STAT34800 HW5

Sarah Adilijiang

Problem A

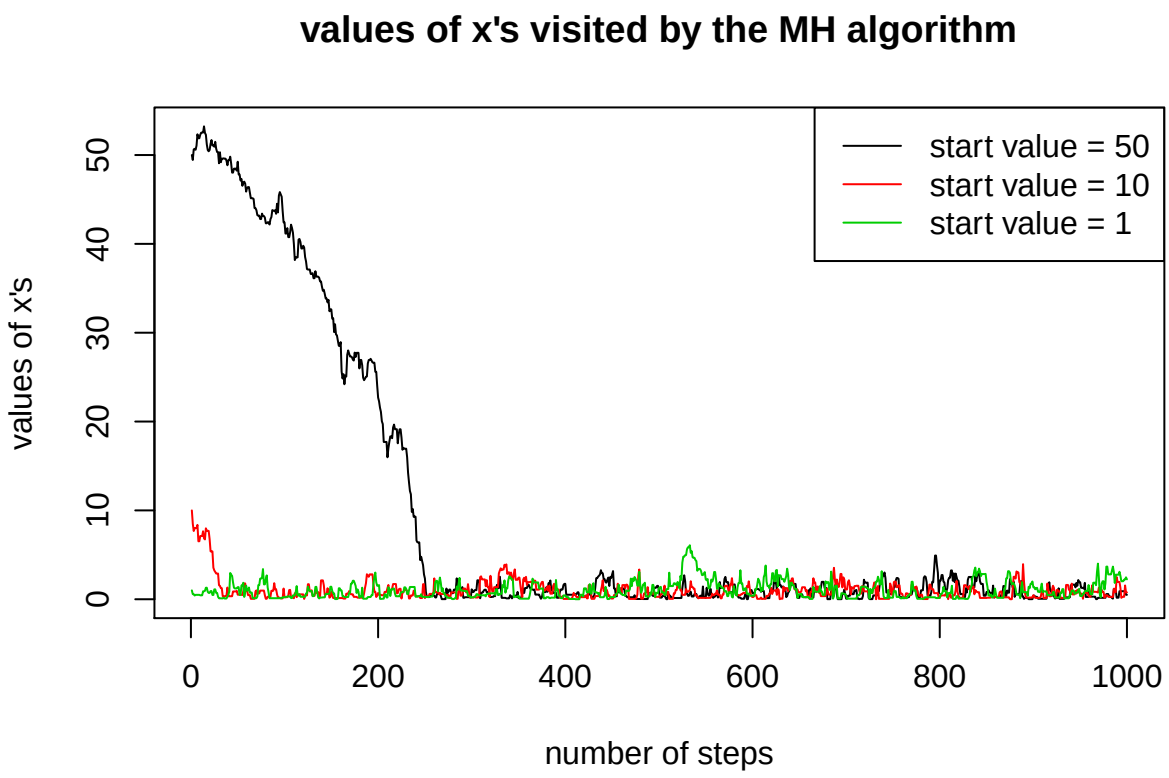
Exercise 1: sampling from exponential distribution

```
# target distribution: exponential distribution with mean 1
target = function(x){
  return( ifelse(x<0, 0, exp(-x)) )
}
```

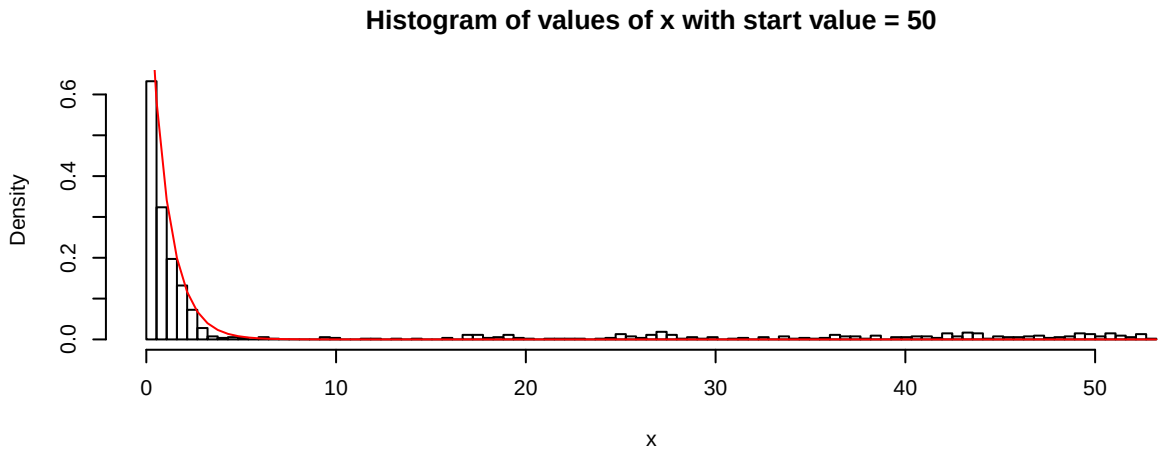
```
# the easyMCMC() function for sampling from target distribution
easyMCMC = function(niter, startval, proposalsd){
  x = rep(0,niter)
  x[1] = startval
  for(i in 2:niter){
    currentx = x[i-1]
    proposedx = rnorm(1,mean=currentx,sd=proposalsd)
    A = target(proposedx)/target(currentx)
    if(runif(1)<A){
      x[i] = proposedx       # accept move with probabily min(1,A)
    } else {
      x[i] = currentx        # otherwise "reject" move, and stay where we are
    }
  }
  return(x)
}
```

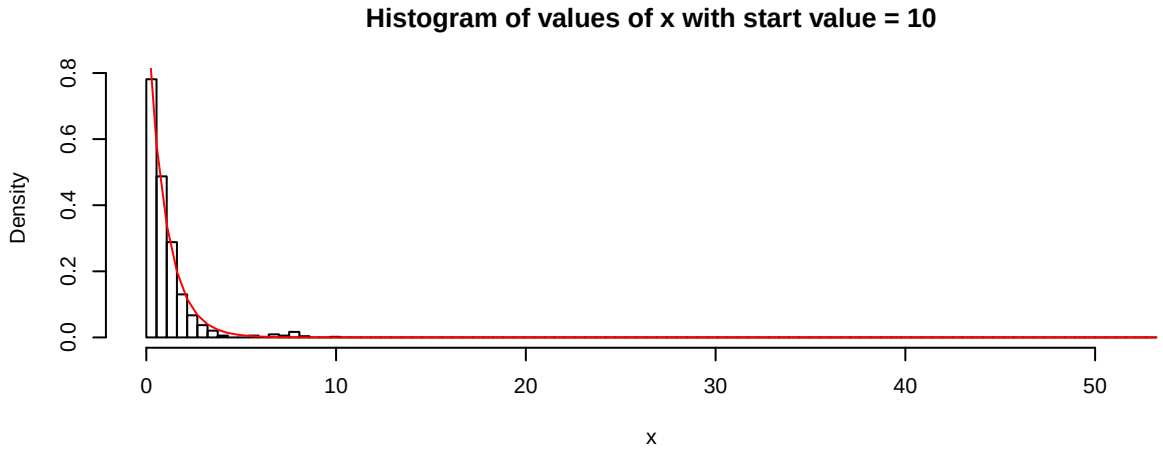
(a) different starting values

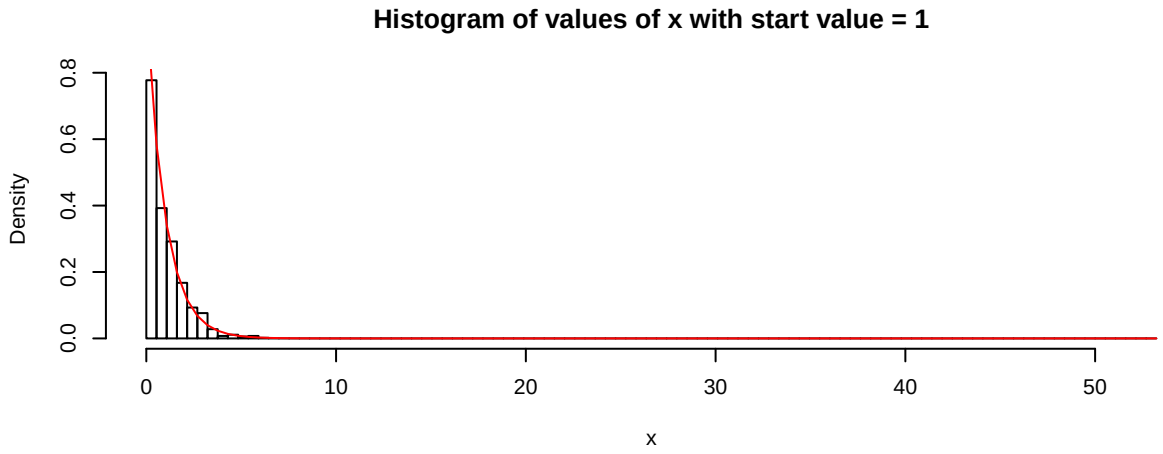
```
# try different starting values
set.seed(123)
z1=easyMCMC(1000, startval=50, proposalsd=1)
z2=easyMCMC(1000, startval=10, proposalsd=1)
z3=easyMCMC(1000, startval=1 , proposalsd=1)

# trace plot
maxz=max(c(z1,z2,z3))
plot(z1, ylim=c(0,maxz), type="l", main="values of x's visited by the MH algorithm",
     xlab="number of steps", ylab="values of x's")
lines(z2,col=2)
lines(z3,col=3)
legend("topright", legend=c("start value = 50","start value = 10","start value = 1"), col=c(1,2,3),lty=
```




```
# histograms
xx = seq(0,maxz,length=100)

par(mfcol=c(3,1))
hist(z1, breaks=seq(0,maxz,length=100), xlim=c(0,maxz), probability=TRUE,
     main="Histogram of values of x with start value = 50", xlab="x")
lines(xx,target(xx),col="red")
hist(z2, breaks=seq(0,maxz,length=100), xlim=c(0,maxz), probability=TRUE,
     main="Histogram of values of x with start value = 10", xlab="x")
lines(xx,target(xx),col="red")
hist(z3, breaks=seq(0,maxz,length=100), xlim=c(0,maxz), probability=TRUE,
     main="Histogram of values of x with start value = 1", xlab="x")
lines(xx,target(xx),col="red")
```







Comments:

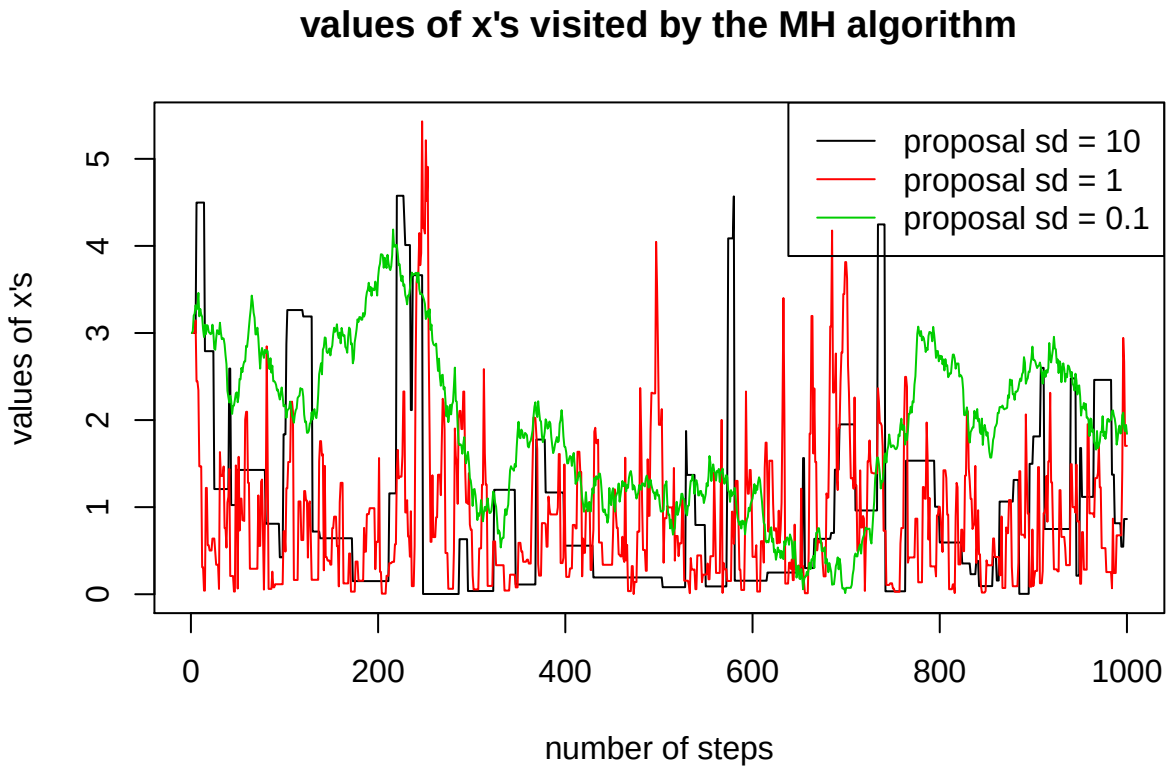
Here I run the MCMC scheme 3 times with different starting values: 1,10,50. The results show that the farther away the starting value is from the expectation of the target distribution $\pi(x) = \exp(-x)$, which equals to 1 here, the more steps it will take to converge to the target distribution and fluctuate around 1.

However, the results of histograms are "generally" similar with each other, which all roughly look like our target exponential distribution. Though when starting value is 50 there are more samples in the larger range of x values, it is because it takes more steps to converge thus needs larger number of steps to look more like our target distribution.

Therefore, to sum up, the starting values roughly do not affect the final MCMC scheme as long as the number of steps is "long enough". If the starting value is farther away from the expectation of the target distribution, it needs more steps to converge and needs larger number of steps to look more like the target distribution.

