Project 5

Zihuan Qiao

2016/11/29

1.(a)

```
# auxiliary function: compute log(sum(exp(x)))
lse <- local({
  log.epsilon2 <- -53 * log(2) # ~ -36.74
  function (x) {
    m <- max(x); x <- x - m
    m + log(sum(exp(x[x > log.epsilon2])))
  }
})


# forward probability
forward <- function (y, logI, logP, E) {
  n <- length(y); ns <- length(logI) # = nrow(logP)
  f <- matrix(nrow = n, ncol = ns)
  f[1,] <- logI + log(sapply(c(1:ns),FUN=function(j) dnorm(y[1],E[1,j],E[2,j]))) #use conditional emiss
  for (i in 2:n) # for each position in sequence
    for (j in 1:ns) # for each X_i
      f[i, j] <- lse(f[i - 1,] + logP[,j]) + log(dnorm(y[i],E[1,j],E[2,j]))
  return(f)
}


#initialize input
#initial probabilities `logI`: since the chain starts from state 2
I <- c(0,1,0)
logI <- log(I)
# transition probs `logP`
P <- matrix(c(0.5,0.05,0,0.5,0.9,0.5,0,0.05,0.5),nrow=3,ncol=3)
logP <- log(P)
# initialize emmision Probability Distributions
E <- matrix(c(-1,0.7,0,0.5,1,0.7),nrow=2,ncol=3)
# read observations y
y <- scan("cgh.txt")


f <- forward(y,logI,logP,E)
logPY <- log(sum(exp(f[200,])))
logPY
```

```
## [1] -196.3416
```

(b).

```
# viterbi algorithm
viterbi <- function (s, logI, logP, E) {
  n <- length(s); ns <- length(logI) # = nrow(logP)
  m <- matrix(nrow = n, ncol = ns) # maxima
  b <- matrix(nrow = n, ncol = ns) # backtrack pointers
  # recurse
  m[1,] <- logI + log(sapply(c(1:ns),FUN=function(j) dnorm(y[1],E[1,j],E[2,j]))) #use conditional emiss
  for (i in 2:n) { # for each position in sequence
    for (j in 1:ns) { # for each X_i
      u <- m[i - 1,] + logP[,j]
      m[i, j] <- max(u) + log(dnorm(y[i],E[1,j],E[2,j]))
      b[i - 1, j] <- which.max(u) #argmax
    }
  }
  # backtrack
  v <- numeric(n)
  v[n] <- which.max(m[n,]) # b[n]
  for (i in (n - 1):1) v[i] <- b[i, v[i + 1]]
  list(m = m, b = b, seq = v)
}

#obtain the MAP estimate X.hat for X
MAP <- viterbi(y,logI,logP,E)
X.hat <- MAP$seq
X.hat
```

