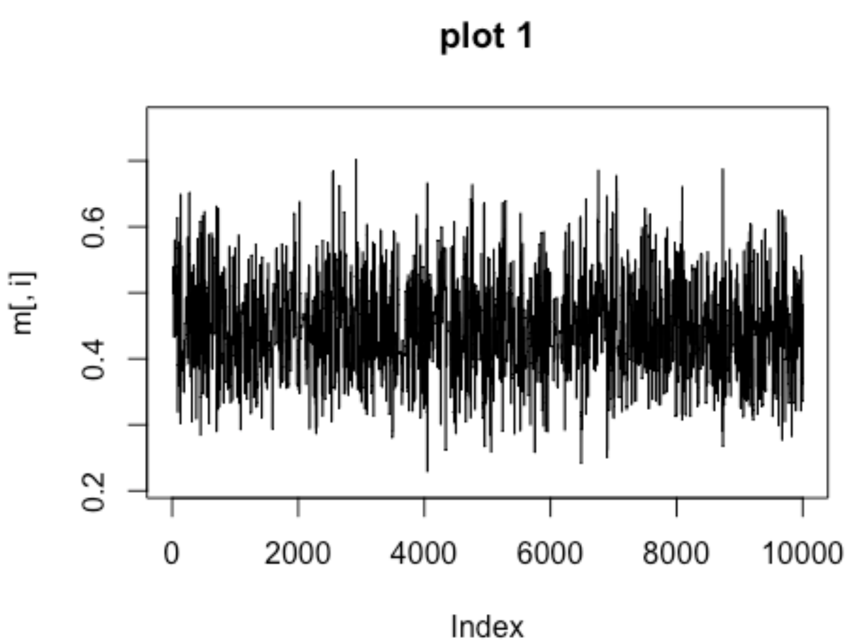
Q2

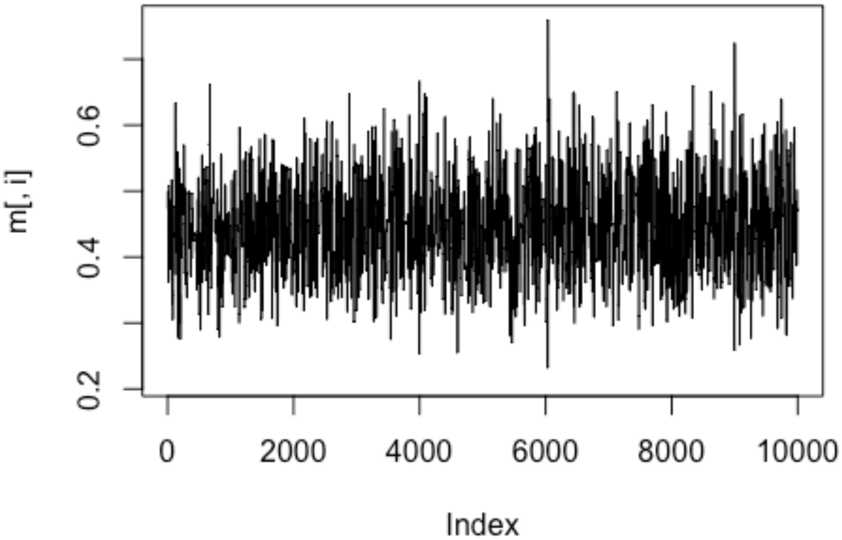
```r
set.seed(440)
x = rweibull(n = 24, shape = 0.5, scale = 64)
prob <- function(alpha = NULL, datax = NULL){
 if (alpha <= 0 )
   return(-Inf)
 else{n = length(datax)
   result = n*log(alpha)+(alpha-1)*sum(log(datax))-n*alpha*log(64)-sum(datax^alpha)/(64^alpha)

 return (result)}

}
mcmc1 <- function(n = NULL, xx=0.5){
 l = numeric(n)
 l[1] = xx
 for (i in 2:n){
   val = rgamma(1, shape = 1, rate = 2)
   val2 = runif(1)

   accept = exp(prob(alpha = val, datax = x))/exp(prob(alpha = l[i-1],datax = x))*dgamma(l[i-
1],shape = 1, rate = 2)/dgamma(val, shape = 1,rate = 2)

   if (val2 < accept) {l[i] = val}
   else {l[i] = l[i-1]}


 }


 return (l)

 }
mcmc2 <- function(n = NULL, xx = 0.5){
 l = numeric(n)
 l[1] = xx

 for (i in 2:n){
   val = rgamma(1, shape = 1,rate = 1/l[i-1])
   val2 = runif(1)

   accept = exp(prob(alpha = val, datax = x))/exp(prob(alpha = l[i-1],datax = x))*dgamma(l[i-1],
shape = 1, rate = 1/val)/dgamma(val, shape = 1, rate = 1/l[i-1])

   if (val2 < accept) {l[i] = val}
   else {l[i] = l[i-1]}
 }
 return(l)
}
mcmc3 <- function(n = NULL, xx = 0.5){
 l = numeric(n)
 l[1] = xx

 for (i in 2:n){
   val = rnorm(1, mean = l[i-1], sd = sqrt(0.1))
   val2 = runif(1)