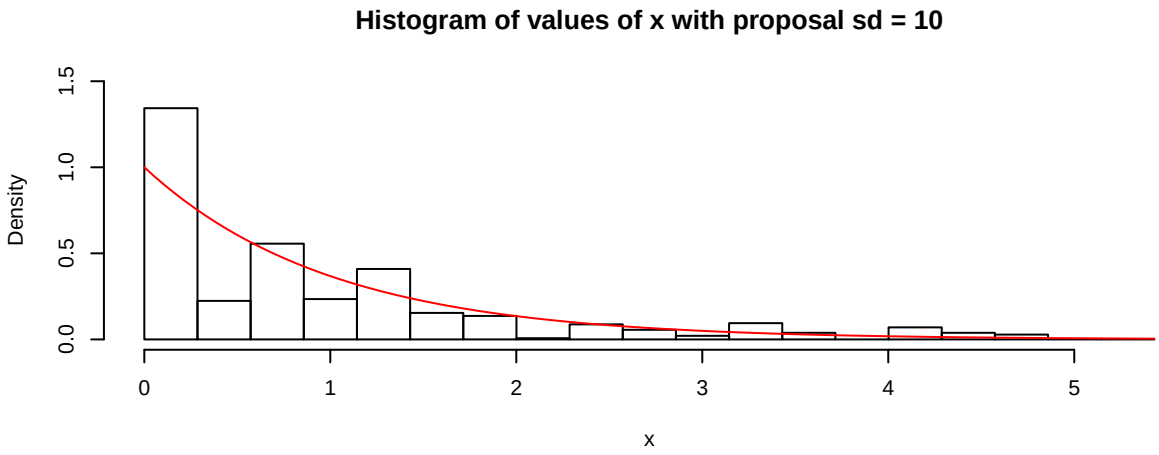
(b) bigger/smaller proposal standard deviation

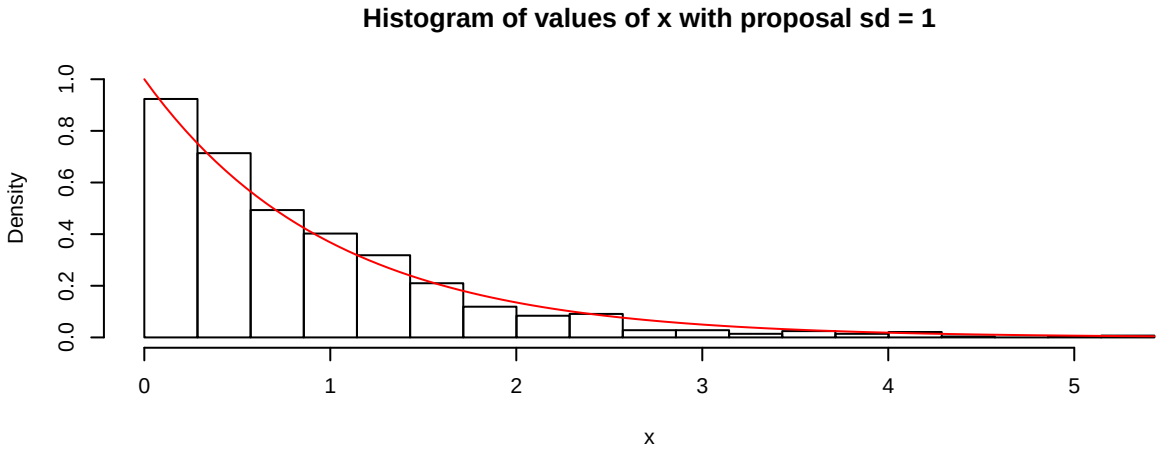
```
# try different proposal standard deviation
z1=easyMCMC(1000, startval=3, proposalsd=10)
z2=easyMCMC(1000, startval=3, proposalsd=1)
z3=easyMCMC(1000, startval=3, proposalsd=0.1)

# trace plot
maxz=max(c(z1,z2,z3))
plot(z1, ylim=c(0,maxz), type="l", main="values of x's visited by the MH algorithm",
     xlab="number of steps", ylab="values of x's")
lines(z2,col=2)
lines(z3,col=3)
legend("topright", legend=c("proposal sd = 10","proposal sd = 1","proposal sd = 0.1"), col=c(1,2,3),lty=
```

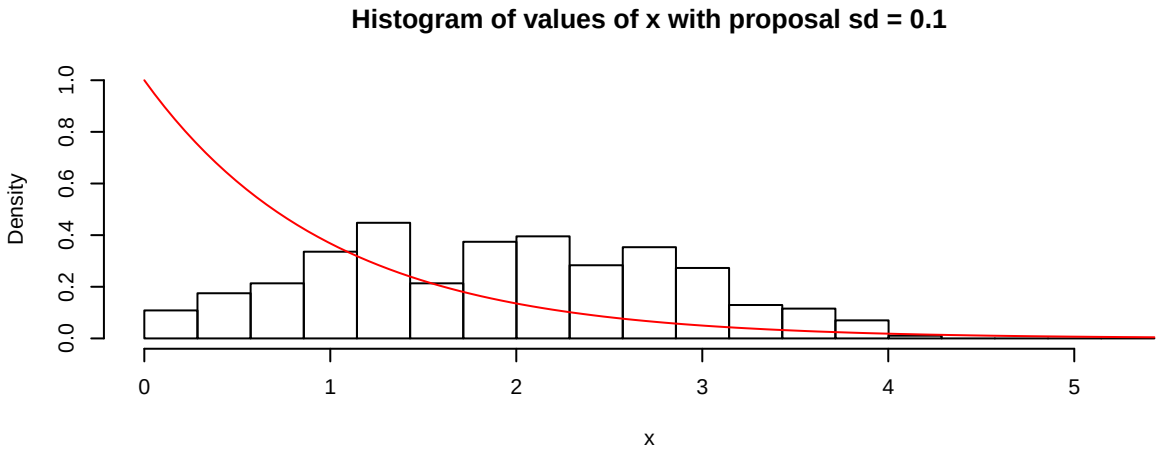



```
# histograms
xx = seq(0,maxz,length=100)
```

```
par(mfcol=c(3,1))
hist(z1, breaks=seq(0,maxz,length=20), xlim=c(0,maxz), ylim=c(0,1.5), probability=TRUE,
     main="Histogram of values of x with proposal sd = 10", xlab="x")
lines(xx,target(xx),col="red")
hist(z2, breaks=seq(0,maxz,length=20), xlim=c(0,maxz), ylim=c(0,1), probability=TRUE,
     main="Histogram of values of x with proposal sd = 1", xlab="x")
lines(xx,target(xx),col="red")
hist(z3, breaks=seq(0,maxz,length=20), xlim=c(0,maxz), ylim=c(0,1), probability=TRUE,
     main="Histogram of values of x with proposal sd = 0.1", xlab="x")
lines(xx,target(xx),col="red")
```







Comments:

Here I run the MCMC scheme 3 times with different proposal standard deviations: 10,1,0.1, we can see that the results are different.

In the trace plot, when the proposal standard deviation is bigger, the trace of x values looks like a piece-wise

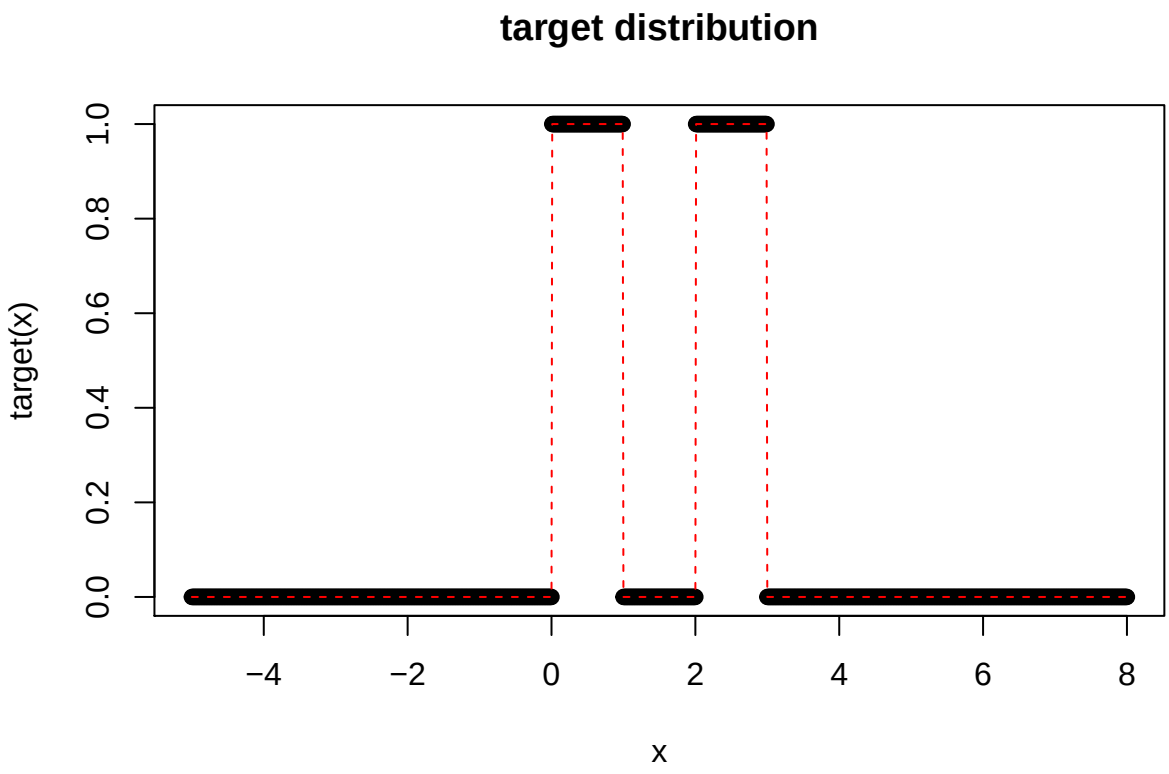
constant function. It is bacause now it takes wider steps (bigger changes), thus when $\pi(y) > \pi(x_t)$, it is more likely to reject the move and stay at x_t. On the other hand, when the proposal standard deviation is smaller, it takes narrower steps (smaller changes), so it is more likely to accept the move and move to y.

In the histogram plot, in both cases where the proposal standard deviation is bigger or smaller, the distribution of x values do not look like our target distribution. Therefore, it is crucial to use appropriate proposal standard deviation for the random walk proposal Q.

(c) changing the target function

```
# new target function
target = function(x){
  return((x>0 & x <1) + (x>2 & x<3))  # return 1 if 0<x<1 or 2<x<3, otherwise 0
}
```

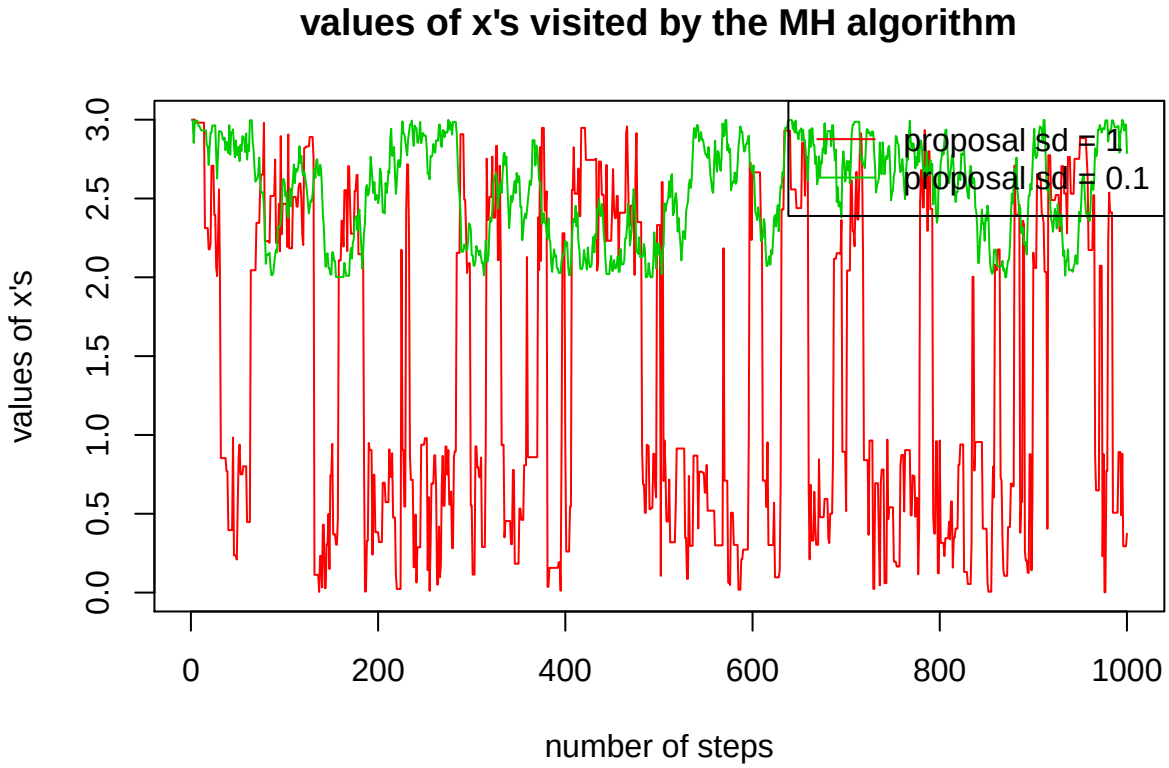
```
# plot the target distribution
x = seq(-5,8,0.01)
plot(x,target(x), main="target distribution")
lines(x,target(x), col="red", lty=2)
```



