## Some BCa intervals may be unstable
```

Problem 1

Consider the 1-dimensional random walk Metropolis Hastings function given in the R-example.

In typical applications of MCMC, the log-target function (log $g(\boldsymbol{\theta})$ in the lecture notes) is often expensive to evaluate. Modify the code so that only one evaluation of log-target function per MCMC iteration is required. Check that your implementation is still correct using a target distribution of your choice.

Modified code:

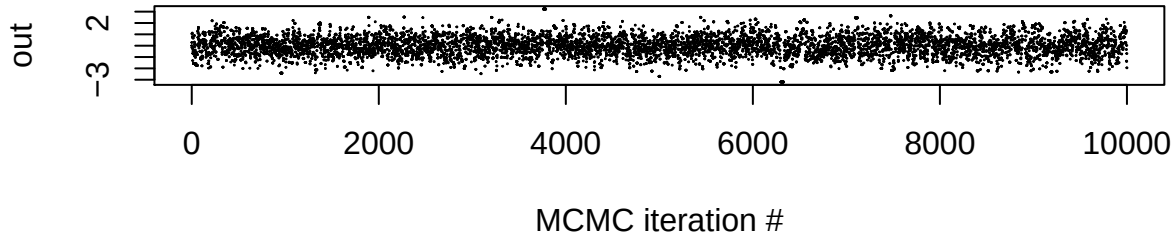
```r
# general 1d Gaussian proposal random walk MH
oneD.RWMH <- function(lprob, #notice log-density kernel!
                      sigma=1.0,
                      theta1=0.0,
                      n.iter=10000){
  # space for output
  output <- numeric(n.iter)
  # first iterate given
  output[1] <- theta1
  # Calculate probability of theta 1
  lp.old <- lprob(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
    lp.star <- lprob(thetaStar)
    # the log-accept prob, and then take exp
    alpha <- exp(min(0.0,lp.star-lp.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)
}
```

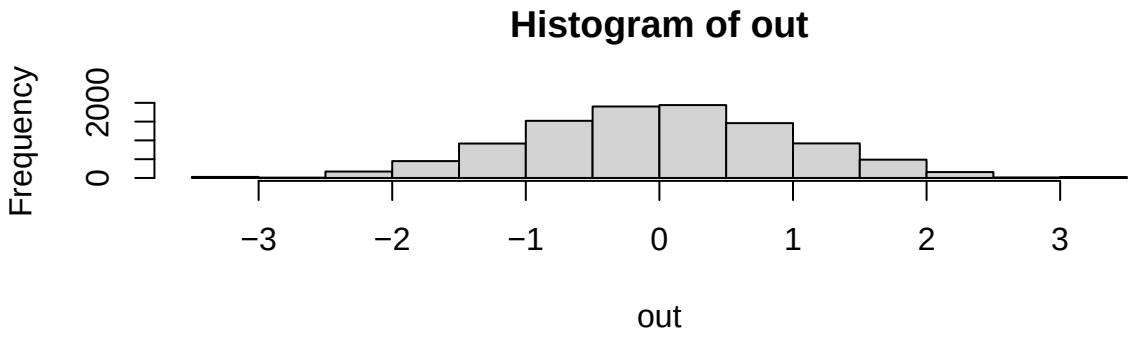
```r
lp_std_norm <- function(x){return(dnorm(x, mean=0, sd=1, log=TRUE))}

# try algorithm (change initial condition and sigma for illustration)
out <- oneD.RWMH(lp_std_norm, theta1 = 0.0, sigma=1.0)
```

```
## [1] "RWMH done, accept prob : 0.576157615761576"
```

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