   accept = exp(prob(alpha = val, datax = x))/ exp(prob(alpha = l[i-1], datax = x))*dnorm(l[i-1],
mean = val,sd = sqrt(0.1))/dnorm(val, mean = l[i-1], sd = sqrt(0.1))

   if (val2 < accept)
     l[i] = val
   else
     l[i] = l[i-1]

 }

 return (l)
}
```

```
loglik <- function(alpha = NULL, data = NULL){

 n = length(data)
 val = numeric(length(alpha))

 for (i in 1:length(alpha)){
   val[i] = n*log(alpha[i])+(alpha[i]-1)*sum(log(data))-n*alpha[i]*log(64)-sum((data/64)^alpha[i])

 }
 return (val)
}

alpha_est = optimize(f = loglik, interval = c(0, 50), data = x,maximum=TRUE )$maximum

mcmc4 <- function(n = NULL, xx=0.5){
 l = numeric(n)
 l[1] = xx

 for (i in 2:n){
   val = rnorm(1, mean = alpha_est, sd = sqrt(1/(24/alpha_est^2+sum((x/64)^alpha_est*
(log(x/64))^2))))
   val2 = runif(1)

   accept = exp(prob(alpha = val, datax = x))/exp(prob(alpha = l[i-1],datax = x))*dnorm(l[i-
1],mean = alpha_est,sd = sqrt(1/(24/alpha_est^2+sum((x/64)^alpha_est*
(log(x/64))^2))))/dnorm(val, mean = alpha_est,sd =
sqrt(1/(24/alpha_est^2+sum((x/64)^alpha_est*(log(x/64))^2))))

   if (val2<accept) {l[i] = val}

   else {l[i] = l[i-1]}
 }

 return (l)
}
set.seed(440)
n0= 10^4
m = matrix(0, nrow = n0, ncol = 4)
m[,1] = mcmc1(n0)
m[,2] = mcmc2(n0)
m[,3] = mcmc3(n0)
m[,4] = mcmc4(n0)

for (i in 1:4){
 plot(m[,i], main = paste('plot', i), type = 'l',ylim = range(m))
}
```