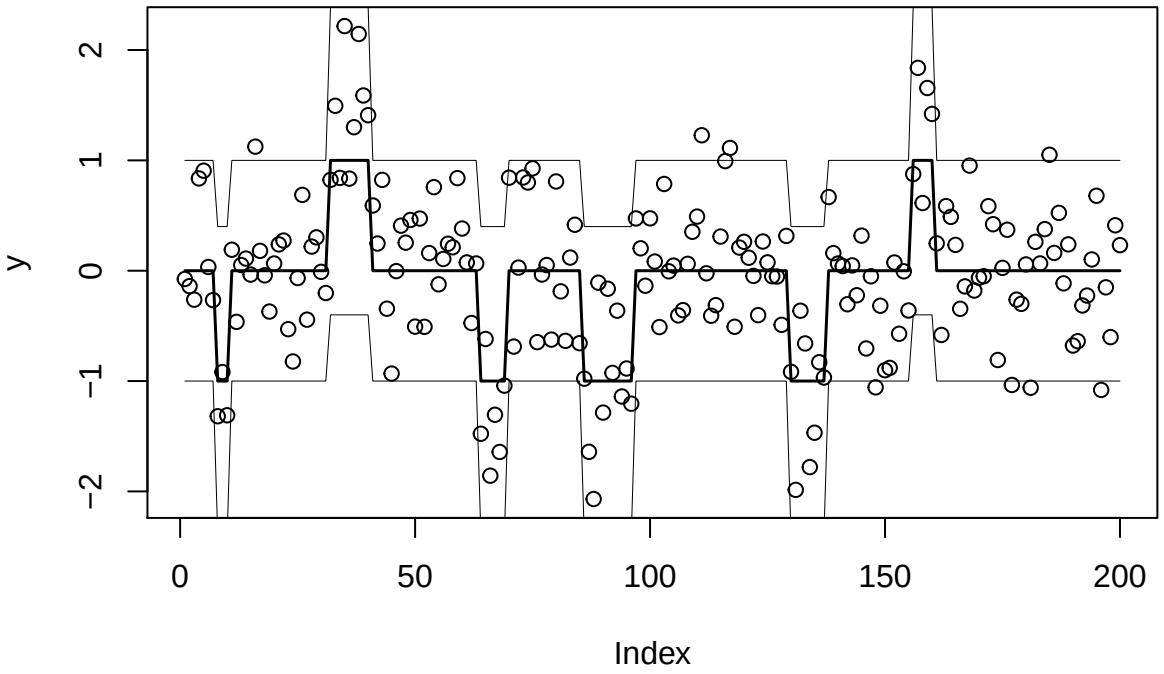
```
##   [1] 2 2 2 2 2 2 2 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3
##  [36] 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 2
##  [71] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2
## [106] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 2 2 2
## [141] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2
## [176] 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
```

```
#calculate the joint probability function, result is in log format
logJointprob <- function(logP, E, logI, X) {
  n <- length(y)
  p <- numeric(n)
  #recurse
  p[1] <- logI[X[1]]
  for (i in 2:n) {
    p[i] <- logP[X[i-1],X[i]] + log(dnorm(y[i],E[1,X[i]],E[2,X[i]]))
  }
  sum(p)
}

#obtain log(P(X.hat|Y)): = log(P(X.hat, Y)) - log(P(Y))
logxy <- logJointprob(logP, E, logI, X.hat)
logPX.Y<- logxy - logPY
logPX.Y
```

```
## [1] -16.35883
```

(c).

```
plot(y)
lines(E[1,X.hat], lwd = 1.5)
lines(E[1,X.hat] + 2 * E[2,X.hat], lwd = .5)
lines(E[1,X.hat] - 2 * E[2,X.hat], lwd = .5)
```



We can see from the figure that X.hat seems to provide a reasonable fit. Because most of points lie between the 95 confidence interval. But noticing that in the regions where index are around 20, 120 and 175, there seem to be differences from X.hat, being amplified.

(d).

```
print("Probability that last probe has a normal copy number given Y is")
```

```
## [1] "Probability that last probe has a normal copy number given Y is"
```

```
exp(f[200,2]-logPY)
```

```
## [1] 0.9586747
```

```
print("Given Y, the last probe is more likely to be in a deleted region instead of in a duplicated regi
```

```
## [1] "Given Y, the last probe is more likely to be in a deleted region instead of in a duplicated reg
```

```r
exp(f[200,1]-f[200,3])
```

```
## [1] 0.3216633
```