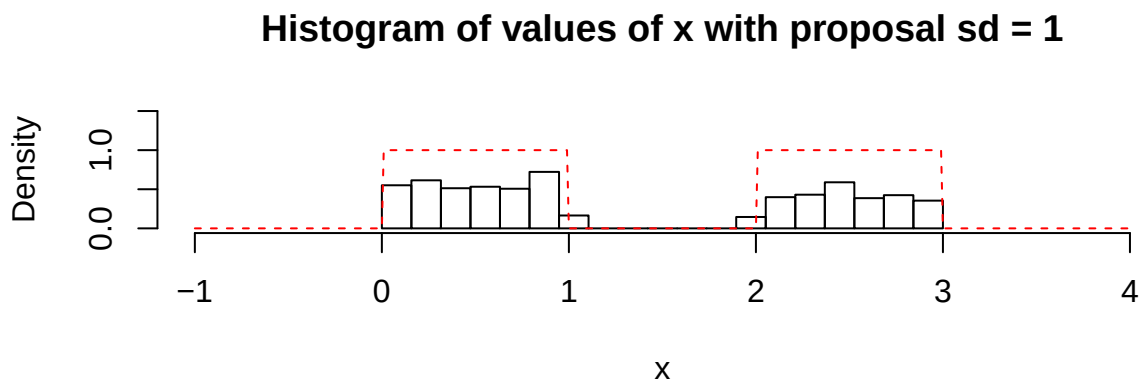

```
# adjust the easyMCMC() function for sampling from target distribution
easyMCMC = function(niter, startval, proposalsd){
  x = rep(0,niter)
  x[1] = startval
  for(i in 2:niter){
    currentx = x[i-1]
    proposedx = rnorm(1,mean=currentx,sd=proposalsd)
```

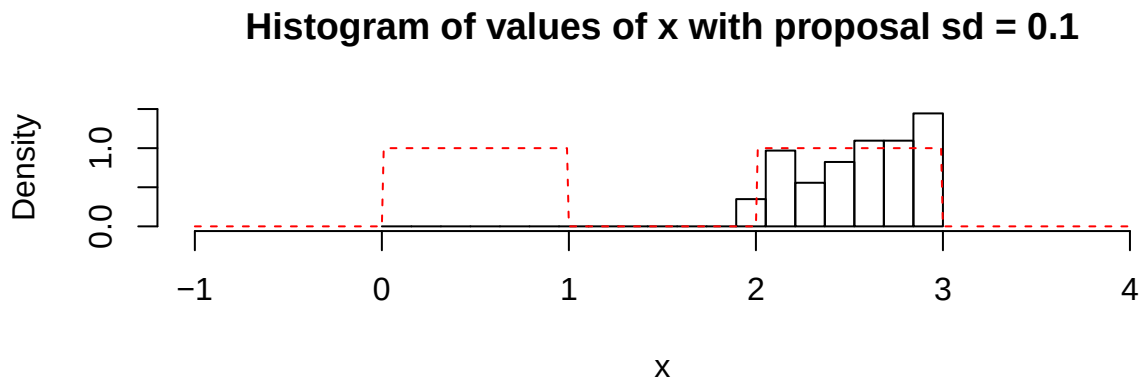
```
    if(target(currentx)==0){
        A = target(proposedx)/10^(-5)    # avoid zero on the denominator
    }else{
        A = target(proposedx)/target(currentx)
    }

    if(runif(1)<A){
      x[i] = proposedx        # accept move with probabily min(1,A)
    } else {
      x[i] = currentx         # otherwise "reject" move, and stay where we are
    }
  }
  return(x)
}
```

```
# try different proposal standard deviation
z1=easyMCMC(1000,startval=3, proposalsd=1)
z2=easyMCMC(1000,startval=3, proposalsd=0.1)

# trace plot
maxz=max(c(z1,z2))
plot(z1, ylim=c(0,maxz), type="l", main="values of x's visited by the MH algorithm",
     col=2, xlab="number of steps", ylab="values of x's")
lines(z2,col=3)
legend("topright", legend=c("proposal sd = 1","proposal sd = 0.1"), col=c(2,3),lty=1)
```



```r
# histograms
xx = seq(-1,4,0.01)

par(mfcol=c(2,1))
hist(z1, breaks=seq(0,maxz,length=20), xlim=c(-1,4), ylim=c(0,1.5), probability=TRUE,
     main="Histogram of values of x with proposal sd = 1", xlab="x")
lines(xx,target(xx),col="red", lty=2)
hist(z2, breaks=seq(0,maxz,length=20), xlim=c(-1,4), ylim=c(0,1.5), probability=TRUE,
     main="Histogram of values of x with proposal sd = 0.1", xlab="x")
lines(xx,target(xx),col="red", lty=2)
```





Comments:

Here I adjust the easyMCMC function to avoid zeros on the denominators $\pi(x_t)$ since that this target distribution can take zero values.

Then I run the MCMC scheme 2 times with different proposal standard deviations: 1 and 0.1, we can see that the results are different.

In the trace plot, when the proposal standard deviation is 1, the x values fluctuate between 0 and 3. However, when the proposal standard deviation is 0.1, which is small, the x values fluctuate only between 3 and 2. It is becasue in this case, $\pi(y)$ is always equal to 0 outside the interval $(2, 3)$, so it cannot move outside this interval and jump to the other interval $(0, 1)$. Note that our starting values are 3 here for both MCMC schemes.

In the histogram plot, when the proposal standard deviation is 1, it looks more like our target distribution. However, when the proposal standard deviation is 0.1, the target distribution in the range of $0 < x < 1$ is missing.

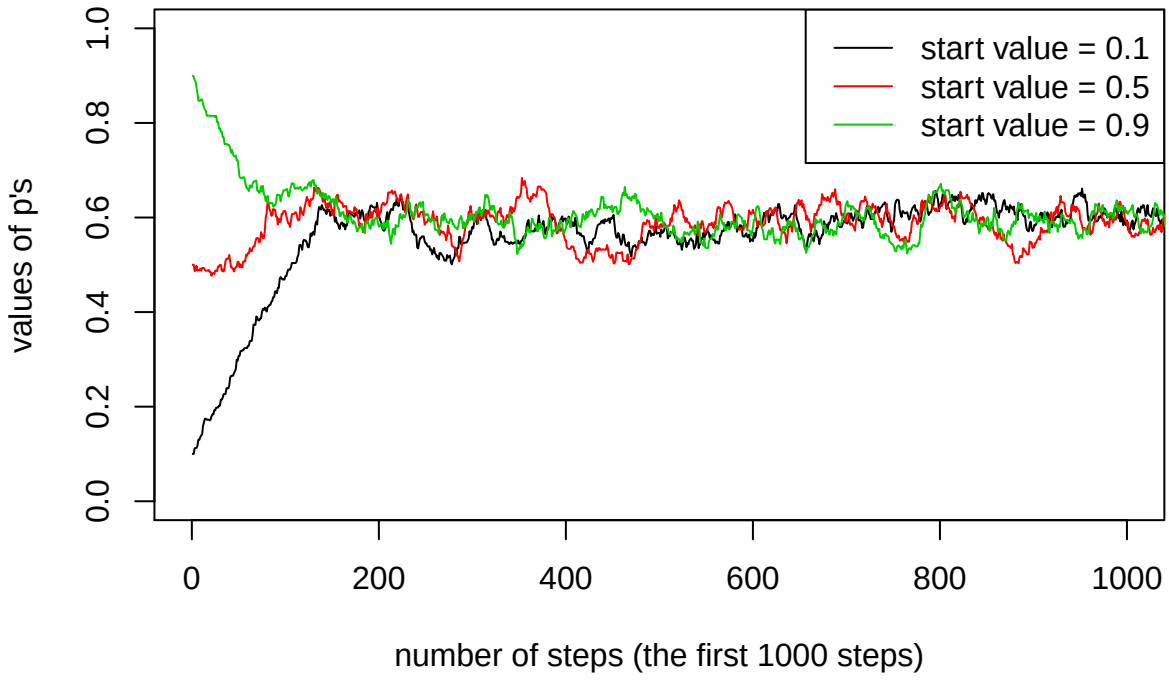
Therefore, when the the proposal standard deviation is small, the MCMC simulation takes small steps and may get stuck in a local distribution near the starting value and cannot jump to another interval.

Exercise 2: estimating an allele frequency

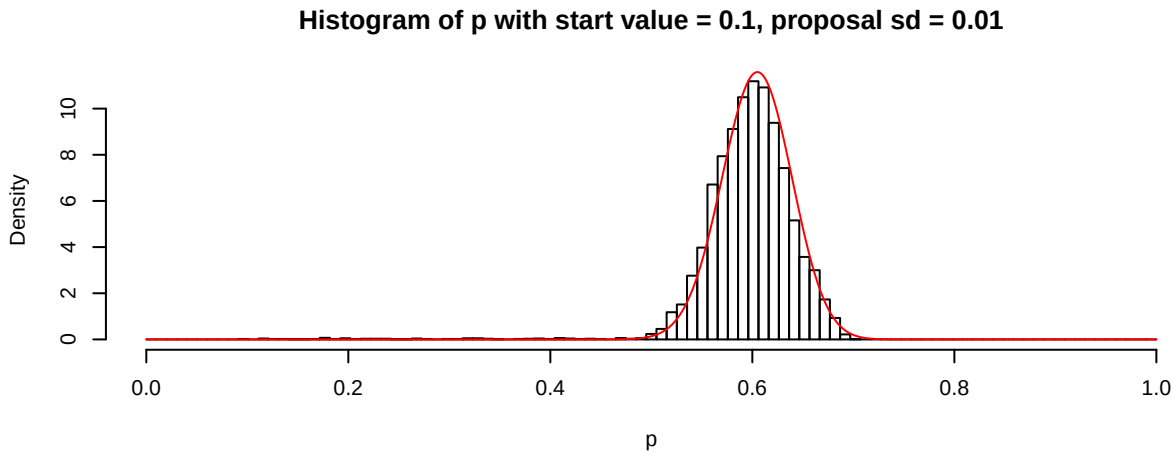
```
# prior: uniform on [0,1]
prior = function(p){
  if((p<0) || (p>1)){     # || here means "or"
    return(0)}
  else{
    return(1)}
}

# likelihood: Hardy Weinberg Equilibrium
likelihood = function(p, nAA, nAa, naa){
  return( p^(2*nAA) * (2*p*(1-p))^nAa * (1-p)^(2*naa) )
}

# MH sampler: sample from posterior distribution
psampler = function(nAA, nAa, naa, niter, pstartval, pproposalsd){
  p = rep(0,niter)
  p[1] = pstartval
  for(i in 2:niter){
    currentp = p[i-1]
    newp = currentp + rnorm(1,0,pproposalsd)
    A = prior(newp)*likelihood(newp,nAA,nAa,naa)/(prior(currentp)*likelihood(currentp,nAA,nAa,naa))
    if(runif(1)<A){
      p[i] = newp       # accept move with probabily min(1,A)
    } else {
      p[i] = currentp   # otherwise "reject" move, and stay where we are
    }
  }
  return(p)
}
```

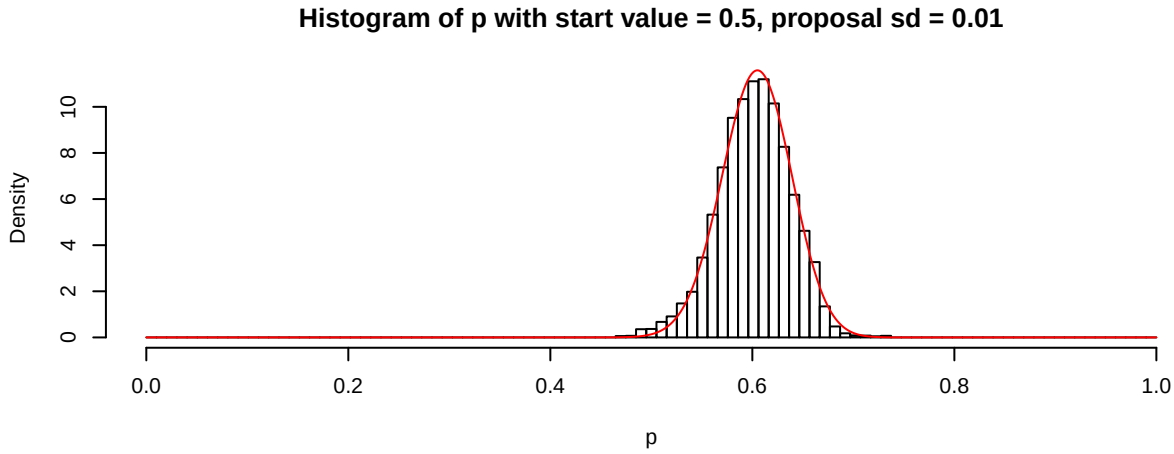
(a) different starting values

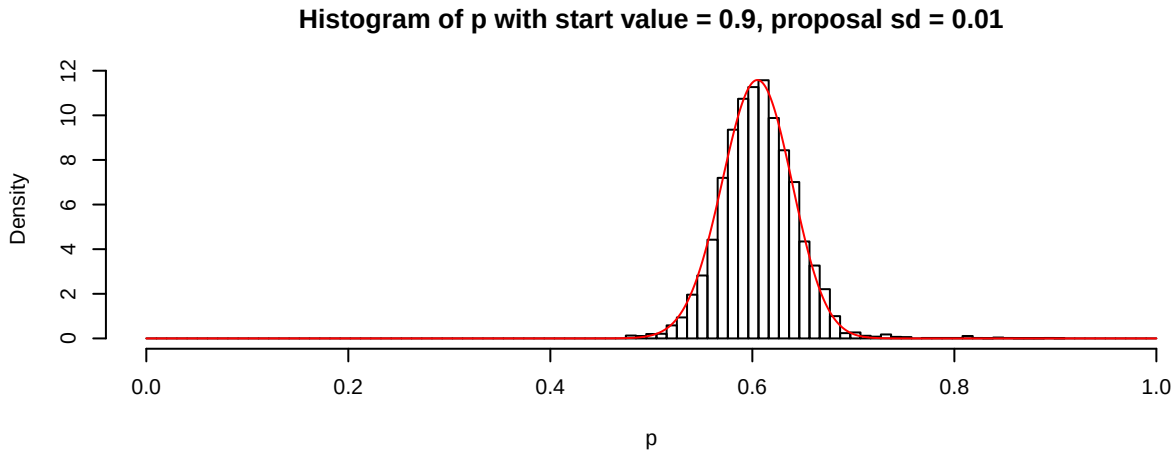
```
# try different starting values
set.seed(123)
z1=psampler(50,21,29,10000, pstartval=0.1, pproposalsd=0.01)
z2=psampler(50,21,29,10000, pstartval=0.5, pproposalsd=0.01)
z3=psampler(50,21,29,10000, pstartval=0.9, pproposalsd=0.01)

# trace plot
plot(z1, xlim=c(0,1000), ylim=c(0,1), type="l", main="values of p's visited by the MH algorithm, proposa
lines(z2,col=2)
lines(z3,col=3)
legend("topright", legend=c("start value = 0.1","start value = 0.5","start value = 0.9"), col=c(1,2,3),l
```




```
# histograms
x = seq(0,1,length=1000)

par(mfcol=c(3,1))
hist(z1, breaks=seq(0,1,length=100), xlim=c(0,1), probability=TRUE,
     main="Histogram of p with start value = 0.1, proposal sd = 0.01", xlab="p")
lines(x,dbeta(x,122, 80),col="red")
hist(z2, breaks=seq(0,1,length=100), xlim=c(0,1), probability=TRUE,
     main="Histogram of p with start value = 0.5, proposal sd = 0.01", xlab="p")
lines(x,dbeta(x,122, 80),col="red")
hist(z3, breaks=seq(0,1,length=100), xlim=c(0,1), probability=TRUE,
     main="Histogram of p with start value = 0.9, proposal sd = 0.01", xlab="p")
lines(x,dbeta(x,122, 80),col="red")
```