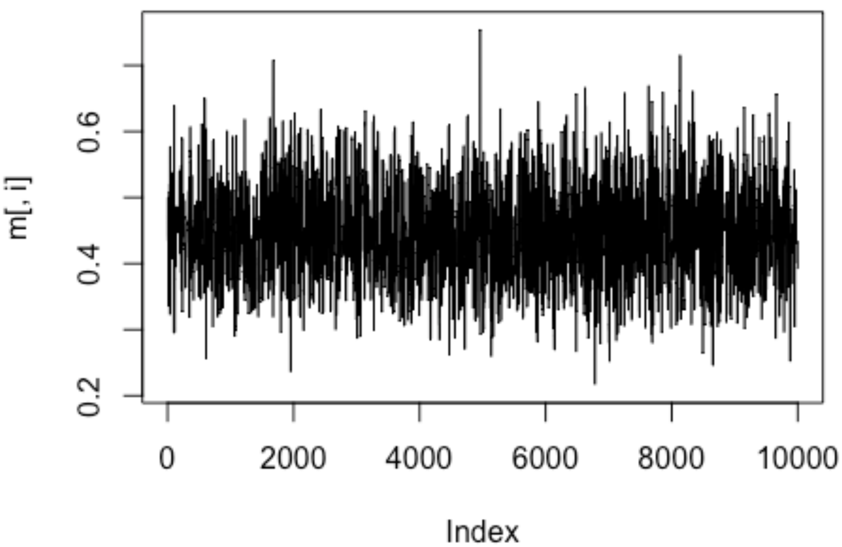


plot 2

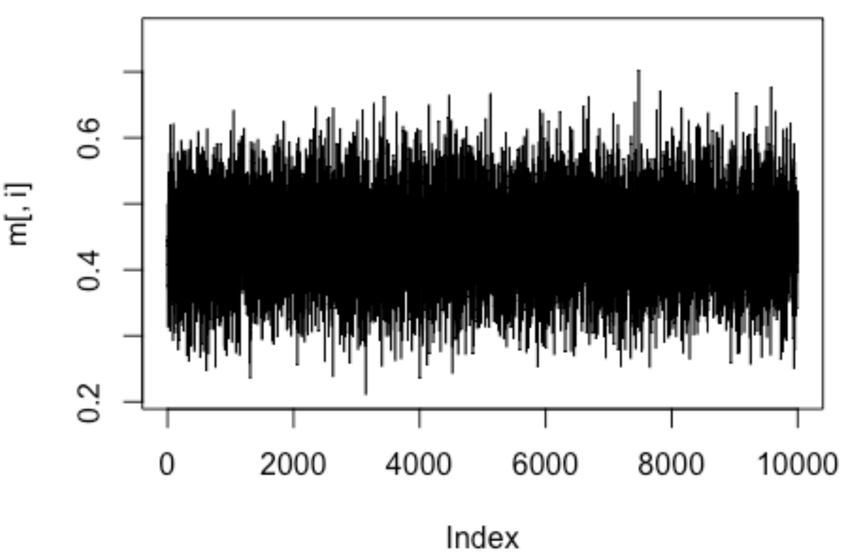
plot 2



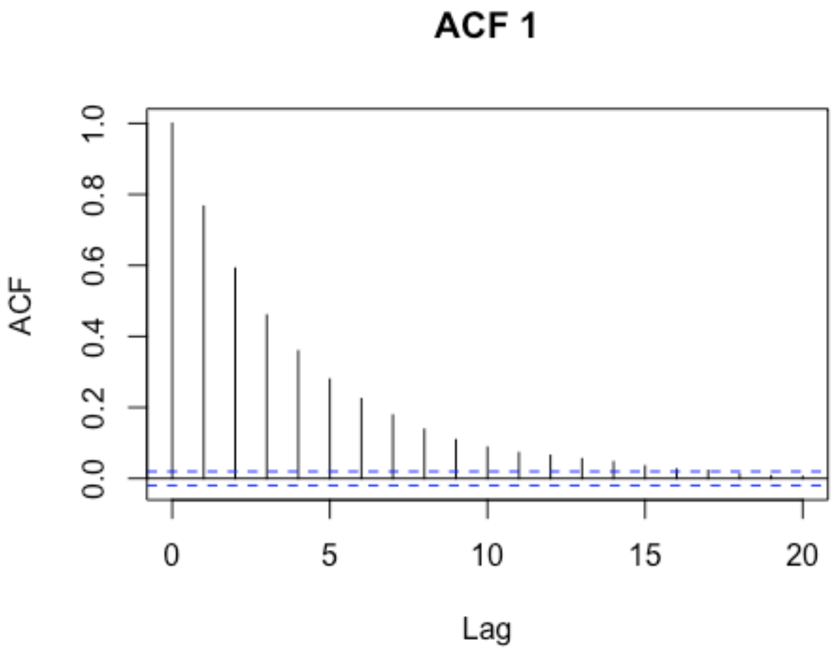
plot 3

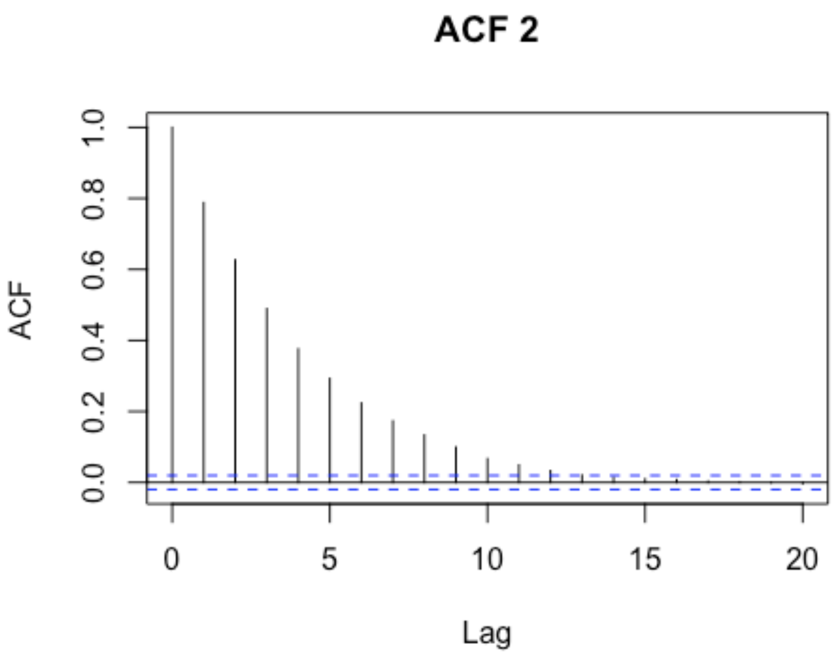


