HW4

Jinhong Du - 12243476

May 8, 2020

Contents

Problem A: EM Algorithm

Derive and implement an EM algorithm to estimate the means of a mixture of two normal distributions. Consider the mixture of two normal distributions, each with variance 1 and equal mixture proprortions. That is: $X_1, \ldots, X_n \sim f(\cdot)$ where $f(\cdot) = 0.5N(\cdot; \mu_1, 1) + 0.5N(\cdot; \mu_2, 1)$. Here $N(\cdot; \mu, \sigma^2)$ denotes the density of the normal distribution with mean μ and variance σ^2.

- Derive an EM algorithm to estimate μ_1, μ_2 (Recall introduce missing data indicators $Z_{ik} = 1$ if observation i comes from component k, 0 otherwise)
- Implement the EM algorithm and check it by simulation. Specifically, check i) the likelihood increases, and ii) estimates are close to true values, at least when n is large and μ_1, μ_2 are not too similar.

i) Derivation of the EM Algorithm Let $Z_{ik} \in \{0, 1\}$ be the indicator of whether observation i comes from component k for $i = 1, \ldots, n$ and $k = 1, 2$. Let π_1, π_2 be the mixture proprortions, then $\pi_1 = \pi_2 = \frac{1}{2}$.

The incomplete data log-likelihood is

$$l(\boldsymbol{\mu}; \boldsymbol{X}) = \sum_{i=1}^{n} \log p(X_i|\boldsymbol{\mu}) = \sum_{i=1}^{n} \log \left(\sum_{k=1}^{2} \pi_k p(X_i|\mu_k) \right).$$

The complete data likelihood is

$$\begin{aligned} L(\boldsymbol{\mu}; \boldsymbol{X}, \boldsymbol{Z}) &= \prod_{i=1}^{n} p(X_i, \boldsymbol{Z}_i|\boldsymbol{\mu}) \\ &= \prod_{i=1}^{n} \prod_{k=1}^{2} \left[\pi_k p(X_i|\mu_k)\right]^{Z_{ik}} \\ &= \frac{1}{2^{\frac{3n}{2}} \pi^{\frac{n}{2}}} \prod_{i=1}^{n} \prod_{k=1}^{2} \left(e^{-\frac{1}{2}(X_i - \mu_k)^2} \right)^{Z_{ik}} \end{aligned}$$

and the complete data log-likelihood is

$$l(\boldsymbol{\mu}; \boldsymbol{X}, \boldsymbol{Z}) = -\frac{3n}{2} \log(2) - \frac{n}{2} \log(\pi) - \frac{1}{2} \sum_{i=1}^{n} \sum_{k=1}^{2} Z_{ik}(X_i - \mu_k)^2.$$

Then we can calculate the conditional distribution of the latent variable by law of total probability

$$p(Z_{ik}|X_i, \boldsymbol{\mu}) = \frac{\pi_k p(X_i, Z_{ik}|\mu_k)}{\sum_{k'=1}^{2} \pi_{k'} p(X_i, Z_{ik'}|\mu_{k'})} = \frac{e^{-\frac{1}{2}(X_i - \mu_k)^2}}{\sum_{k'=1}^{2} e^{-\frac{1}{2}(X_i - \mu_{k'})^2}},$$

for $Z_{ik} \in \{0, 1\}$. So the conditional expectation is given by

$$\begin{aligned} \mathbb{E}_{Z_{i1}|X_i, \boldsymbol{\mu}}(Z_{i1}) &= 1 \times p(Z_{i1}|X_i, \boldsymbol{\mu}) + 0 \times p(Z_{i2}|X_i, \boldsymbol{\mu}) = p(Z_{i1}|X_i, \boldsymbol{\mu}) \\ \mathbb{E}_{Z_{i2}|X_i, \boldsymbol{\mu}}(Z_{i2}) &= 0 \times p(Z_{i1}|X_i, \boldsymbol{\mu}) + 1 \times p(Z_{i2}|X_i, \boldsymbol{\mu}) = p(Z_{i2}|X_i, \boldsymbol{\mu}). \end{aligned}$$