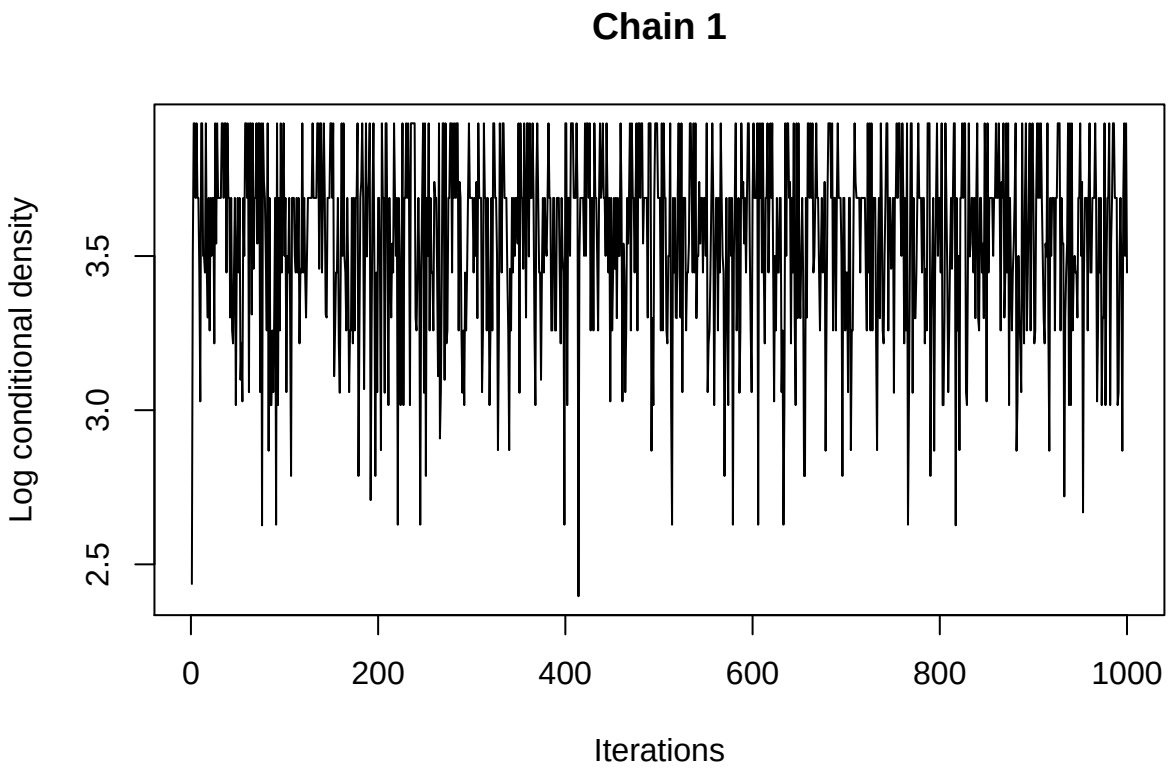
2.(a)

```r
library("coda")

# Compute sum(X_j) that (i,j) is in G
sum.neighbor <- function(X, i){
    if(i == 1){s = X[2]+X[4]}
    else if(i == 2){s <- X[1]+X[3]+X[5]}
    else if(i == 3){s <- X[2]+X[6]}
    else if(i == 4){s <- X[1]+X[5]+X[7]}
    else if(i == 5){s <- X[2]+X[4]+X[6]+X[8]}
    else if(i == 6){s <- X[3]+X[5]+X[9]}
    else if(i == 7){s <- X[4]+X[8]}
    else if(i == 8){s <- X[5]+X[7]+X[9]}
    else s <- X[6] + X[8]
    return(s)
}

# compute sum(X_i*X_j) that (i,j) is in G
sum.total <- function(X){
    sum.total <- 0
    for(i in 1:9){
        sum.total <- X[i] * sum.neighbor(X, i)
    }
    return(sum.total/2.0)
}

# Compute the log proposal distribution
log.prop <- function(J, X, Y, i){
    #Y: data
    prob.1 <- exp(J * 1 * sum.neighbor(X, i) + dnorm(Y[i], mean=2, sd=1, log=TRUE))
    prob.2 <- exp(J * (-1) * sum.neighbor(X, i) + dnorm(Y[i], mean=0.5, sd=0.5, log=TRUE))
    return(prob.1/(prob.1+prob.2))
}

gibbsJ <- function(J, n, t=500){
    # Acceptance
    Y <- c(2,2,2,2,0,0,1,2,1)
    X <- matrix(NA, n, 9)
    X[1, ] <- 2 * (runif(9) < 0.5) - 1
    for(i in 2:n){
        X.new <- X[i-1,]
        for(j in 1:9){
            X[i, j] <- 2 * (runif(1) < log.prop(J, X.new, Y, j)) - 1
            X.new[j] <- X[i,j]
        }
    }
```