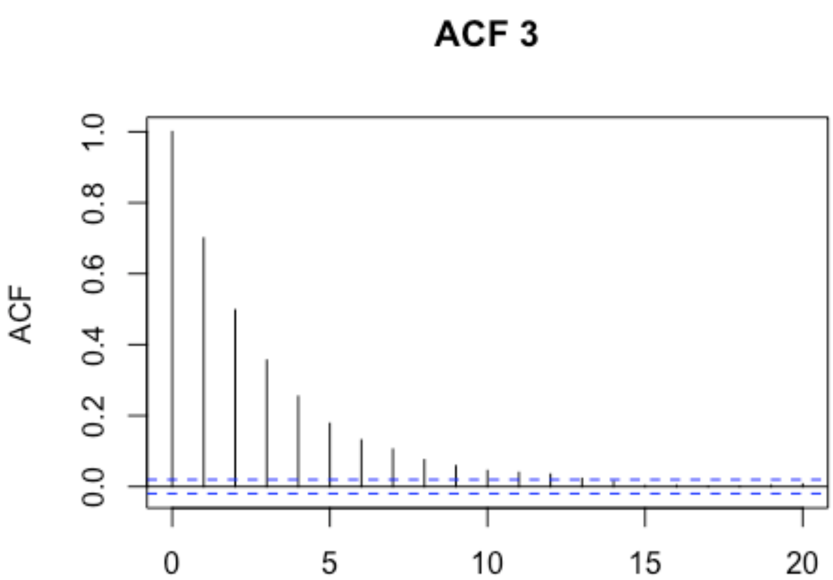
plot 4



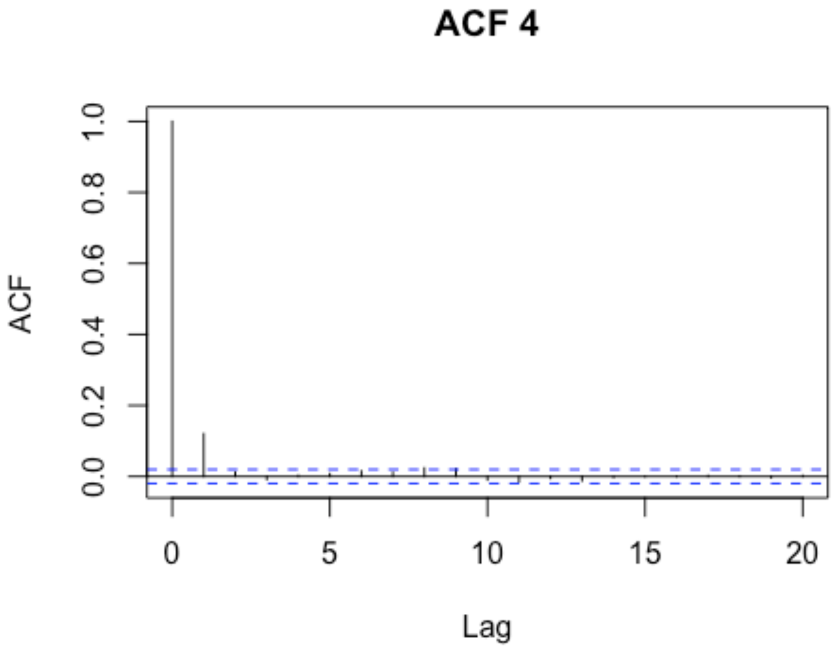
```
for (i in 1:4) acf( m[,i], main=paste("ACF", i), lag.max=20)
```