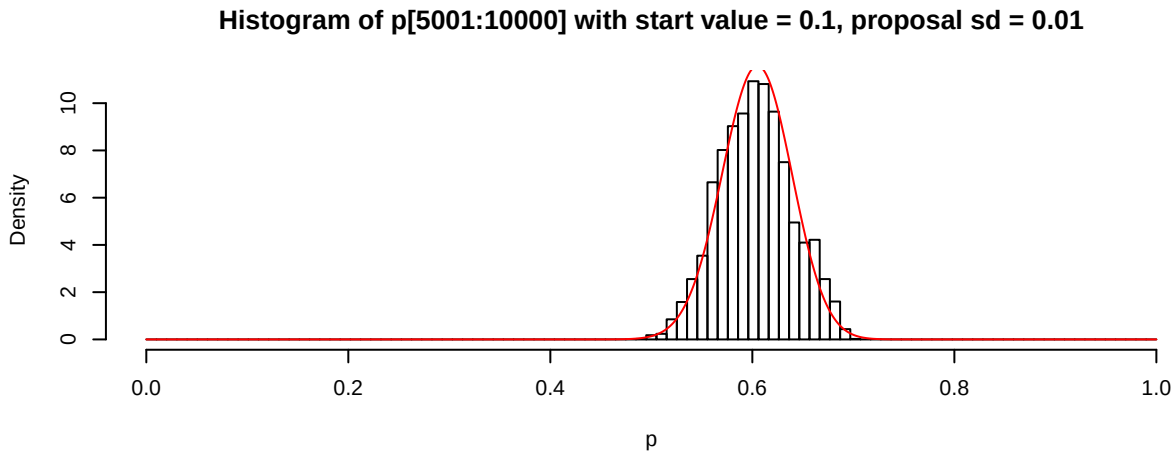


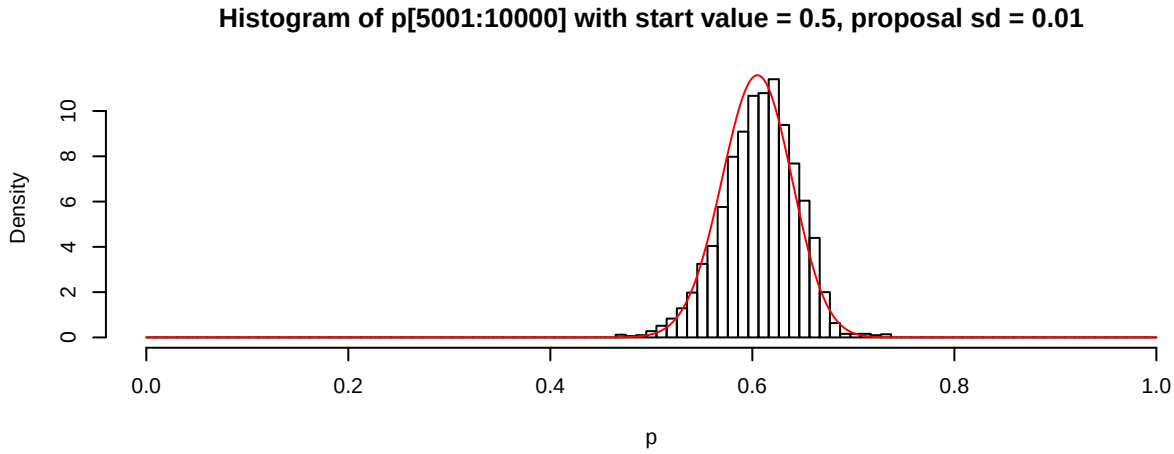


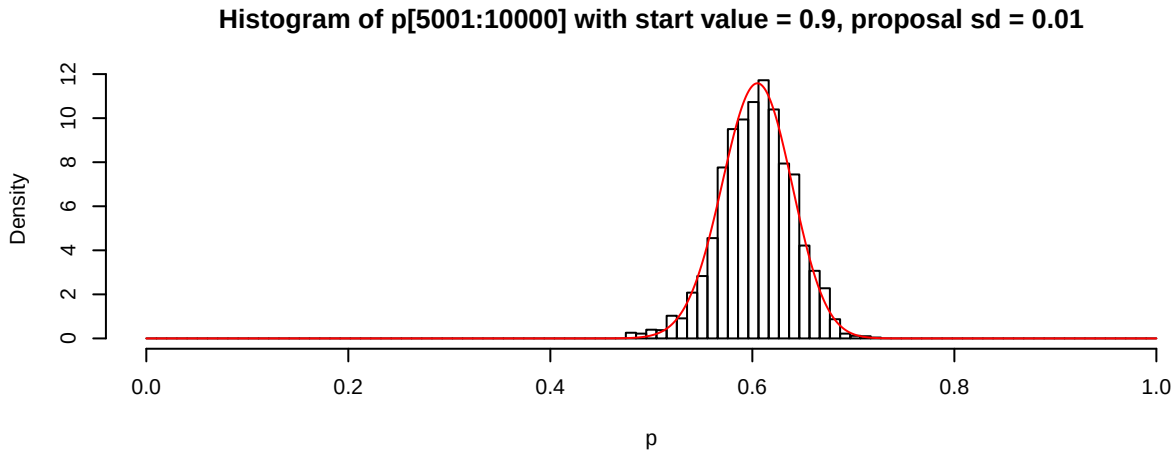



```r
# histograms of the last 5000 p's
x = seq(0,1,length=1000)

par(mfcol=c(3,1))
hist(z1[5001:10000], breaks=seq(0,1,length=100), xlim=c(0,1), probability=TRUE,
```

```
     main="Histogram of p[5001:10000] with start value = 0.1, proposal sd = 0.01", xlab="p")
lines(x,dbeta(x,122, 80),col="red")
hist(z2[5001:10000], breaks=seq(0,1,length=100), xlim=c(0,1), probability=TRUE,
     main="Histogram of p[5001:10000] with start value = 0.5, proposal sd = 0.01", xlab="p")
lines(x,dbeta(x,122, 80),col="red")
hist(z3[5001:10000], breaks=seq(0,1,length=100), xlim=c(0,1), probability=TRUE,
     main="Histogram of p[5001:10000] with start value = 0.9, proposal sd = 0.01", xlab="p")
lines(x,dbeta(x,122, 80),col="red")
```







Comments:

In this example, the theoretical posterior distribution of p is $Beta(122, 80)$, and its posterior mean is $\frac{122}{122+80} \approx 0.6$.

Here I run the MCMC scheme 3 times with different starting values: 0.1,0.5,0.9. The results show that the

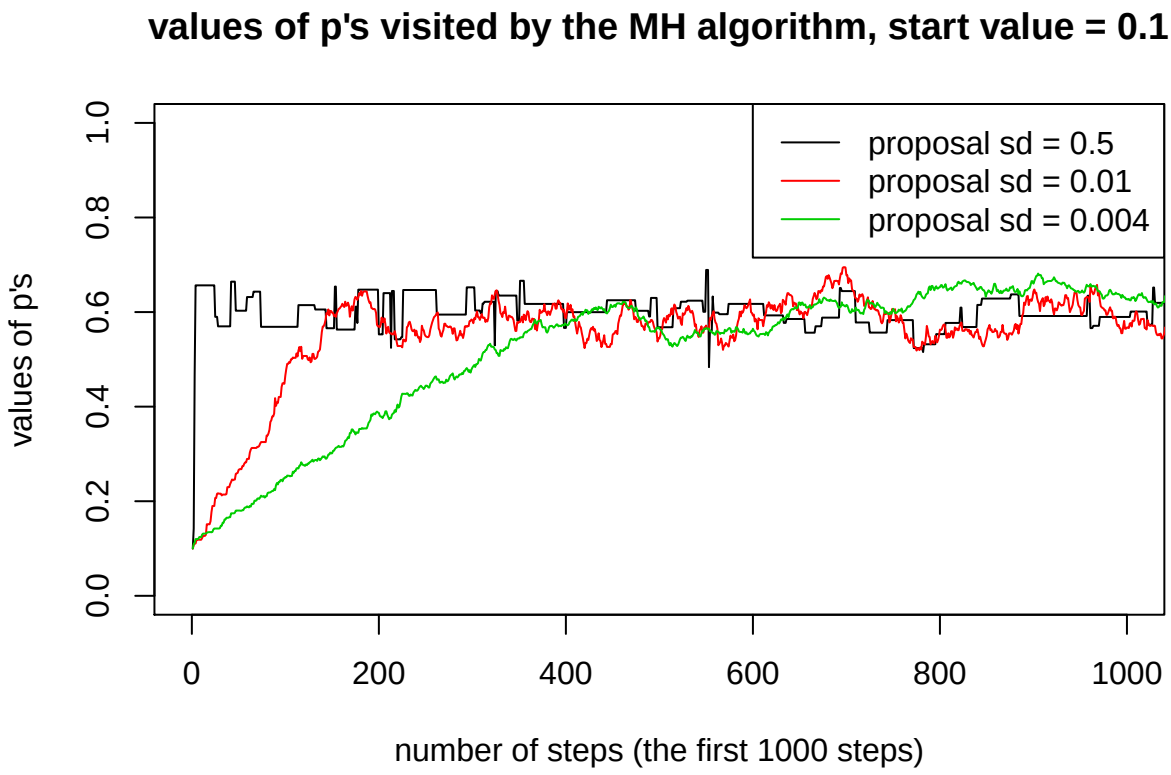
farther away the starting value is from the posterior mean 0.6, the more steps it will take to converge to the target posterior distribution and fluctuate around 0.6.

However, the results of histograms are "generally" similar with each other, which all roughly look like our target Beta posterior distribution. Though for starting value of 0.1, there are more samples in the left tail, and for starting value of 0.9, there are more samples in the right tail, it is because it takes more steps to converge thus needs larger number of steps to look more like our target posterior distribution. When discarding the first 5000 p's as "burnin", these tails are not present any more and the three distributions are more similar with each other, which all look like the target posterior distribution.

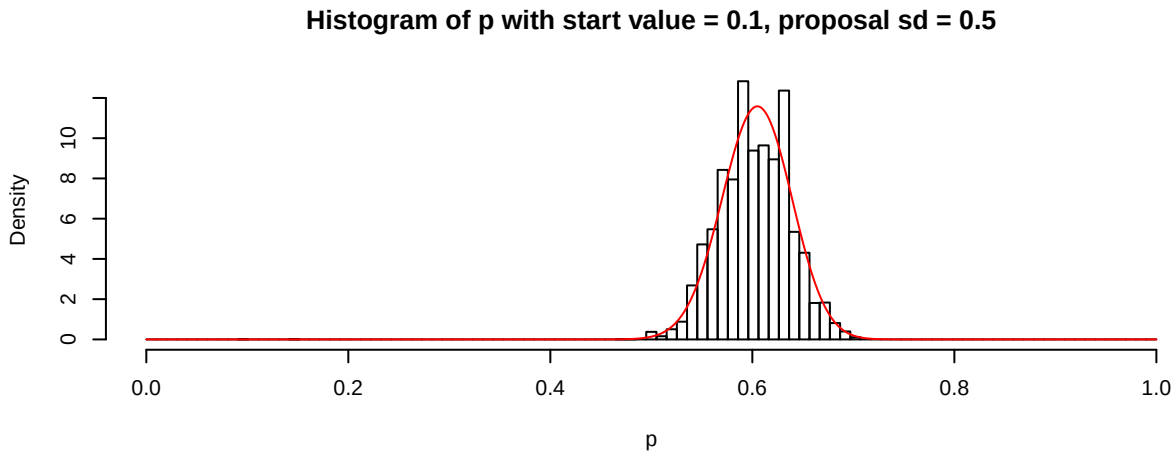
Therefore, to sum up, the starting values roughly do not affect the final MCMC scheme as long as the number of steps is "long enough". If the starting value is farther away from the posterior mean of the target posterior distribution, it needs more steps to converge and needs larger number of steps to look more like the target posterior distribution.