```r
mean(out)
```

```
## [1] -0.001130901
```

```r
sd(out)
```

```
## [1] 0.9864623
```

Problem 2

Considering the situation in example 11.3, we will try to use Metropolis-Hastings algorithm with the target function being the posterior distribution for p

$$g(p) = 431 \log(p) + 4 \log(1 - p) \tag{4}$$

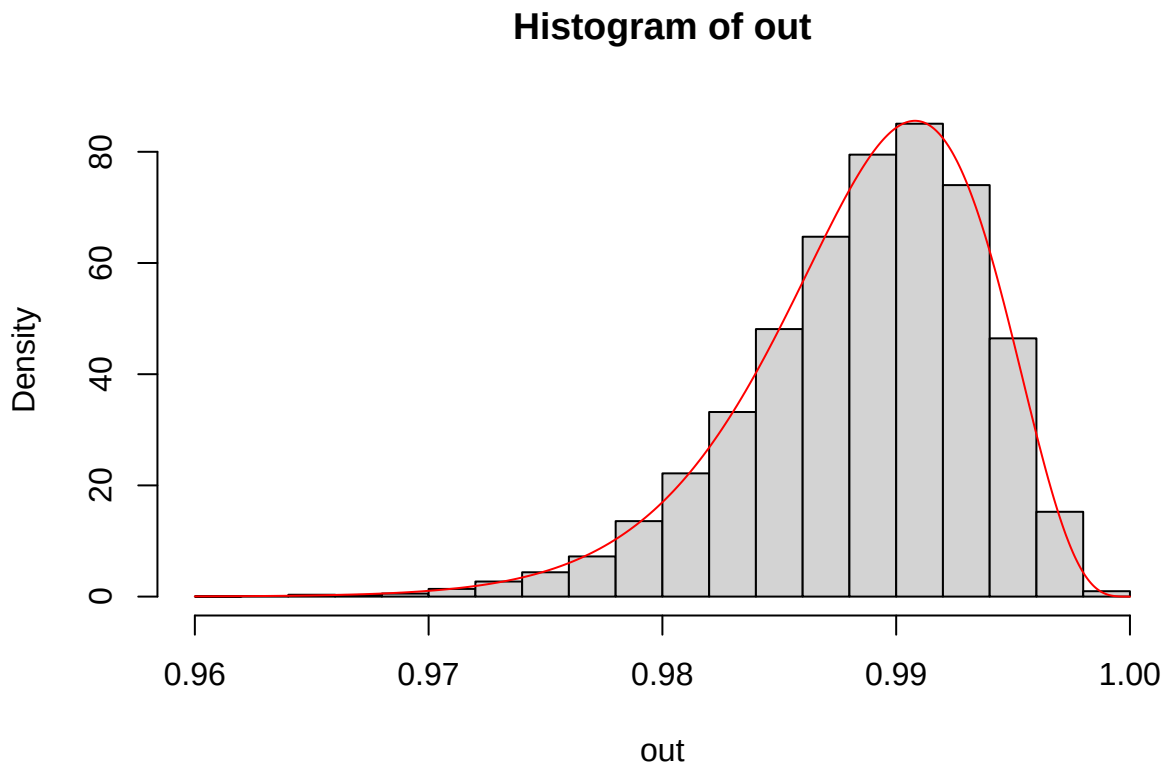
Find a proposal σ that result in an acceptance-rate

of 20% to 40%.

```
# p-target
lprob.p <- function(p){
  if(p<0.0 || p>1.0) return(-1e100)
  return(431.0*log(p) + 4.0*log(1.0-p))
}
# run RWMH and compare to reference
out <- oneD.RWMH(lprob.p,sigma=0.01,theta1=0.99,n.iter=500000)
```

```
## [1] "RWMH done, accept prob : 0.387210774421549"
```

```
# sigma = 0.02 seems like a good choice
hist(out,probability = TRUE)
xg <- seq(from=0.96,to=1.0,length.out = 1000)
lines(xg,dbeta(xg,shape1=432,shape2=5),col="red")
```




```
# looks correct,
```

Find a good proposal for mu that gives a acceptance rate of $20 - 40\%$

```
# mu-target
lprob.mu <- function(mu){
  if(mu<0.0) return(-1.0e100)
  return(431.0*log(mu/(1.0+mu)) - 6.0*log(1.0+mu))
}
```