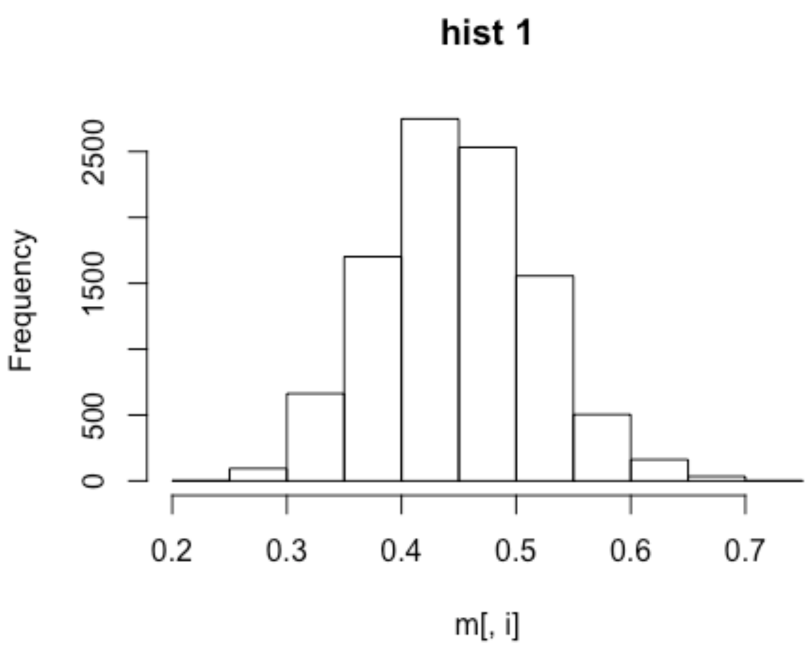


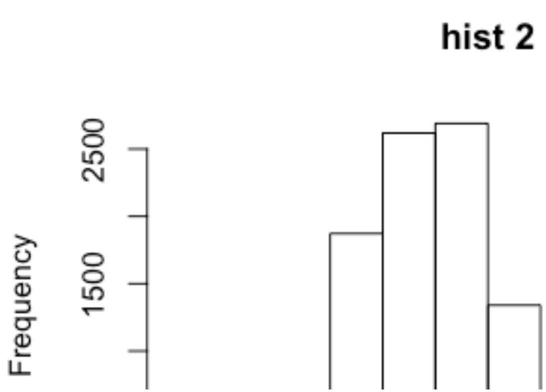


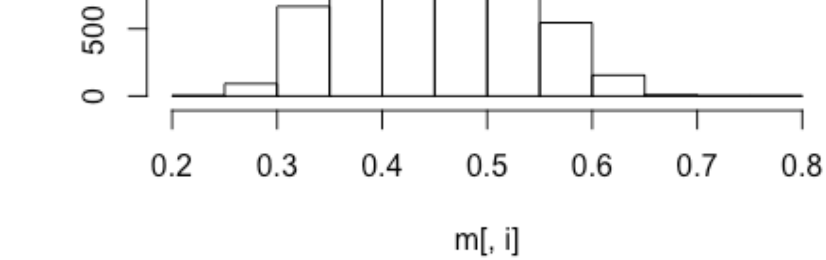
Lag