The Q-function is given by

$$Q(\boldsymbol{\mu}, \boldsymbol{\mu}^{(t)}) = \mathbb{E}_{\boldsymbol{Z}|\boldsymbol{X},\boldsymbol{\mu}^{(t)}} l(\boldsymbol{\mu}; \boldsymbol{X}, \boldsymbol{Z}) = -\frac{3n}{2}\log(2) - \frac{n}{2}\log(\pi) - \frac{1}{2}\sum_{i=1}^{n}\sum_{k=1}^{2}\mathbb{E}_{\boldsymbol{Z}_{ik}|X_i,\boldsymbol{\mu}^{(t)}}(Z_{ik})(X_i-\mu_k)^2.$$

Taking derivative with respect to μ_k and setting it to zero,

$$\frac{Q(\boldsymbol{\mu}, \boldsymbol{\mu}^{(t)})}{\partial \mu_k} = -\sum_{i=1}^{n}\mathbb{E}_{\boldsymbol{Z}_{ik}|X_i,\boldsymbol{\mu}^{(t)}}(Z_{ik})(X_i-\mu_k) = 0,$$

yields

$$\mu_k = \frac{\sum_{i=1}^{n}\mathbb{E}_{\boldsymbol{Z}_{ik}|X_i,\boldsymbol{\mu}^{(t)}}(Z_{ik})X_i}{\sum_{i=1}^{n}\mathbb{E}_{\boldsymbol{Z}_{ik}|X_i,\boldsymbol{\mu}^{(t)}}(Z_{ik})}.$$

ii) Implement of the EM Algorithm.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm

def generate_data(n, mu):
    '''
    Params:
        n  - number of observations
        mu - [2,] array of means
    Returns:
        X  - [n,] array of observations
        Z  - [n,] array of missing data
    '''
    Z = np.random.binomial(1, 0.5, n)
    X = np.random.randn(n) + np.where(Z==1, mu[0], mu[1])
    return X, Z

def log_likelihood(X, mu):
    l = np.sum(np.log(0.5*(norm.pdf(X, mu[0]) + norm.pdf(X, mu[1]))))
    return l