```
out.mu  <- oneD.RWMH(lprob.mu,sigma=150.0,theta1=0.99/(1.0-0.99),n.iter=500000)
```

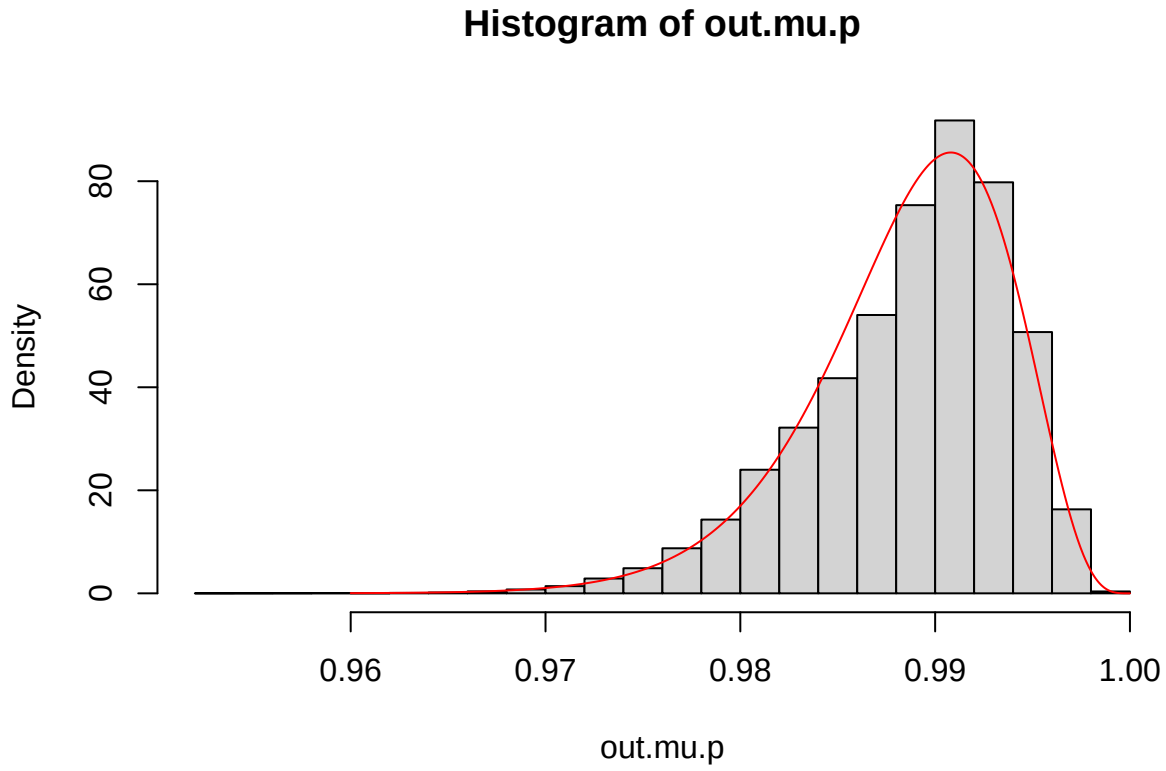
```
## [1] "RWMH done, accept prob : 0.258146516293033"
```

```
# sigma=150 seems like a reasonable choice
```

Sample μ

```
#inverse transformation to obtain p from mu:
out.mu.p <- out.mu/(1.0+out.mu)

# plot transformed samples
hist(out.mu.p,probability = TRUE)
lines(xg,dbeta(xg,shape1=432,shape2=5),col="red")
```




```
# also looks correct
```

Problem 3

Let

$$\boldsymbol{P} = \begin{pmatrix} 1 & 1 \\ 1 & 2 \end{pmatrix} \tag{5}$$

and

$$\boldsymbol{m} = \begin{pmatrix} 1 \\ -1 \end{pmatrix} \tag{6}$$

Below we is an implementation of a bivariate RWMH algorithm in R:

```r
# bivariate RWMH method
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
```

Test the implementation

In the first case, we will test the implementation by trying to sample the following cases:

- A bivariate normal distribution $N(\boldsymbol{m}, \boldsymbol{P}^{-1})$
- A bivariate t-distribution with location $\boldsymbol{m}$, scale matrix $\boldsymbol{P}^{-1}$ and $\nu = 4$ degrees of freedom.

Bivariate normal distribution

The bivariate normal $N(\boldsymbol{m}, \boldsymbol{P}^{-1})$ has log density:

$$\log(\boldsymbol{x}) = -0.5(\boldsymbol{x} - \boldsymbol{m})^T \boldsymbol{P}(\boldsymbol{x} - \boldsymbol{m}) \tag{7}$$

In R:

```
P <- matrix(c(1,1,1,2),2,2)
m <- c(1.0,-1.0)
Pinv <- solve(P)