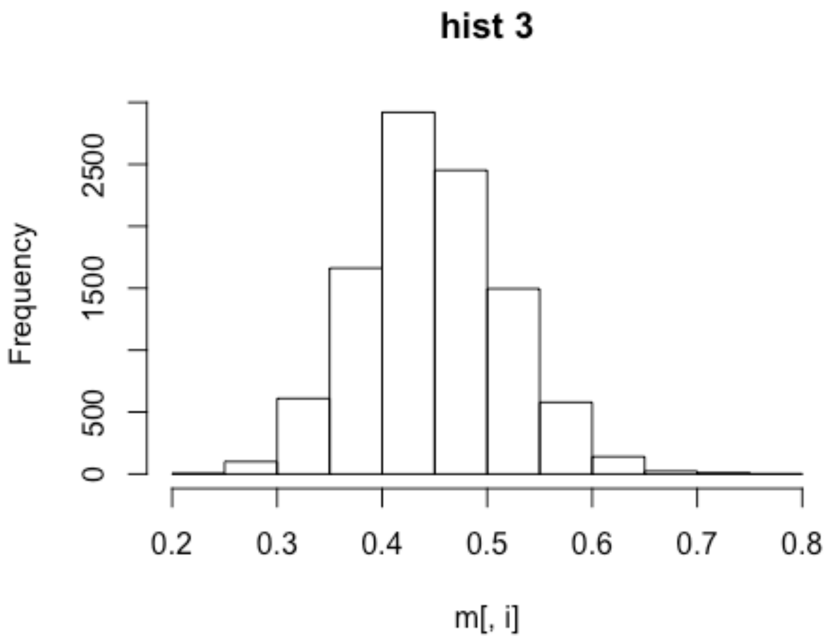


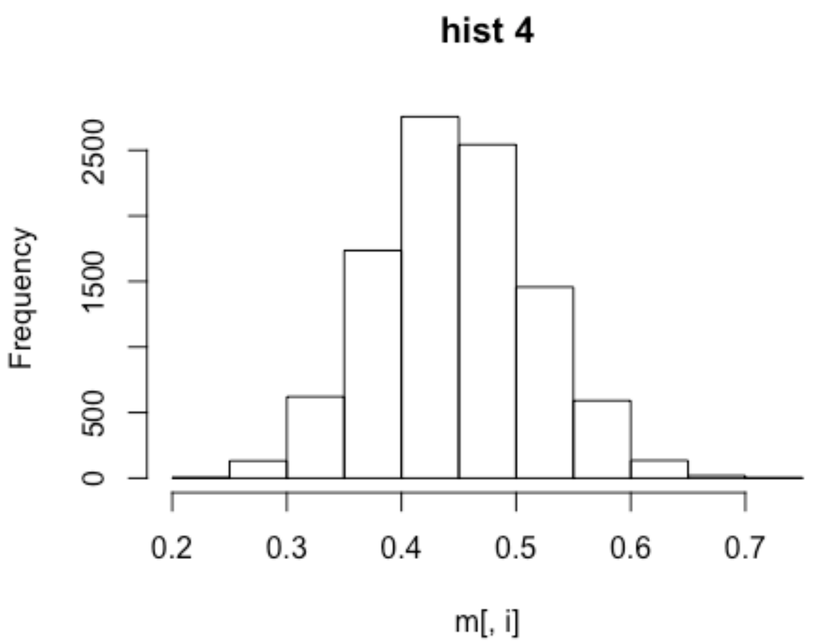

```r
for (i in 1:4){
 hist(m[,i], main = paste('hist',i))
}
```