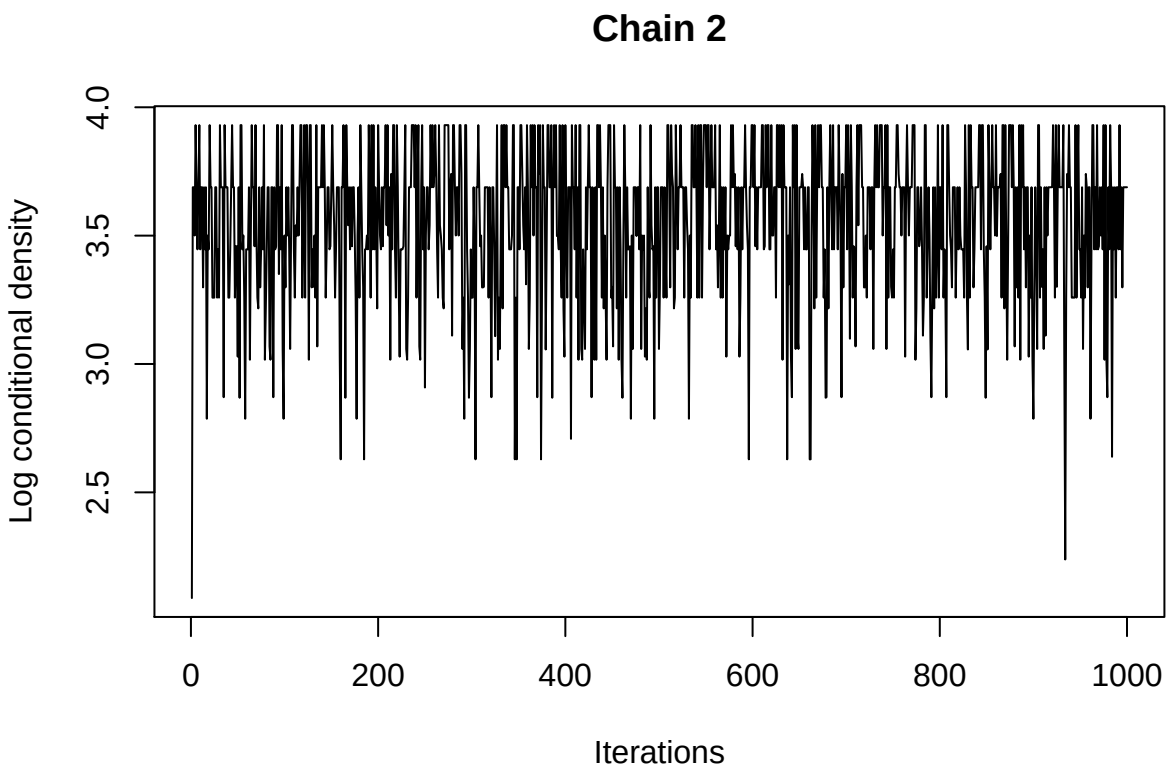
```
    return(list(X_sample_t=X[t,], Xdraws=mcmc(X)))
}
```

(b).