### test 1 - bivariate normal distribution
lp_gauss <- function(x){return(-0.5*t(x-m)%*%P%*%(x-m))}

out.gauss <- twoDRWMH(lp_gauss,Sigma=2.0*Pinv,
                      theta1 = m,n.iter=100000)
```

```
## [1] "RWMH done, accept rate : 0.42310423104231"
```

```
# estimated mean
colMeans(out.gauss)
```

```
## [1]  0.9991485 -0.9981672
```

```
m
```

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

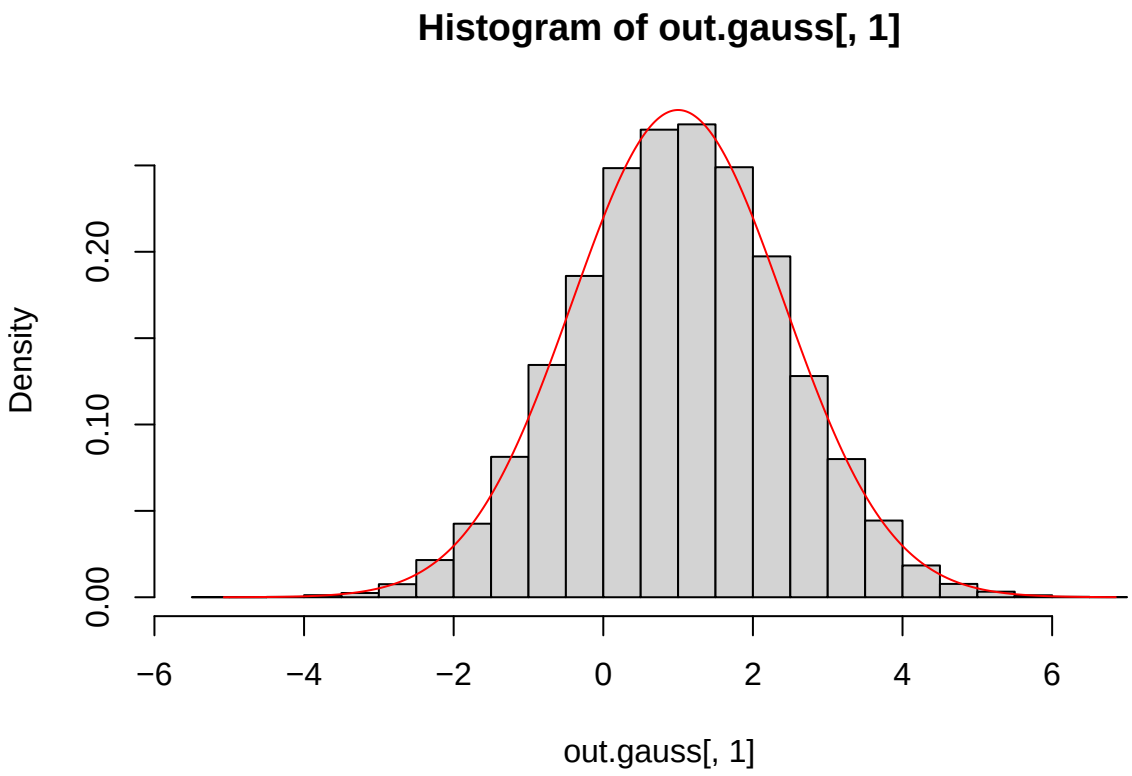
```
# estimated covariance
cov(out.gauss)
```

```
##            [,1]       [,2]
## [1,]  1.9830608 -0.9882161
## [2,] -0.9882161  0.9908166
```

```
Pinv
```

```
##      [,1] [,2]
## [1,]    2   -1
## [2,]   -1    1
```

```
# check x1 marginal (should N(m[1],Pinv[1,1]))
par(mfrow=c(1,1))
hist(out.gauss[,1],probability = TRUE)
xg <- seq(from=min(out.gauss[,1]),to=max(out.gauss[,1]),length.out = 1000)
lines(xg,dnorm(xg,mean=m[1],sd=sqrt(Pinv[1,1])),col="red")
```



Bivariate t-distribution

The log density for our t-distribution is

$$\log g(\boldsymbol{x}) = -3\log(1 + 0.25(\boldsymbol{x} - \boldsymbol{m})^T P(\boldsymbol{x} - \boldsymbol{m})) \tag{8}$$

in R:

```
lp_t <- function(x){return(-3.0*log(1.0+0.25*t(x-m)%*%P%*%(x-m)))}

out.t <- twoDRWMH(lp_t,Sigma=3.0*Pinv,
                      theta1 = m,n.iter=100000)
```

```
## [1] "RWMH done, accept rate : 0.397133971339713"
```

```
# check mean (should be = m)
colMeans(out.t)
```

```
## [1]  1.014247 -1.003606
```

```
m
```

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

```
# check covariance (should be nu/(nu-2)*Pinv = 2*Pinv)
cov(out.t)
```

```
##           [,1]      [,2]
## [1,]  3.913467 -1.994106
## [2,] -1.994106  1.944710
```

```
2*Pinv
```

```
##      [,1] [,2]
## [1,]    4   -2
## [2,]   -2    2
```

Problem 4

Consider the bayesian model:

- Parameter μ with prior distribution $\mu \sim N(0, 10^2)$
- Parameter τ with prior distribution $\tau \sim \text{Exp}(1)$
- Observations $\boldsymbol{y} = [y_1, \ldots, y_n]$ where each $y_i|\mu \sim N(\mu, \tau^{-1})$ independently

The posterior distribution of $\boldsymbol{\theta} = [\mu, \tau]$ may be written as

$$p(\boldsymbol{\theta}|\boldsymbol{y}) \propto \left(\prod_{i=1}^{n} p(y_i|\mu, \tau) \right) p(\mu)p(\tau) \tag{9}$$

In R:

```
# from exercise text
y <- c(1.2, 6.3, 5.4, 3.4, 7.5, 4.8, 1.9)
lprob <- function(theta){
  mu <- theta[1]
  tau <- theta[2]
  if(tau<0.0) return(-1e100) # log target effectively - infinity for negative tau
  s.dev <- sqrt(1.0/tau)
  loglike <- sum(dnorm(y,mean=mu,sd=s.dev,log=TRUE))
  mu.log.pri <- dnorm(mu,mean=0,sd=10,log=TRUE)
  tau.log.pri <- -tau # = log(exp(-tau))
  return(loglike+mu.log.pri+tau.log.pri)
}
```

Obtain 10000 samples from the posterior distribution

```
# run RWMH sampler from previous point
Sig <- matrix(c(0.8,0.0,0.0,0.014),2,2) # obtained from initial run with identity proposal cov
out.mu.tau <- twoDRWMH(lprob=lprob,Sigma=2.0*Sig,theta1 = c(mean(y),1.0/var(y)),n.iter = 100000)
```

```
## [1] "RWMH done, accept rate : 0.374643746437464"
```

```
# OK tuning

# make data frame for easier presentation
out.sigma <- 1.0/sqrt(out.mu.tau[,2])
out.df <- data.frame(out.mu.tau,out.sigma)
colnames(out.df) <- c("mu","tau","s")
# print summary statistics
summary(out.df)
```

```
##        mu               tau                s
##  Min.   :-0.5811   Min.   :0.01293   Min.   :1.005
##  1st Qu.: 3.7761   1st Qu.:0.14965   1st Qu.:1.829
```

```
##  Median : 4.3319   Median :0.21609   Median :2.151
##  Mean   : 4.3311   Mean   :0.23477   Mean   :2.279
##  3rd Qu.: 4.8879   3rd Qu.:0.29903   3rd Qu.:2.585
##  Max.   : 9.0762   Max.   :0.98993   Max.   :8.795
```

```
# print standard deviations
print("standard deviations : ")
```

```
## [1] "standard deviations : "
```

```
sqrt(diag(var(out.df)))
```

```
##        mu       tau         s
## 0.8877055 0.1153666 0.6557313
```

Bibliography

Wikipedia contributors. 2020a. "Sample Mean and Covariance — Wikipedia, the Free Encyclopedia." https://en.wikipedia.org/w/index.php?title=Sample_mean_and_covariance&oldid=938430490#Variance_of_the_sampling_distribution_of_the_sample_mean.

———. 2020b. "Standard Error — Wikipedia, the Free Encyclopedia." https://en.wikipedia.org/w/index.php?title=Standard_error&oldid=978217434.