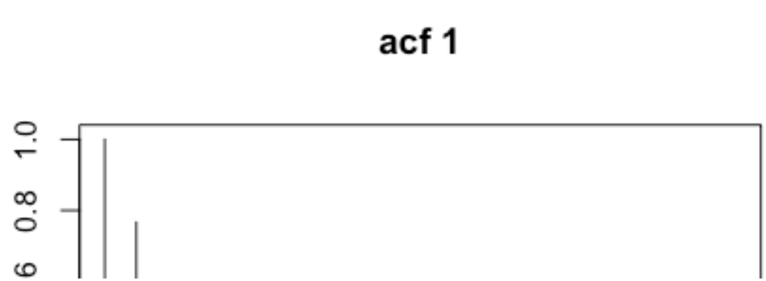


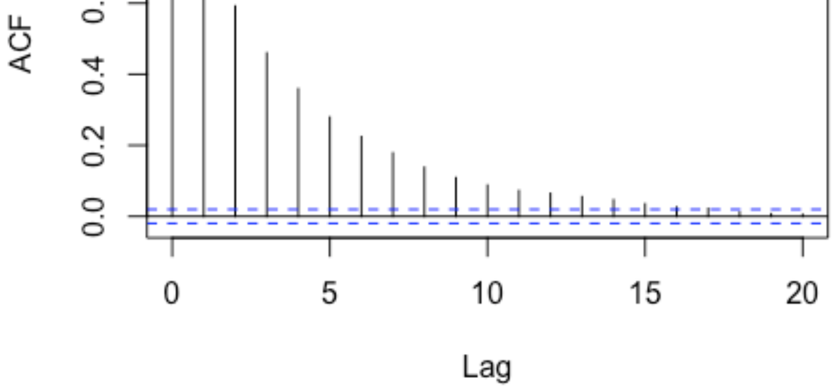




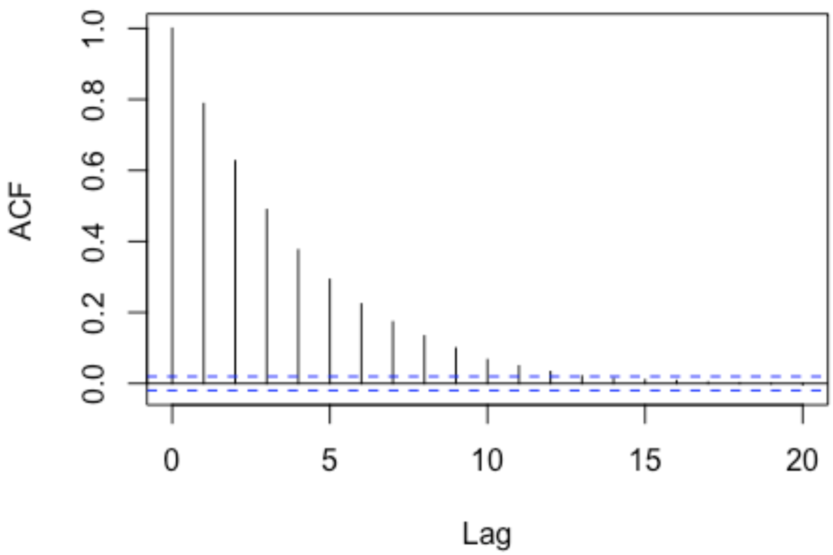



```
for (i in 1:4) acf(m[,i], main = paste('acf',i),lag.max=20)
```