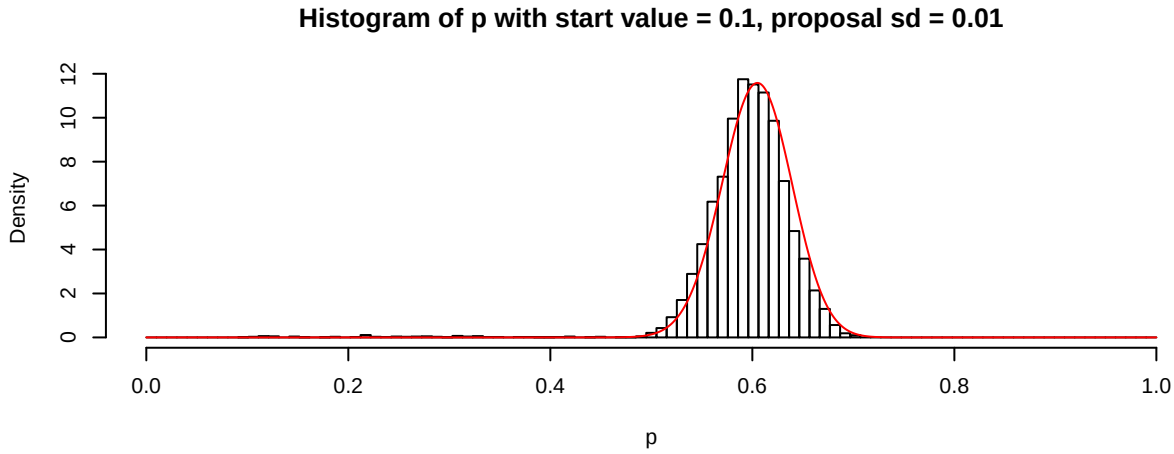
(b) bigger/smaller proposal standard deviation

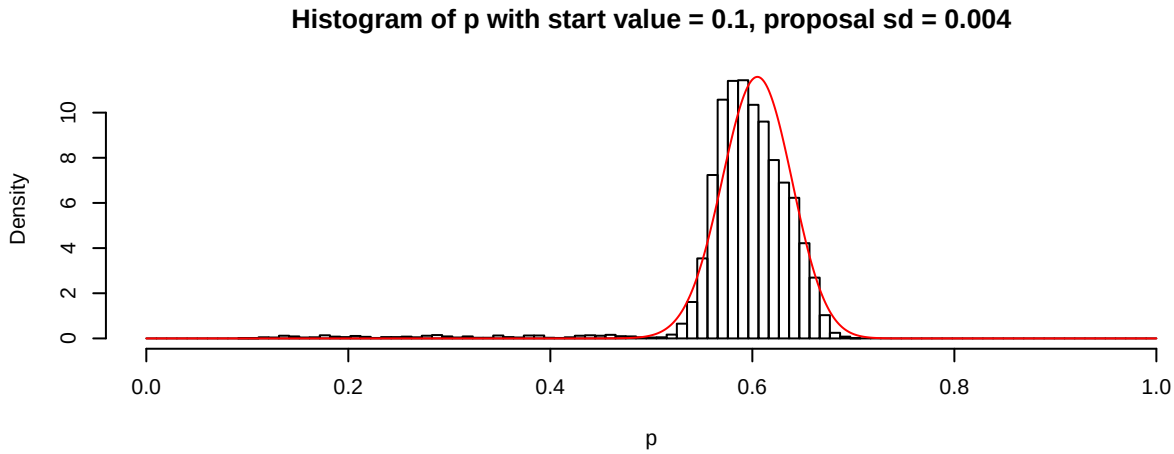
```
# try different proposal standard deviation
z1=psampler(50,21,29,10000, pstartval=0.1, pproposalsd=0.5)
z2=psampler(50,21,29,10000, pstartval=0.1, pproposalsd=0.01)
z3=psampler(50,21,29,10000, pstartval=0.1, pproposalsd=0.004)

# trace plot
plot(z1, xlim=c(0,1000), ylim=c(0,1), type="l", main="values of p's visited by the MH algorithm, start v
lines(z2,col=2)
lines(z3,col=3)
legend("topright", legend=c("proposal sd = 0.5","proposal sd = 0.01","proposal sd = 0.004"), col=c(1,2,3
```



```r
# histograms
x = seq(0,1,length=1000)

par(mfcol=c(3,1))
hist(z1, breaks=seq(0,1,length=100), xlim=c(0,1), probability=TRUE,
     main="Histogram of p with start value = 0.1, proposal sd = 0.5", xlab="p")
lines(x,dbeta(x,122, 80),col="red")
hist(z2, breaks=seq(0,1,length=100), xlim=c(0,1), probability=TRUE,
     main="Histogram of p with start value = 0.1, proposal sd = 0.01", xlab="p")
lines(x,dbeta(x,122, 80),col="red")
hist(z3, breaks=seq(0,1,length=100), xlim=c(0,1), probability=TRUE,
     main="Histogram of p with start value = 0.1, proposal sd = 0.004", xlab="p")
lines(x,dbeta(x,122, 80),col="red")
```

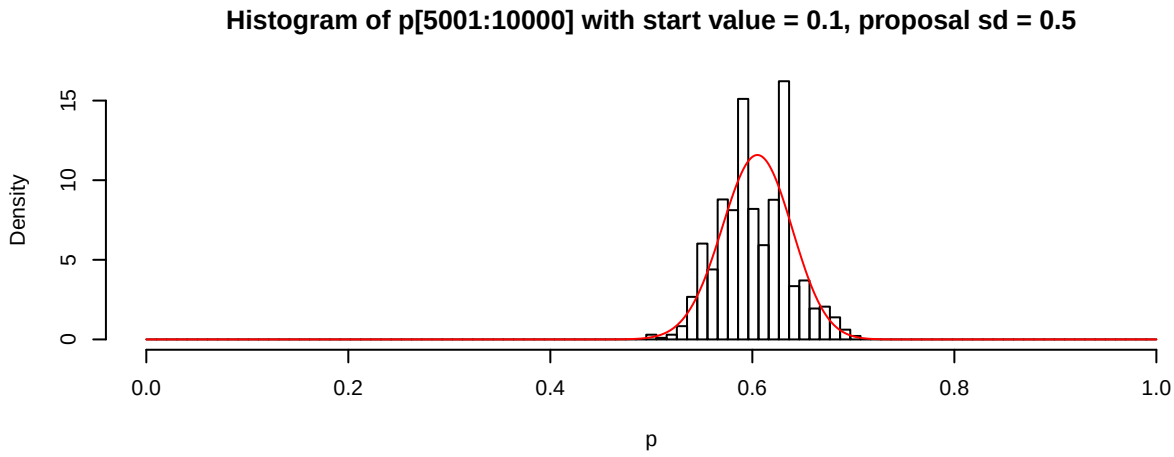


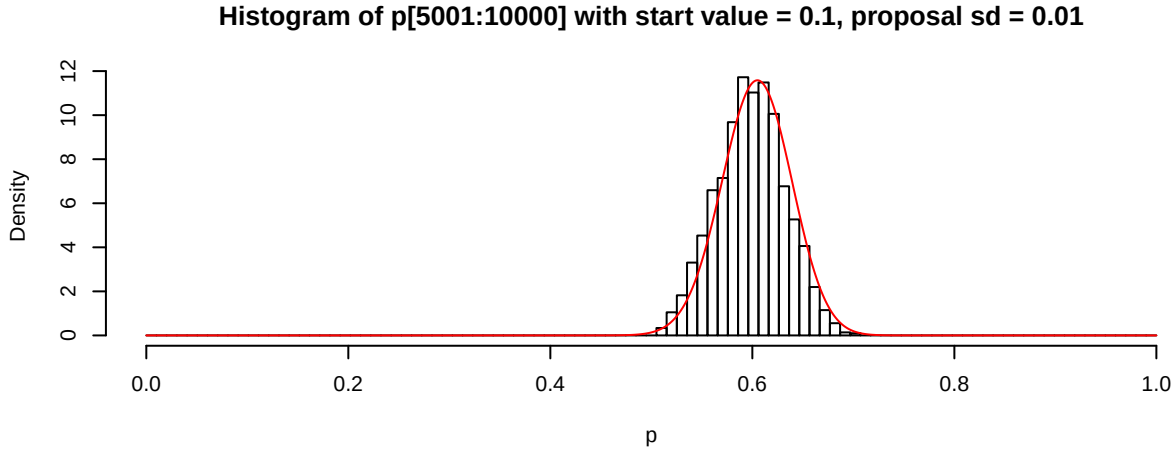


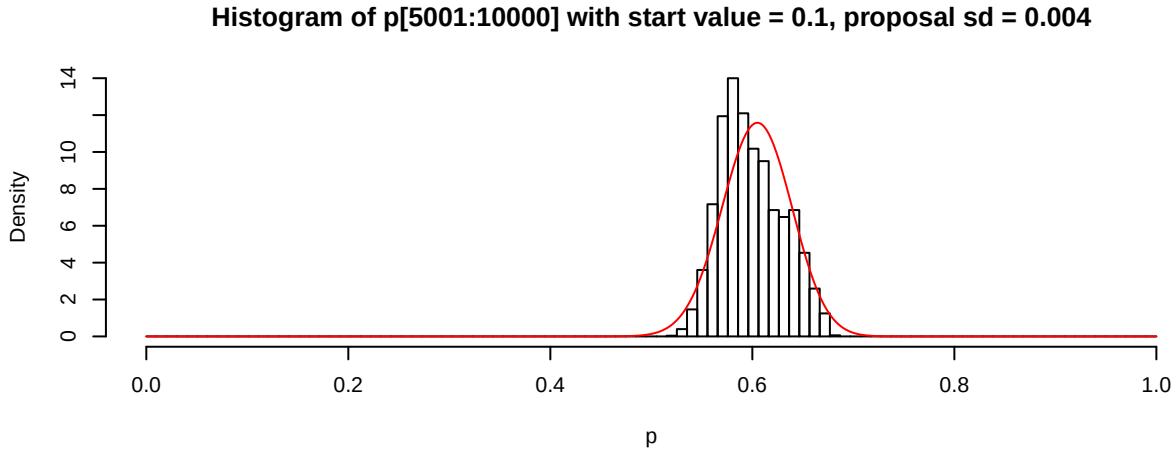



```r
# histograms of the last 5000 p's
x = seq(0,1,length=1000)

par(mfcol=c(3,1))
hist(z1[5001:10000], breaks=seq(0,1,length=100), xlim=c(0,1), probability=TRUE,
```

```
     main="Histogram of p[5001:10000] with start value = 0.1, proposal sd = 0.5", xlab="p")
lines(x,dbeta(x,122, 80),col="red")
hist(z2[5001:10000], breaks=seq(0,1,length=100), xlim=c(0,1), probability=TRUE,
     main="Histogram of p[5001:10000] with start value = 0.1, proposal sd = 0.01", xlab="p")
lines(x,dbeta(x,122, 80),col="red")
hist(z3[5001:10000], breaks=seq(0,1,length=100), xlim=c(0,1), probability=TRUE,
     main="Histogram of p[5001:10000] with start value = 0.1, proposal sd = 0.004", xlab="p")
lines(x,dbeta(x,122, 80),col="red")
```







Comments:

Here I run the MCMC scheme 3 times with different proposal standard deviations: 0.5,0.01,0.004, but all with the same start value of 0.1. In the previous (a) part, using 0.1 as start value takes longer time to converge, so I pick this start value to see how the proposal standard deviation will affect the convergence of the algorithm.

Here we can see that the results are different.

In the trace plot, when the proposal standard deviation is bigger (0.5), the trace of x values looks like a piece-wise constant function. It is bacause now it takes wider steps (bigger changes), thus when $\pi(y) > \pi(x_t)$, it is more likely to reject the move and stay at x_t. On the other hand, when the proposal standard deviation is smaller (0.01 or 0.004), it takes narrower steps (smaller changes), so it is more likely to accept the move and move to y.

Also in the trace plot, we see that the bigger the proposal standard deviation is, the faster the algorithm converges to the target posterior distribution of p and starts to fluctuate around the posterior mean 0.6.

In the histogram plot, in both cases where the proposal standard deviation is bigger (0.5) or smaller (0.004), the distribution of p values do not look like our target posterior Beta distribution, even when we discard the first 5000 p's as "burnin". Therefore, it is crucial to use appropriate proposal standard deviation for the random walk proposal Q.

Exercise 3: estimating an allele frequency & inbreeding coevcient

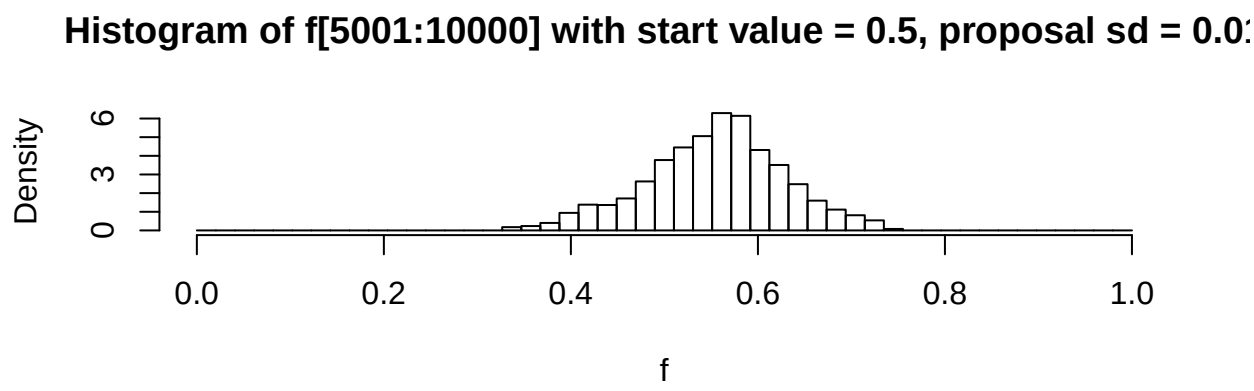
(a) MH sampler to sample from the joint distribution of f and p