def plot_result(L, mu_true, mu_init, mu_fianl):
    plt.figure()
    plt.plot(L)
    plt.xlabel('iteration')
    plt.ylabel('complete data log-likelihood')
    plt.title('$\mu_{true}$=(%.2f,%.2f),$\mu_{true}$=(%.2f,%.
↪2f),$\mu_{true}$=(%.2f,%.2f)'%(
        mu_true[0],mu_true[1],mu_init[0],mu_init[1],mu_fianl[0],mu_fianl[1]))
    plt.plot()
```

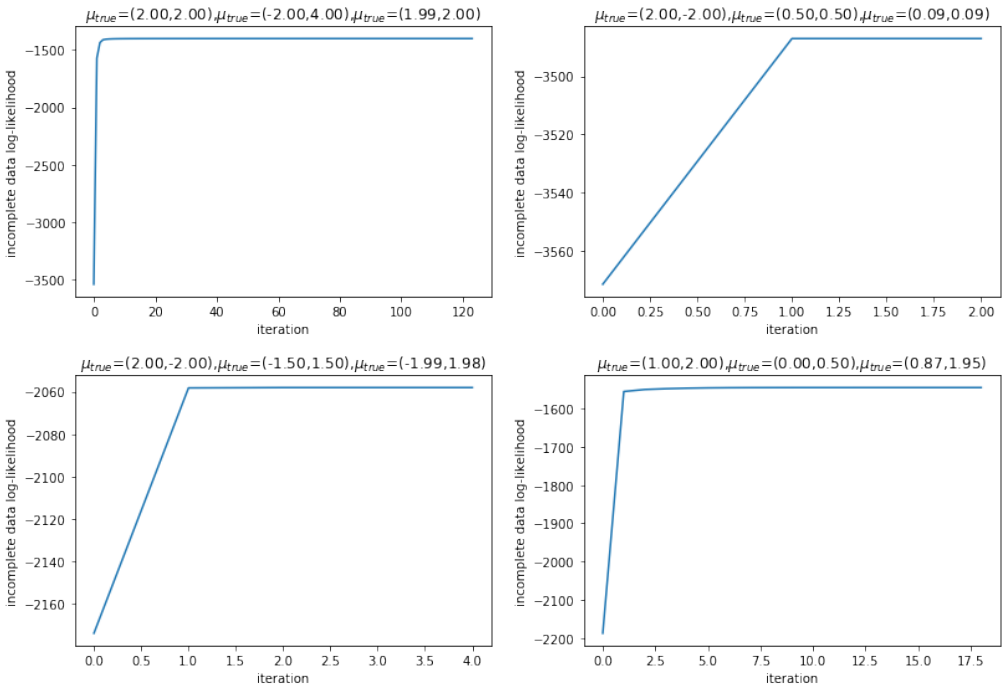
[2]:
```
def run_simulation(n, true_mu, init_mu, epsilon=1e-5):
    '''
    Params:
        n       - number of observations
        true_mu - [2,] array of true means
        init_mu - [2,] array of initial means for EM algorithms
    Returns:
        L_list  - array of incomplete log-likelihood
        mu_list - array of means in all steps
    '''
    X, Z = generate_data(n, true_mu)
    L_list = [log_likelihood(X, init_mu)]
    mu_list = [init_mu]
    mu = np.array(init_mu)
    eps = 1
    while eps>epsilon:
        # E step
        E_Z = np.exp(-0.5*(np.c_[X,X]-mu)**2)
        E_Z = E_Z/np.sum(E_Z, axis=-1, keepdims=True)

        # M step
        mu = np.sum(E_Z * np.expand_dims(X, -1), axis=0) / np.sum(E_Z, axis=0)
        l = log_likelihood(X, mu)

        mu_list.append(mu)
        L_list.append(l)
        eps = L_list[-1]-L_list[-2]
    return L_list, mu_list

n = 1000
mu_list = np.array([[2.0,2.0], [2.0,-2.0], [2.0,-2.0], [1.0, 2.0]])
init_mu_list = np.array([[-2.0,4.0], [0.5,0.5], [-1.5,1.5], [0.0, 0.5]])