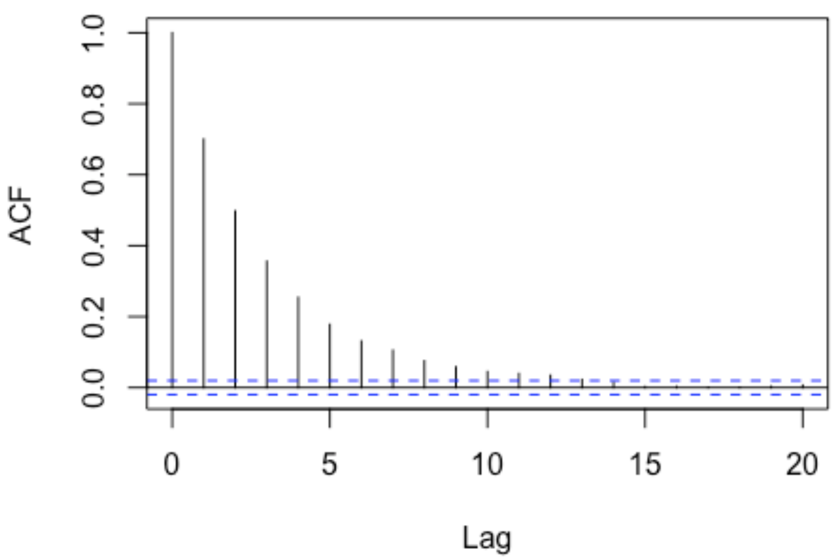




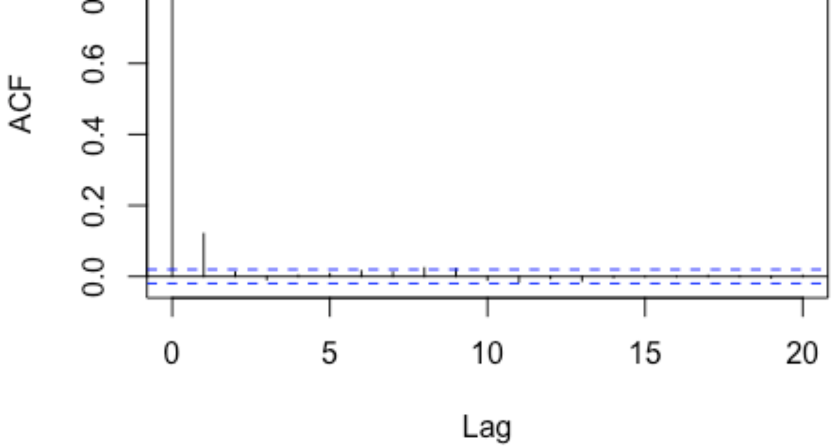
acf 2



acf 3



acf 4



As can be seen from the trace plots, the second and third model are mixing well. As can be seen from the histogram plots, the results of four models are similar. As can be seen from the acf plots, the last models has less correlation.

```
a = rbind(1:4,
      accept.rate = apply(m,2,function(x){mean(x[-1] != x[-length(x)])}),
      average = apply(m,2,mean),
      variance = apply(m,2,var),
      mixing = apply(m,2,function(x){mean(diff(x)^2)}))
a
##                    [,1]        [,2]        [,3]        [,4]
##             1.000000000 2.000000000 3.000000000 4.000000000
## accept.rate 0.179817982 0.176017602 0.263926393 0.958395840
## average     0.448379060 0.445791558 0.448360564 0.447574259
## variance    0.004678330 0.004752066 0.004713946 0.004800260
## mixing      0.002184189 0.002015804 0.002824891 0.008445014
```

In conclusion, the fourth model seems like is better than other three models. However, their results are similar.