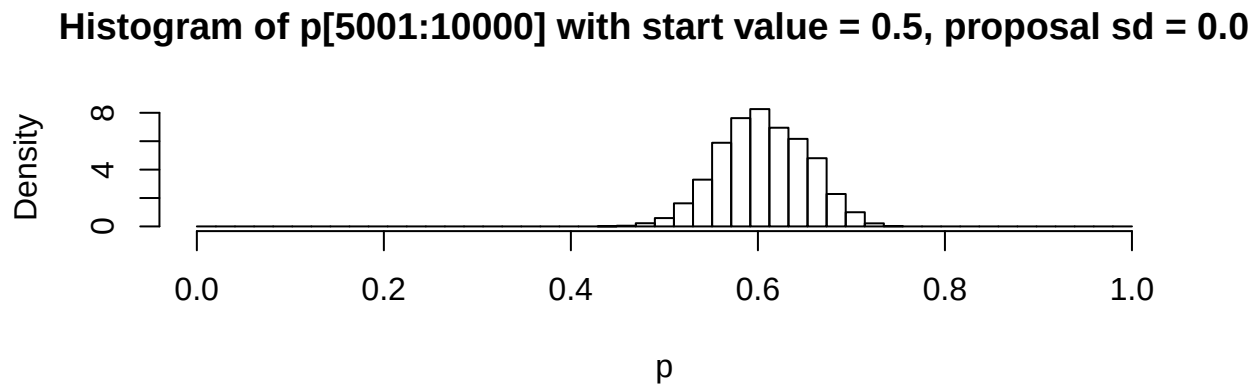
```
# prior: both f and p are uniform on [0,1]
prior = function(x){
  if((x<0) || (x>1)){     # || here means "or"
    return(0)}
  else{
    return(1)}
}

# likelihood
likelihood = function(f, p, nAA, nAa, naa){
  return( (f*p+(1-f)*p^2)^nAA * ((1-f)*2*p*(1-p))^nAa * (f*(1-p)+(1-f)*(1-p)^2)^naa )
}

# MH sampler: sample from the joint distribution of f and p
fpsampler = function(nAA, nAa, naa, niter, fstartval, pstartval, fproposalsd, pproposalsd){
  f = rep(0,niter)
  p = rep(0,niter)
  f[1] = fstartval
  p[1] = pstartval
  for(i in 2:niter){
    currentf = f[i-1]
    currentp = p[i-1]
    newf = currentf + rnorm(1,0,fproposalsd)
    newp = currentp + rnorm(1,0,pproposalsd)

    A =  prior(newf)*prior(newp)*likelihood(newf,newp,nAA,nAa,naa)/( prior(currentf)*prior(currentp)*li

    if(runif(1)<A){
      f[i] = newf
      p[i] = newp       # accept move with probabily min(1,A)
    } else {
      f[i] = currentf
      p[i] = currentp   # otherwise "reject" move, and stay where we are
    }
  }
```

```
  return(list(f=f, p=p))  # return a "list" with two elements named f and p
}
```

(b) point estimates and interval estimates for both f and p

```
set.seed(123)
z = fpsampler(50,21,29,10000,0.5,0.5,0.01,0.01)
f = z$f[5001:10000]
p = z$p[5001:10000]

# histograms of the last 5000 f's and p's
par(mfrow=c(2,1))
hist(f, breaks=seq(0,1,length=50), xlim=c(0,1), probability=TRUE,
     main="Histogram of f[5001:10000] with start value = 0.5, proposal sd = 0.01", xlab="f")
hist(p, breaks=seq(0,1,length=50), xlim=c(0,1), probability=TRUE,
     main="Histogram of p[5001:10000] with start value = 0.5, proposal sd = 0.01", xlab="p")
```






```
# posterior means
mean(f)
```

```
## [1] 0.5542239
```

```r
mean(p)
```

```
## [1] 0.6057525
```

```r
# 90% posterior credible intervals
quantile(f, probs=c(0.05,0.95))
```

```
##        5%       95%
## 0.4196399 0.6753719
```

```r
quantile(p, probs=c(0.05,0.95))
```

```
##        5%       95%
## 0.5302968 0.6819473
```

Comments:

According to the previous exercise, here in this question I used the starting values are 0.5 and the proposal standard deviations are 0.01 for both f and p.

Also, to get the target posterior distributions that the algorithms converge to, I discarded the first 5000 f's and p's as "burnin", and only used the last 5000 samples for both parameters.

The point estimation, i.e. posterior mean, of f is about 0.5542239, and its interval estimtes, i.e. 90% posterior credible interval, is about (0.4196399, 0.6753719).

The point estimation, i.e. posterior mean, of f is about 0.6057525, and its interval estimtes, i.e. 90% posterior credible interval, is about (0.5302968, 0.6819473).

Exercise 4: Gibbs Sampling

(1) sample z from $p(z|g,f,p)$

$$p(z_i=1|g_i,f,p)=\frac{p(g_i|z_i=1)\times f}{p(g_i|z_i=1)\times f+p(g_i|z_i=0)\times(1-f)}$$
$$p(z_i=0|g_i,f,p)=\frac{p(g_i|z_i=0)\times(1-f)}{p(g_i|z_i=1)\times f+p(g_i|z_i=0)\times(1-f)}$$

(2) sample f,p from $p(f,p|g,z)$

$$p(f,p|g,z)\ \propto\ p(f,p,g,z)\ =\prod_{i=1}^{n}p(f,p,g_i,z_i)\ =\ p(f)\ p(p)\ \prod_{i=1}^{n}p(z_i|f)\ p(g_i|z_i,p)$$
$$=\left(p(f)\prod_{i=1}^{n}p(z_i|f)\right)\left(p(p)\prod_{i=1}^{n}p(g_i|z_i,p)\right)$$

We can also see it as:

$$p(f,p|g,z)=p(f|g,z)\ \ p(p|g,z)=p(f|z)\ \ p(p|g,z)$$

Thus we can sample f,p from $p(f,p|g,z)$ by two steps: sample f from $p(f|z)$ and sample p from $p(p|g,z)$. Now we try to derive these two distributions.

Since:

$$p(f|z)\ \propto\ p(f)\prod_{i=1}^{n}p(z_i|f),\quad where\ \ f\sim Uniform(0,1)\ \ and\ \ z_i|f\sim Bernoulli(f)$$

So we can get that:

$$f|z \ \sim \ Beta(1+\sum_{i=1}^{n} z_i, \ 1+n-\sum_{i=1}^{n} z_i)$$

Also, since:

$$p(p|g,z) \ \propto \ p(p)\prod_{i=1}^{n} p(g_i|z_i,p), \quad where \ \ p \sim Uniform(0,1)$$

and the $p(g_i|z_i,p)$ is given as below:

$$p(g_i = AA|z_i = 1) = p; \qquad p(g_i = AA|z_i = 0) = p^2$$
$$p(g_i = Aa|z_i = 1) = 0; \qquad p(g_i = Aa|z_i = 0) = 2p(1-p)$$
$$p(g_i = aa|z_i = 1) = 1-p; \qquad p(g_i = aa|z_i = 0) = (1-p)^2$$

So we can get that:

$$p|g,z \sim Beta(1+nAA_1+2\times nAA_0+nAa_0, \quad 1+naa_1+2\times naa_0+nAa_1)$$

where the subscripts 0 or 1 indicates the number of genotypes when $z_i = 0$ or 1.

Therefore, the Gibbs sampler code is shown below:

```
# sample z from p(z|g,f,p)
sample_z = function(g,f,p){
    z = rep(0,length(g))
    for(i in 1:n){
        if(g[i]=="AA"){z[i]=rbinom(1,1,f*p/(f*p+(1-f)*p^2))}
        if(g[i]=="Aa"){z[i]=0}
        if(g[i]=="aa"){z[i]=rbinom(1,1,f*(1-p)/(f*(1-p)+(1-f)*(1-p)^2))}
    }
    return(z)
}

# sample f from p(f|z)
sample_f = function(z){
    n = length(z)
    f = rbeta(1,1+sum(z==1), 1+n-sum(z==1))
    return(f)
}

# sample p from p(p|g,z)
sample_p = function(g,z){
    nAA1 = sum(g=="AA" & z==1)
    naa1 = sum(g=="aa" & z==1)
    nAA0 = sum(g=="AA" & z==0)
    naa0 = sum(g=="aa" & z==0)
    nAa0 = sum(g=="Aa" & z==0)
    p = rbeta(1, 1+nAA1+2*nAA0+nAa0, 1+naa1+2*naa0+nAa0)
    return(p)
}

# Gibbs sampler
gibbs = function(g, niter=100){
    f = 0.5  # initialize
```

```
    g = 0.5
    z = sample_z(g,f,p)
    res = list(f = numeric(niter),p = numeric(niter),
               z = matrix(nrow=niter,ncol=length(g)))
    res$f[1] = f
    res$p[1] = p
    res$z[1,]= z
    for (i in 2:niter){
        f = sample_f(z)
        p = sample_p(g,z)
        z = sample_z(g,f,p)
        res$f[i] = f
        res$p[i] = p
        res$z[i,]= z
    }
    return(res)
}
```

Problem B

(1) Priors and Likelihood

Here we introduce a parameter π to denote the mixture component weights: $P(z_i = k) = \pi_k \quad (k = 0, 1)$

Likelihood $p(X|Z, P)$ is:

$$p(X|Z,P) = \prod_{i=1}^{n} p(x_i|z_i = k, P) = \prod_{i=1}^{n}\prod_{j=1}^{R} P_{kj}^{x_{ij}}(1 - P_{kj})^{1-x_{ij}}$$

Prior distribution $p(P, Z, \pi)$ is:

$$p(P, Z, \pi) = p(P)\ p(Z, \pi) = p(P)\ p(Z|\pi)\ p(\pi)$$

where the prior distributions are:

$$p(P) = \prod_{k=0}^{1}\prod_{j=1}^{R} p(P_{kj}), \qquad where \;\; P_{kj} \sim Uniform(0, 1), \;\; so \;\; p(P_{kj}) = 1$$

$$p(Z|\pi) = \prod_{i=1}^{n} p(z_i|\pi), \qquad where \;\; z_i|\pi \sim Multinomial(1, \pi), \;\; so \;\; P(z_i = k) = \pi_k \quad (k = 0, 1)$$

$$\pi = \{\pi_0, \pi_1\} \sim Dirichlet(1, 1)$$

$$or \;\; use: \qquad \pi_0 \sim Uniform(0, 1) = Beta(1, 1), \quad and \quad \pi_1 = 1 - \pi_0$$

(2) Full conditional distribution of pi

The full conditional distribution of $\pi = \{\pi_0, \pi_1\}$ is:

$$p(\pi|P, Z, X) \propto p(\pi, P, Z, X) = \prod_{i=1}^{n} p(\pi, P, z_i, x_i) = p(\pi)p(P)\prod_{i=1}^{n} p(z_i|\pi)\ p(x_i|z_i, P) \;\propto\; p(\pi)\prod_{i=1}^{n} p(z_i|\pi)$$

Since $\pi = \{\pi_0, \pi_1\} \sim Dirichlet(1, 1)$ and $z_i|\pi \sim Multinomial(1, \pi)$, so according to the results in homework 3, we can derive that:

$$p(\pi|P,Z,X) \;\propto\; p(\pi)\prod_{i=1}^{n} p(z_i|\pi) \;=\; \prod_{i=1}^{n} \pi_0^{1-z_i}\pi_1^{z_i} \;=\; \pi_0^{\,n-\sum_{i=1}^{n} z_i}\;\pi_1^{\sum_{i=1}^{n} z_i}$$

Therefore the conditional (or posterior) distribution of $\pi = \{\pi_0, \pi_1\}$ is:

$$\pi|P,Z,X \;\sim\; Dirichlet(1+n-\sum_{i=1}^{n} z_i,\; 1+\sum_{i=1}^{n} z_i)$$

(3) Modify Gibbs sampler

```r
#' @param x an R vector of data
#' @param P a K by R matrix of allele frequencies
#' @return the log-likelihood for each of the K populations
log_pr_x_given_P = function(x,P){
  tP = t(P)  #transpose P so tP is R by K
  return(colSums(x*log(tP)+(1-x)*log(1-tP)))
}

normalize = function(x){return(x/sum(x))}  #used in sample_z below


#' @param x an n by R matrix of data
#' @param P a K by R matrix of allele frequencies
#' @param pi a K vector of model probabilities
#' @return an n vector of group memberships: Z
sample_z = function(x,P,pi){            # sample Z from p(Z|X,P,pi)
  K = nrow(P)
  loglik_matrix = apply(x, 1, log_pr_x_given_P, P=P)
  lik_matrix = exp(loglik_matrix) * pi
  p.z.given.x = apply(lik_matrix,2,normalize) # normalize columns
  z = rep(0, nrow(x))
  for(i in 1:length(z)){
    z[i] = sample(1:K, size=1,prob=p.z.given.x[,i],replace=TRUE)
  }
  return(z)
}



#' @param x an n by R matrix of data
#' @param z an n vector of cluster allocations
#' @return a 2 by R matrix of allele frequencies : P
sample_P = function(x, z){              # sample P from p(P|X,Z,pi)
  R = ncol(x)
  P = matrix(ncol=R,nrow=2)
  for(i in 1:2){
    sample_size = sum(z==i)
    if(sample_size==0){
      number_of_ones=rep(0,R)
    } else {
      number_of_ones = colSums(x[z==i,])
    }
```

```
    P[i,] = rbeta(R,1+number_of_ones,1+sample_size-number_of_ones)
  }
  return(P)
}


#' @param z an n vector of cluster allocations
#' @return a K=2 vector of model probabilities: pi
sample_pi = function(z){             # sample pi from p(pi|X,Z,P)
  counts = colSums(outer(z,1:2,FUN="=="))  # a vector c(#1, #2)
  pi = as.vector(gtools::rdirichlet(1,counts+1))
  return(pi)
}


# Gibbs Sampler
gibbs = function(x, niter=100){
  pi = rep(1/2,2)  # initialize
  z = sample(1:2,nrow(x),replace=TRUE)
  res = list(z = matrix(nrow=niter, ncol=nrow(x)),
             pi= matrix(nrow=niter, ncol=2))
  res$z[1,] = z
  res$pi[1,]= pi

  for(i in 2:niter){
    P = sample_P(x,z)       # sample P  from p(P|Z,X,pi)
    z = sample_z(x,P,pi)    # sample Z  from p(Z|P,X,pi)
    pi= sample_pi(z)        # sample pi from p(pi|X,Z,P)
    res$z[i,] = z
    res$pi[i,]= pi
  }
  return(res)
}
```

(4) Illustration using simulated data

```
# simulate data from the model
set.seed(100)

#' @param n number of samples
#' @param P a 2 by R matrix of allele frequencies
#' @param pi a K vector of model probabilities

r_simplemix = function(n,P,pi){
  R = ncol(P)
  z = sample(1:2,prob=pi,size=n,replace=TRUE)  #simulate z as 1 or 2
  x = matrix(nrow=n, ncol=R)
  for(i in 1:n){
    x[i,] = rbinom(R,rep(1,R),P[z[i],])
  }
  return(list(x=x, z=z))
}
```

```
P = rbind(c(0.5,0.5,0.5,0.5,0.5,0.5),c(0.001,0.999,0.001,0.999,0.001,0.999))
sim = r_simplemix(n=50, P, pi=c(0.2,0.8))    # true pi = c(0.2,0.8)
x = sim$x
```

```
# try the Gibbs sampler on the data simulated above
res = gibbs(x,100)
colMeans(res$pi)
```

```
## [1] 0.2002631 0.7997369
```

Comments:

We can see that the posterior distribution we get from the sampled values of $\pi = c(0.2002631, 0.7997369)$ using the Gibbs sampler is very close to the true value of $\pi = (0.2, 0.8)$.