for i in range(4):
    L, mu_hat = run_simulation(n, mu_list[i], init_mu_list[i])
    plot_result(L, mu_list[i], init_mu_list[i], mu_hat[-1])
```



From the above plots, we can see that

- The incomplete data log-likelihood increases as iteration increases.
- In cases as in the first and fourth polts, when the means are initialized well, the estimated means are similar to the true means.
- However, when the means are initialized away from the true means, as in the second and third plots, the estimated means are much different from the true menas. This may be because that the EM algorithm stuck in some local maxima.

Problem B: MH Algorithm

Complete the first 3 exercises in: https://stephens999.github.io/fiveMinuteStats/MH-examples1.html

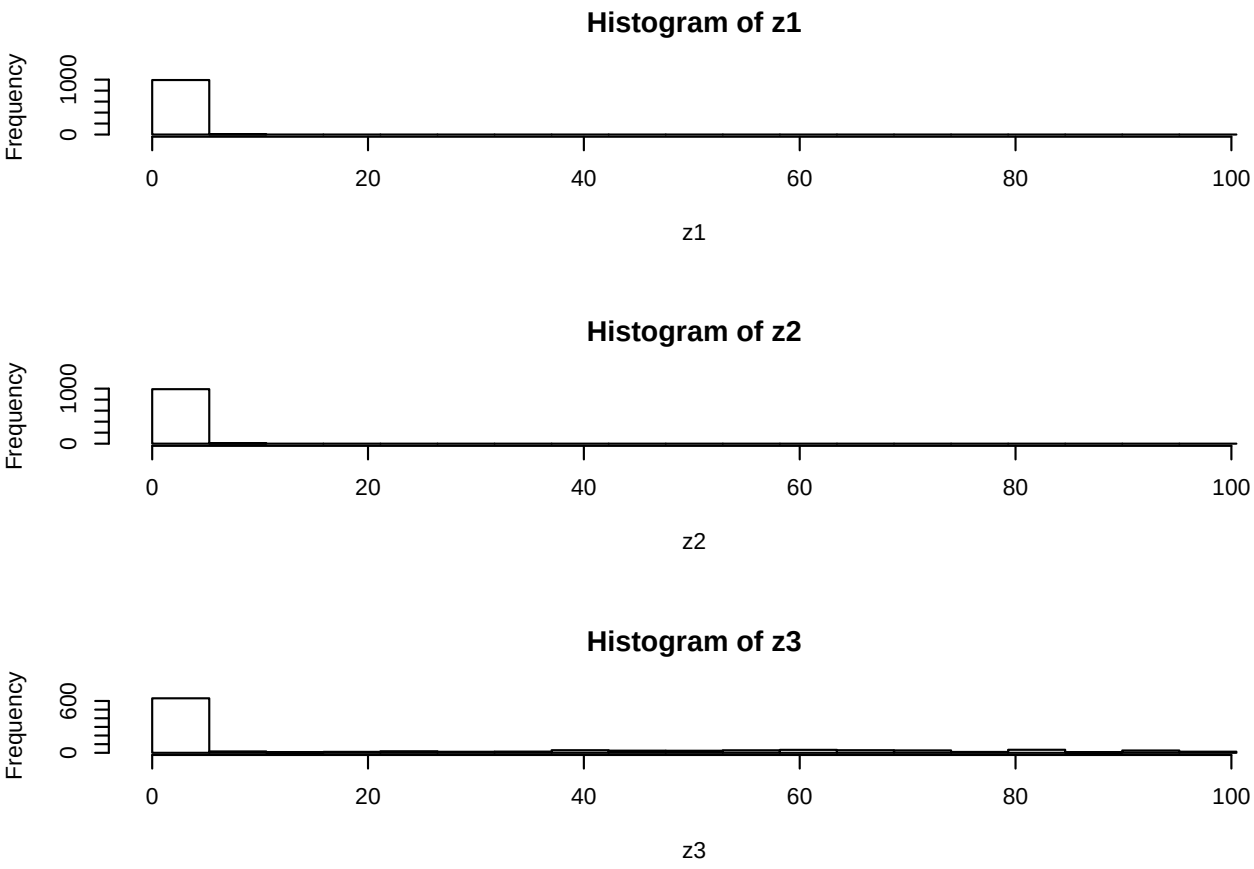
i) Example 1: Sampling from an exponential distribution using MCMC

Use the function easyMCMC to explore the following:

a. how do different starting values affect the MCMC scheme?

[3]:
```
target <- function(x){
  if(x<0){
    return(0)}
  else {
    return( exp(-x))
  }
}