Problem C

(1) Log-likelihood

According to homework 2, we have:

$$X_j|\mu,\sigma,s_j \ \sim \ N(\mu,\ \sigma^2+s_j^2)$$

So the log-likelihood is:

$$l(\mu,\sigma) = \log L(\mu,\sigma) = \log \prod_j P(X_j = x_j|\mu,\sigma,s_j) = \sum_j \log P(X_j = x_j|\mu,\sigma,s_j)$$

$$= \sum_j \log\left(\frac{1}{\sqrt{2\pi(\sigma^2+s_j^2)}}\ \exp\{-\frac{(x_j-\mu)^2}{2(\sigma^2+s_j^2)}\}\right) \ = \ \sum_j\left(-\frac{1}{2}\log(2\pi(\sigma^2+s_j^2)) - \frac{(x_j-\mu)^2}{2(\sigma^2+s_j^2)}\right)$$

```
log_lik = function(x,s,mu,eta) {
    # non-negative constraint: sigma > 0
    # eta = log(sigma),   sigma^2 = exp(2*eta)
    sigma2 = exp(2*eta)
    loglik = sum( -0.5*log(2*pi*(sigma2+s^2)) - 0.5*(x-mu)^2/(sigma2+s^2) )
    return(loglik)
    }
```

(2) MH sampler

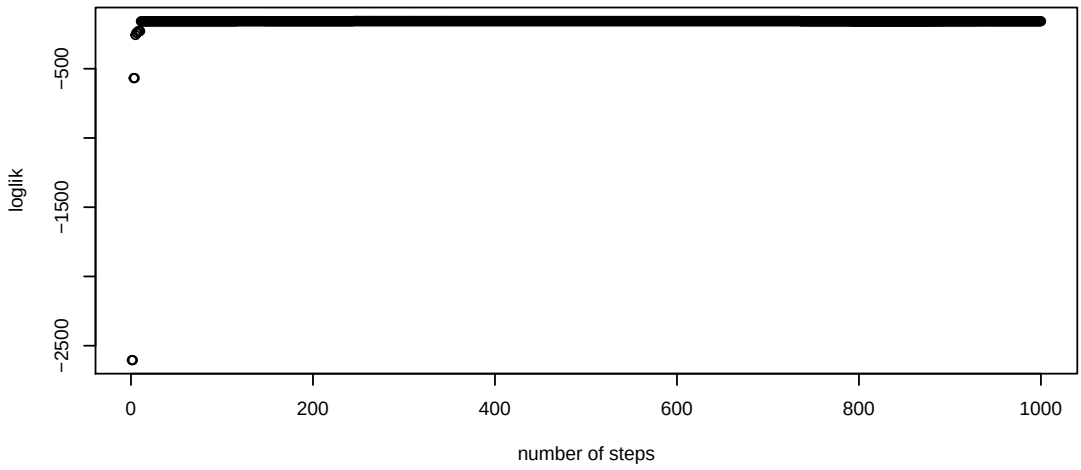
```
# prior of mu and sigma
prior = function(mu, eta){    # eta = log(sigma)
    if(abs(mu)<10^6 & abs(eta)<10) {return(1)}
    else {return(0)}
}

# MH sampler: sample from the joint distribution of mu and eta
MHsampler = function(x,s,niter, mu.startval, eta.startval, mu.proposalsd, eta.proposalsd){
    mu = rep(0,niter)
    eta= rep(0,niter)
    mu[1] = mu.startval
    eta[1]= eta.startval
```

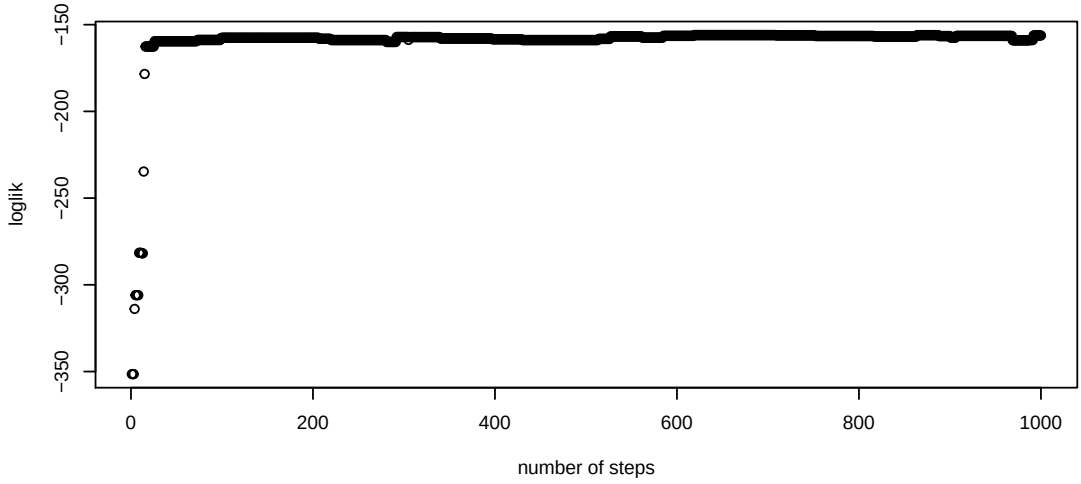
```
    for(i in 2:niter){
        current.mu = mu[i-1]
        current.eta= eta[i-1]
        new.mu = current.mu  + rnorm(1,0,mu.proposalsd)
        new.eta= current.eta + rnorm(1,0,eta.proposalsd)

        LR = exp(log_lik(x,s,new.mu,new.eta)-log_lik(x,s,current.mu,current.eta))
        A = prior(new.mu,new.eta)/prior(current.mu,current.eta) * LR

        if(runif(1)<A){
            mu[i] = new.mu
            eta[i]= new.eta
        } else {
            mu[i] = current.mu
            eta[i]= current.eta
        }
    }
    return(list(mu=mu, eta=eta))
}
```

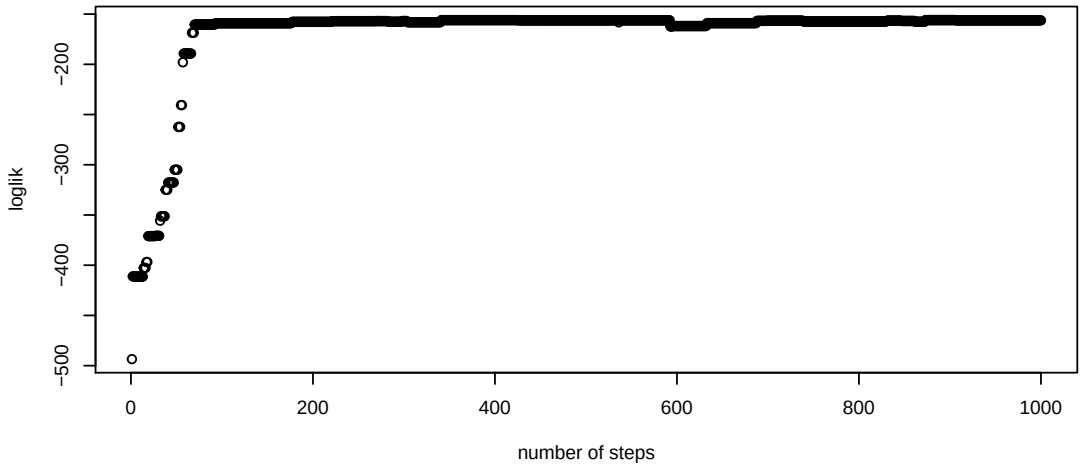
(3) Different starting values

```
# simulate some data
set.seed(123)
n=100;  mu=5;  sigma=1       # true mu=5, sigma=1, eta=log(sigma)=0
theta = rnorm(n,mu,sigma)
s = abs(rnorm(n))    # non-negative constraint: s>0
x = rep(NA,n)
for (i in 1:n) {
    x[i] = rnorm(1,theta[i],s[i])
}
```

```
# try different starting values
mu.startval = c(1,10,20)
eta.startval= c(-1,1,2)

par(mfrow=c(3,1))
samplers = list()
for (i in 1:3) {
    niter = 1000
    samplers[[i]] = MHsampler(x,s,niter,mu.startval[i],eta.startval[i],1,1)
    loglik = rep(0,niter)
    for (j in 1:niter) {
        mu = unlist(samplers[[i]]$mu)[j]
        eta= unlist(samplers[[i]]$eta)[j]
        loglik[j] = log_lik(x,s,mu,eta)
    }
    plot(loglik, xlab="number of steps")
}
```


loglik
−500
−1500
−2500
0
200
400
600
800
1000
number of steps


loglik
−150
−200
−250
−300
−350
0
200
400
600
800
1000
number of steps


loglik
−200
−300
−400
−500
0
200
400
600
800
1000
number of steps

```r
# compare posterior means with true values
burnin = 200
mu.mean = rep(0,3)
eta.mean = rep(0,3)
for (i in 1:3) {
    mu = unlist(samplers[[i]]$mu)[burnin:1000]
    eta= unlist(samplers[[i]]$eta)[burnin:1000]
    mu.mean[i] = mean(mu)
    eta.mean[i] = mean(eta)
}
results = data.frame(mu.pm=mu.mean, eta.pm=eta.mean)
results
```

```
##      mu.pm      eta.pm
## 1 5.147027 -0.09156342
## 2 5.144993 -0.14081012
## 3 5.143020 -0.09324236
```

Comments:

Here I run the MH sampler with different starting values: $\mu = 1, 10, 20$ and $\eta = -1, 1, 2$, while the true values are: $\mu = 5$ and $\eta = 0$.

I run 1000 iterations for my MH sampling algorithm. From the plots we can see that 1000 iterations is enough for the algorithm to converge, and after 200 iterations, the value of $\log \pi(\mu^t, \eta^t)$ settles down to a "steady state", so I choose the first 200 iterations to discard as "burnin".

The posterior mean of μ and η are very close to the true values for all the three samplers with different starting values. So our MH sampler works well here.

(4) 8-shcools data

```r
# 8-shcools data
x = c(28,8,-3,7,-1,1,18,12)
s = c(15,10,16,11,9,11,10,18)
mean(x)
```

```
## [1] 8.75
```

```r
quantile(x)
```

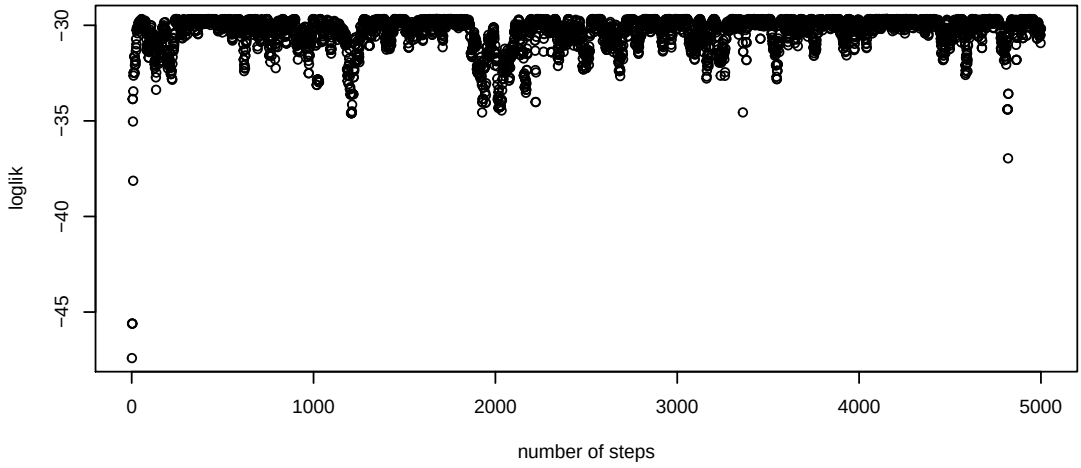
```
##   0%  25%  50%  75% 100%
## -3.0  0.5  7.5 13.5 28.0
```

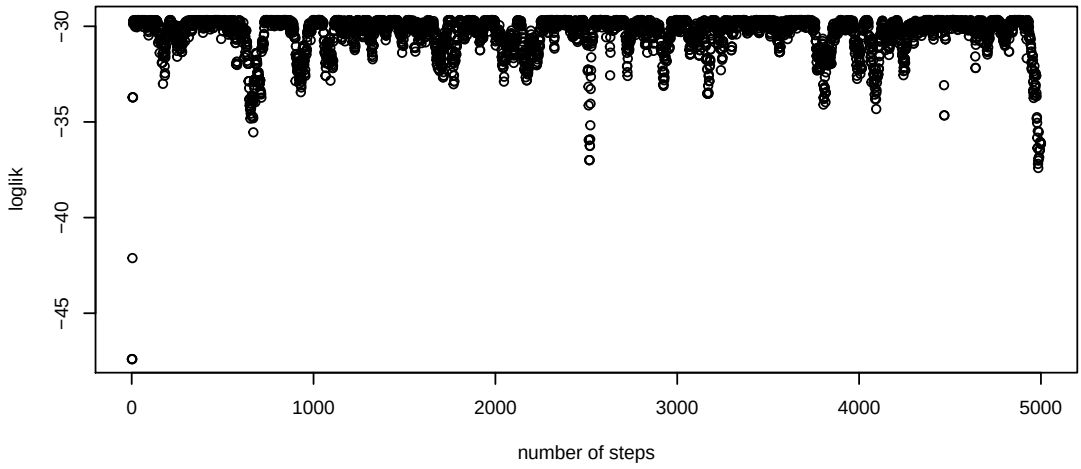
```r
log(s)
```

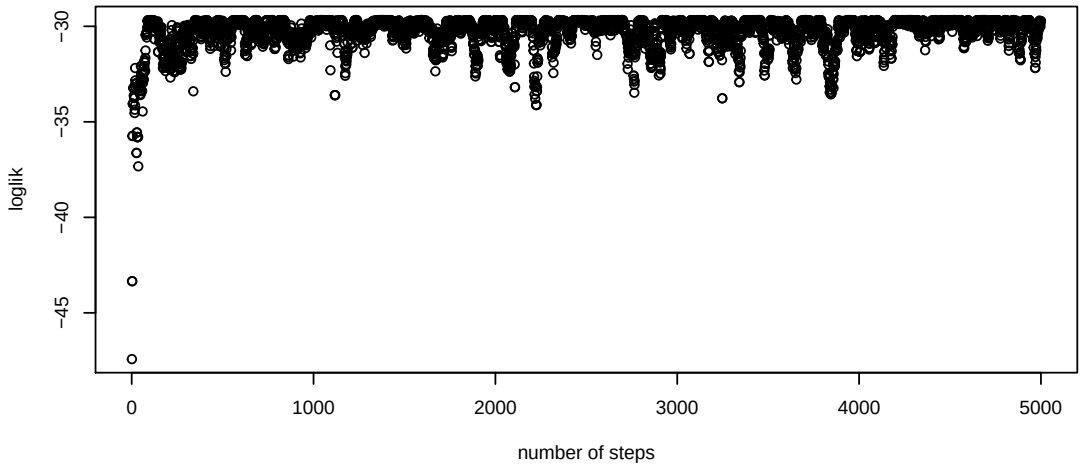
```
## [1] 2.708050 2.302585 2.772589 2.397895 2.197225 2.397895 2.302585 2.890372
```

```r
# try different starting values
set.seed(123)
mu.startval = c(1,10,20)
eta.startval= c(5,5,5)

par(mfrow=c(3,1))
samplers = list()
for (i in 1:3) {
    niter = 5000
    samplers[[i]] = MHsampler(x,s,niter,mu.startval[i],eta.startval[i],1,1)
```

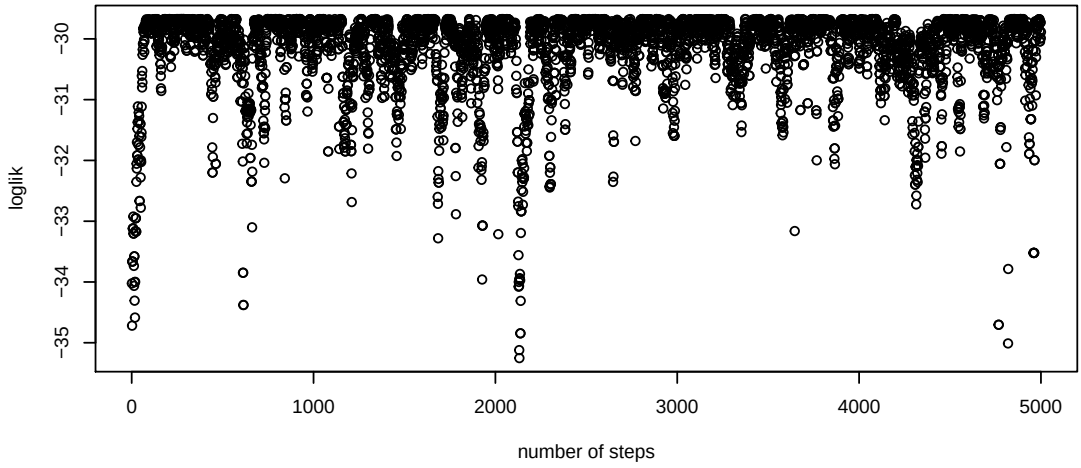
```
    loglik = rep(0,niter)
    for (j in 1:niter) {
        mu = unlist(samplers[[i]]$mu)[j]
        eta= unlist(samplers[[i]]$eta)[j]
        loglik[j] = log_lik(x,s,mu,eta)
    }
    plot(loglik, xlab="number of steps")
}
```

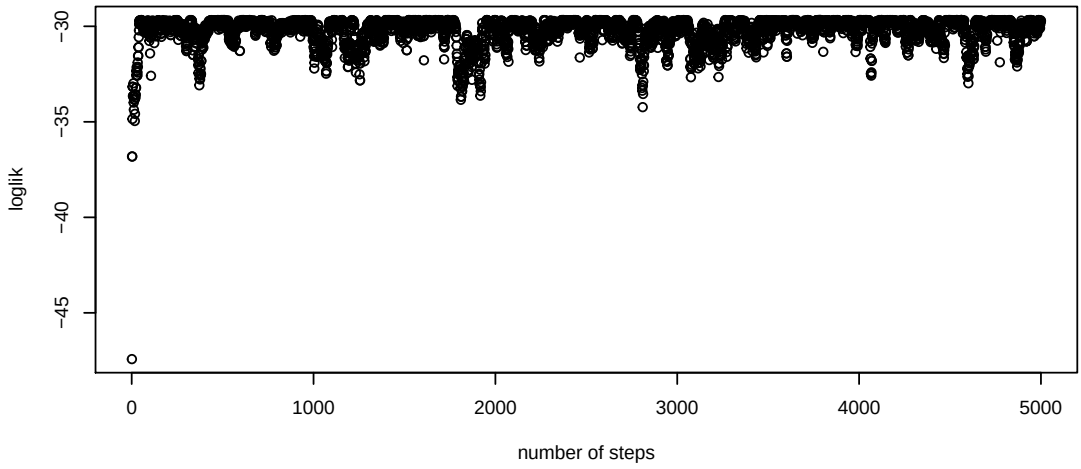

loglik
−30
−35
−40
−45
0
1000
2000
3000
4000
5000
number of steps

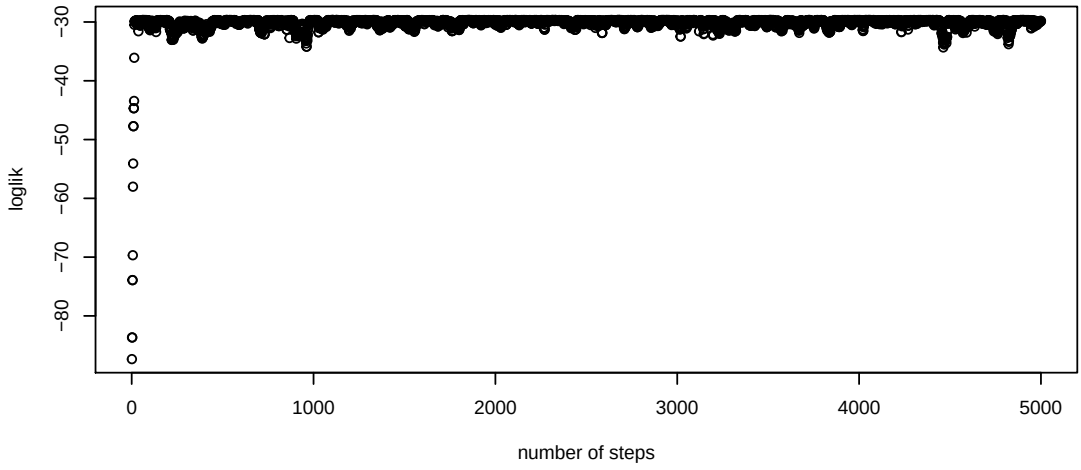

loglik
−30
−35
−40
−45
0
1000
2000
3000
4000
5000
number of steps


loglik
−30
−35
−40
−45
0
1000
2000
3000
4000
5000
number of steps

```r
# try different starting values
mu.startval = c(20,20,20)
eta.startval= c(1,5,10)

par(mfrow=c(3,1))
samplers = list()
for (i in 1:3) {
    niter = 5000
    samplers[[i]] = MHsampler(x,s,niter,mu.startval[i],eta.startval[i],1,1)
    loglik = rep(0,niter)
    for (j in 1:niter) {
        mu = unlist(samplers[[i]]$mu)[j]
        eta= unlist(samplers[[i]]$eta)[j]
        loglik[j] = log_lik(x,s,mu,eta)
    }
    plot(loglik, xlab="number of steps")
}
```


loglik
−30
−31
−32
−33
−34
−35
0
1000
2000
3000
4000
5000
number of steps


loglik
−30
−35
−40
−45
0
1000
2000
3000
4000
5000
number of steps


loglik
−30
−40
−50
−60
−70
−80
0
1000
2000
3000
4000
5000
number of steps

Comments:

Here I still run the MH sampler with different starting values, but in this example, we do not know the true values of μ and η.

Here I run 5000 iterations for my MH sampling algorithm instead of 1000 iterations in the previous question. From the plots we can see that though with large number of iterations, the value of $\log \pi(\mu^t, \eta^t)$ keeps fluctuating and does not settles down to a "steady state".

So to find the reason, first, I tried different starting value of $\mu = 1, 10, 20$ with same starting value of $\eta = 5$, it still fluctuates and does not converge well. Then, I tried different starting value of $\eta = 1, 5, 10$ with same starting value of $\mu = 20$, we can see that when $\eta = 20$, which is large, the $\log \pi(\mu^t, \eta^t)$ value seems to roughly settle down to a more "steady state". So the fluactuation might be because of the large s_j values in this example.

(5) posterior second moment & posterior variance

Since we can approximate the posterior mean by:

$$E(\theta_j|X) \approx (1/T) \sum_t E(\theta_j|X, \mu^t, \eta^t)$$

Similarly, we can approximate the posterior second moment by:

$$E(\theta_j^2|X) \approx (1/T) \sum_t E(\theta_j^2|X, \mu^t, \eta^t)$$

conditional second moment is wrong!!!

Again, according to homework 2, we have the posterior distribution of θ_j is:

$$\theta_j|X_j, \mu, \sigma \ \sim \ N(\frac{\sigma^2 X_j + s_j^2 \mu}{\sigma^2 + s_j^2}, \ \frac{\sigma^2 s_j^2}{\sigma^2 + s_j^2})$$

where $\sigma^2 = \exp(2\eta)$

So the posterior mean of θ_j is:

$$E(\theta_j|X_j, \mu, \sigma) \ = \ \frac{\sigma^2 X_j + s_j^2 \mu}{\sigma^2 + s_j^2}$$

Thus:

$$E(\theta_j^2|X_j, \mu, \sigma) \ = \ \left(\frac{\sigma^2 X_j + s_j^2 \mu}{\sigma^2 + s_j^2}\right)^2$$

Therefore, we can approximate the posterior variance:

$$Var(\theta_j|X) = E(\theta_j^2|X) - (E(\theta_j|X))^2 \approx (1/T) \sum_t E(\theta_j^2|X, \mu^t, \eta^t) - \left((1/T) \sum_t E(\theta_j|X, \mu^t, \eta^t)\right)^2$$

and the posterior standard deviation:

$$sd(\theta_j|X) = \sqrt{Var(\theta_j|X)}$$

(6) 8-schools data

```
approx.post = function(x,s,mu,eta){
    sigma2 = exp(2*eta)
    t = length(mu)
    n = length(x)
    mean = rep(0,n)
    var = rep(0,n)
    for(i in 1:n){
        mean[i] = 1/t*sum(  (sigma2*x[i]+s[i]^2*mu)/(sigma2+s[i]^2)    )
        var[i]  = 1/t*sum( ((sigma2*x[i]+s[i]^2*mu)/(sigma2+s[i]^2))^2 )
                -(1/t*sum(  (sigma2*x[i]+s[i]^2*mu)/(sigma2+s[i]^2)    ))^2
    }
    sd = sqrt(var)
    return(list(post.mean=mean, post.sd=sd))
}
```

```
# 8-schools data
x=c(28,8,-3,7,-1,1,18,12)
s=c(15,10,16,11,9,11,10,18)

# approximate
post.mean = matrix(0,3,8)
post.sd = matrix(0,3,8)
for(i in 1:3){
    mu = unlist(samplers[[i]]$mu)
    eta= unlist(samplers[[i]]$eta)
    posteriors = approx.post(x,s,mu,eta)
    post.mean[i,] = unlist(posteriors$post.mean)
    post.sd[i,] = unlist(posteriors$post.sd)
}
post.mean
```

```
##          [,1]     [,2]     [,3]     [,4]     [,5]     [,6]     [,7]
## [1,] 8.510812 8.113704 7.945680 8.090044 7.778157 7.918058 8.436512
## [2,] 8.980508 8.636635 8.495648 8.615840 8.329859 8.462103 8.928300
## [3,] 8.025202 7.659311 7.494141 7.635987 7.350720 7.476926 7.957718
##          [,8]
## [1,] 8.197345
## [2,] 8.710446
## [3,] 7.730535
```

```
post.sd
```

```
##          [,1]     [,2]     [,3]     [,4]     [,5]     [,6]     [,7]
## [1,] 9.410142 8.875600 8.759750 8.857915 8.601973 8.712546 9.259278
## [2,] 9.916767 9.495900 9.399185 9.481480 9.254040 9.355302 9.814921
## [3,] 8.984654 8.470043 8.357170 8.451228 8.216467 8.316006 8.832846
##          [,8]
## [1,] 8.993544
## [2,] 9.594465
## [3,] 8.569427
```

Comments:

In the previous EB results, the posterior means are very similar with each other and the posterior standard deviation is smaller, however, here the posterior means are different with each other and the posterior standard

deviation is larger.

The two approaches will produce similar results if the posterior distribution is very concentrated around the MLE estimation. However, when it it not concentrated, the EB approach will give different results since it "integrates out the uncertainty" in μ and η.