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 <- Inf
    }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)
}
z1 <- easyMCMC(1000,1,1)
z2 <- easyMCMC(1000,10,1)
z3 <- easyMCMC(1000,100,1)
par(mfcol=c(3,1))
maxz <- max(c(z1,z2,z3))
hist(z1,breaks=seq(0,maxz,length=20))
hist(z2,breaks=seq(0,maxz,length=20))
hist(z3,breaks=seq(0,maxz,length=20))
```

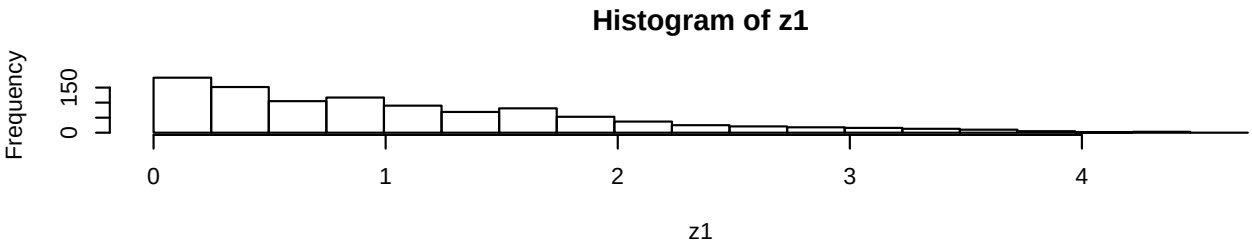


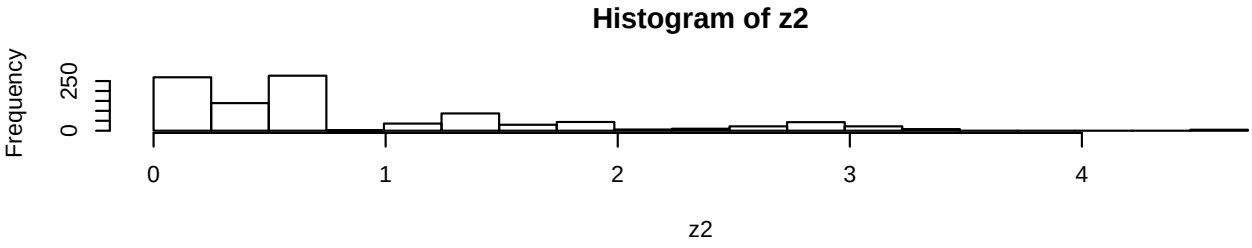
As we can see, when starting at a point away from zero, the estimated distribution of the target function will have a heavy tail.

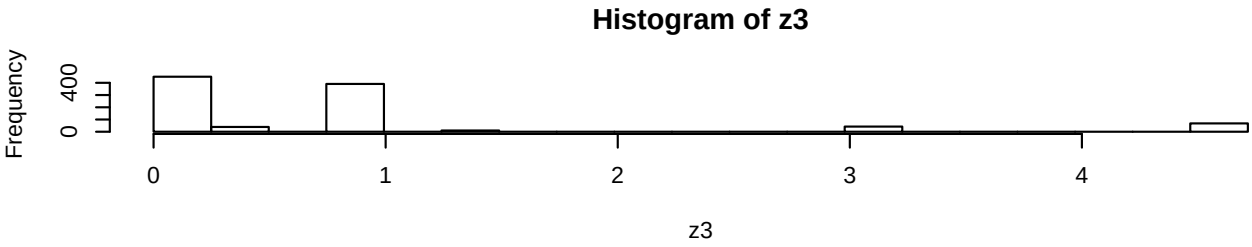
b. what is the effect of having a bigger/smaller proposal standard deviation?

[4]:
```
z1 <- easyMCMC(1000,3,1)
z2 <- easyMCMC(1000,3,10)
z3 <- easyMCMC(1000,3,100)
par(mfcol=c(3,1))
maxz <- max(c(z1,z2,z3))
hist(z1,breaks=seq(0,maxz,length=20))
hist(z2,breaks=seq(0,maxz,length=20))
hist(z3,breaks=seq(0,maxz,length=20))
```

From the plots below, we can see that the estimated distribution is smoother when having a smaller proposal standard deviation. While when having a bigger proposal standard deviation, the estimated distribution will be rather irregular.







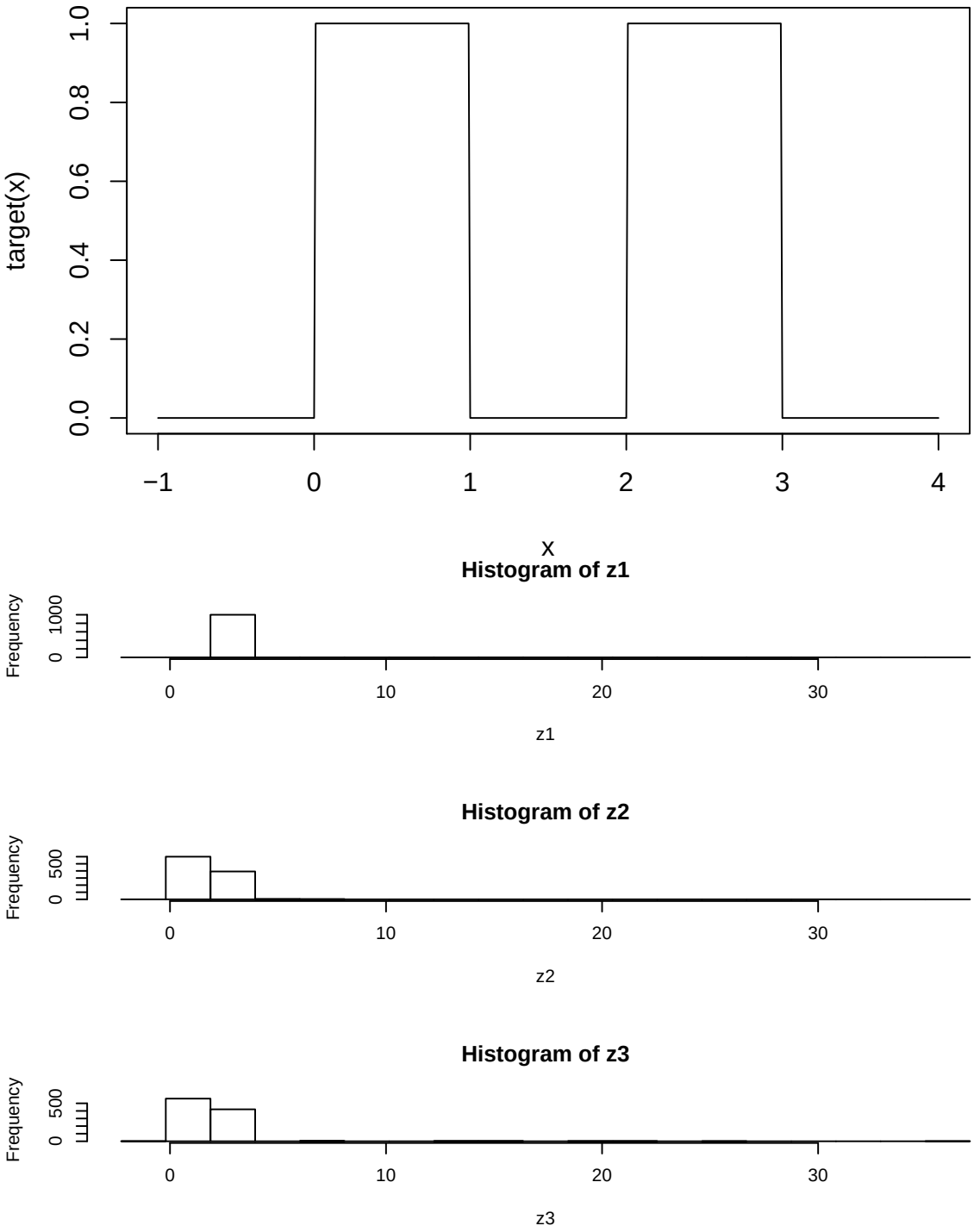
c. try changing the target function to the following

```
target = function(x){
return((x>0 & x <1) + (x>2 & x<3))
}
```

What does this target look like? What happens if the proposal sd is too small here? (try e.g. 1 and 0.1)

[5]:
```
target = function(x){
return((x>0 & x <1) + (x>2 & x<3))
}
x <- seq(-1,4,0.01)
plot(x, target(x), 'l')

z1 <- easyMCMC(1000,3,0.1)
z2 <- easyMCMC(1000,3,1)
z3 <- easyMCMC(1000,3,10)
par(mfcol=c(3,1))
maxz <- max(c(z1,z2,z3))
hist(z1,breaks=seq(0,maxz,length=20))
hist(z2,breaks=seq(0,maxz,length=20))
hist(z3,breaks=seq(0,maxz,length=20))
```



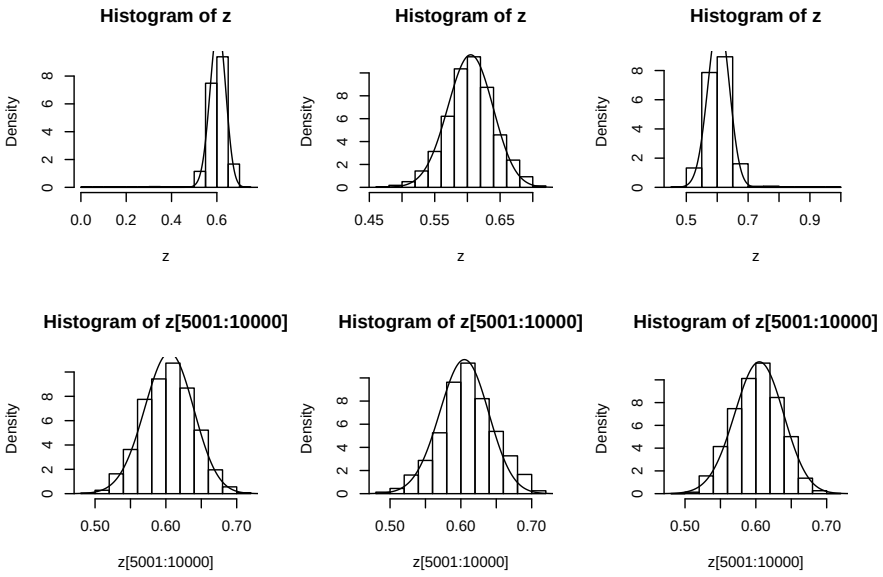
The new target function is a step function. If the proposal standard deviation is too small, then the resulted estimated istribution will be only unimodal. Since the proposal standard deviation is too small, the proposal points can not make it to the other mode of the target distribution.

ii) Investigate how the starting point and proposal standard deviation affect the convergence of the algorithm.

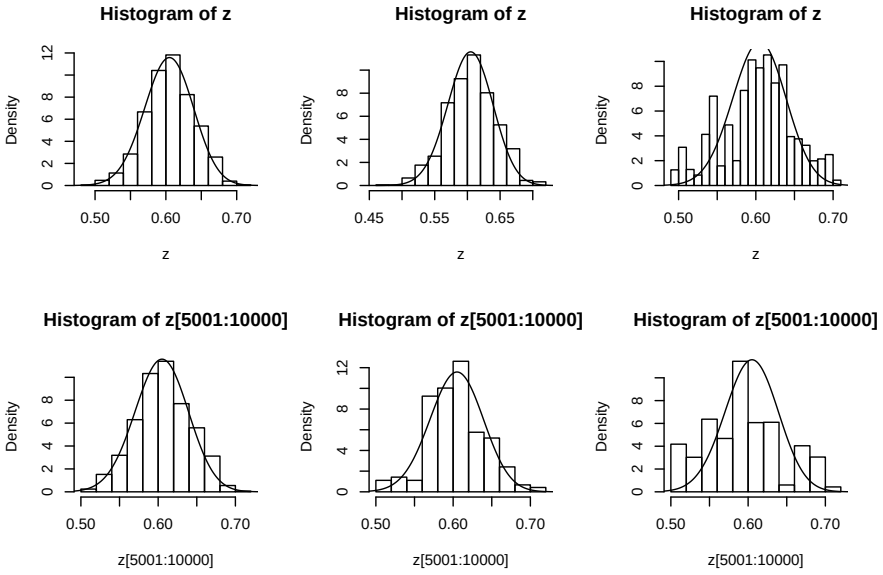
[6]:
```
prior <- function(p){
  if((p<0) || (p>1)){  # || here means "or"
    return(0)}
  else{
    return(1)}
}
likelihood <- function(p, nAA, nAa, naa){
  return(p^(2*nAA) * (2*p*(1-p))^nAa * (1-p)^(2*naa))
}
psampler <- function(nAA, nAa, naa, niter, pstartval, pproposalsd){
  p <- rep(0,niter)
  p[1] <- pstartval
  for(i in 2:niter){
    currentp <- p[i-1]
    newp <- currentp + rnorm(1,0,pproposalsd)
    if(prior(currentp) * likelihood(currentp,nAA,nAa,naa)==0){
        A <- Inf
    }else{
        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)
}
par(mfcol=c(2,3))
for(p in c(0.01,0.5,0.99)){
    z <- psampler(50,21,29,10000,p,0.01)
    x <- seq(0,1,length=1000)
    hist(z, prob=T)
    lines(x,dbeta(x,122, 80))
    hist(z[5001:10000], prob=T) # burnin
    lines(x,dbeta(x,122, 80))
}
```

The three columns of the below plots correspond to initial value of $p = 0.01, 0.5, 0.99$ respectively. The first and second rows are the result without and with burnin procedure respectively.

As we can see, we will have more samples near to zero if the initial value of p is small and more samples near to one if the initial value of p is large. However, the estimated distribution seems not sensitive to the choice of initial value of p with a burnin procedure.



[7]:
```
par(mfcol=c(2,3))
for(sd in c(0.01,1,5)){
    z <- psampler(50,21,29,10000,0.5,sd)
    x <- seq(0,1,length=1000)
    hist(z, prob=T)
    lines(x,dbeta(x,122, 80))
    hist(z[5001:10000], prob=T) # burnin
    lines(x,dbeta(x,122, 80))
}
```



As for the proposal standard deviation, the larger one will cause a more irregular estimated distribution.

iii) Write a short MCMC routine to sample from the joint distribution of f and p. Use this sample to obtain point estimates for f and p (e.g. using posterior means) and interval estimates for both f and p (e.g. 90% posterior credible intervals), when the data are $n_{AA} = 50, n_{Aa} = 21, n_{aa} = 29$

[8]:
```
fplikelihood <- 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)
}
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)
    if(prior(currentp) * fplikelihood(currentf,currentp,nAA,nAa,naa)==0){
        A <- Inf
    }else{
        A <- prior(newp)*fplikelihood(newf,newp,nAA,nAa,naa)/(prior(currentp) *␣
 ↪fplikelihood(currentf,currentp,nAA,nAa,naa))
    }
    if(runif(1)<A){
      p[i] <- newp;f[i] <- newf
    } else {
      p[i] <- currentp;f[i] <- currentf
    }
    }
    return(list(f=f,p=p)) # return a "list" with two elements named f and p
}
res <- fpsampler(50,21,29,10000,0.5,0.5,0.01,0.01)
mean(res$f[5001:10000])
quantile(res$f[5001:10000], c(.05,.95))
mean(res$p[5001:10000])
quantile(res$p[5001:10000], c(.05,.95))
```

```
      5%       95%
0.4132550 0.6833386
[1] 0.6055407
      5%       95%
0.5355091 0.6717942
```

As we can see, the estimated posterior mean of f is about 0.5681879, and its 90% credible interval is $(0.4132550, 0.6833386)$. The estimated posterior mean of p is about 0.6055407, and its 90% credible interval is $(0.5355091, 0.6717942)$.