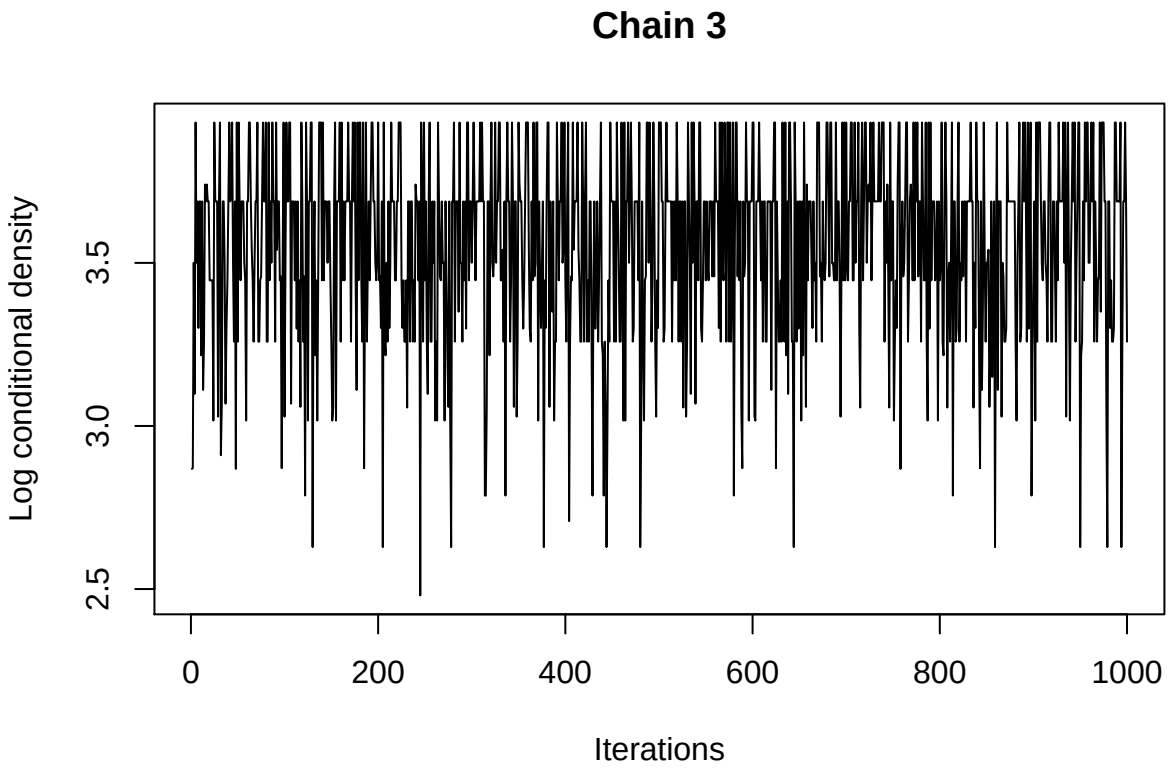
```
param <- matrix(c(0.5, 0.5, 2, 1), 2, 2)
log.cond <- function(J, X, Y){
    n <- nrow(X)
    L.cond <- array(NA, n)
    for(i in 1:n){
       q <- (X[i,] == 1) + 1
       L.cond[i] <- J * sum.total(X[i,]) + sum(dnorm(Y, param[1, q], param[2, q]))
    }
    return(L.cond)
}

Xdraws.1 <- gibbsJ(0.2, 1000)$Xdraws
Xdraws.2 <- gibbsJ(0.2, 1000)$Xdraws
Xdraws.3 <- gibbsJ(0.2, 1000)$Xdraws


L.cond.1 <- mcmc(log.cond(0.2, Xdraws.1, c(2,2,2,2,0,0,1,2,1)))
L.cond.2 <- mcmc(log.cond(0.2, Xdraws.2, c(2,2,2,2,0,0,1,2,1)))
L.cond.3 <- mcmc(log.cond(0.2, Xdraws.3, c(2,2,2,2,0,0,1,2,1)))

# Plot trace
traceplot(L.cond.1, ylab="Log conditional density", main = "Chain 1")
```



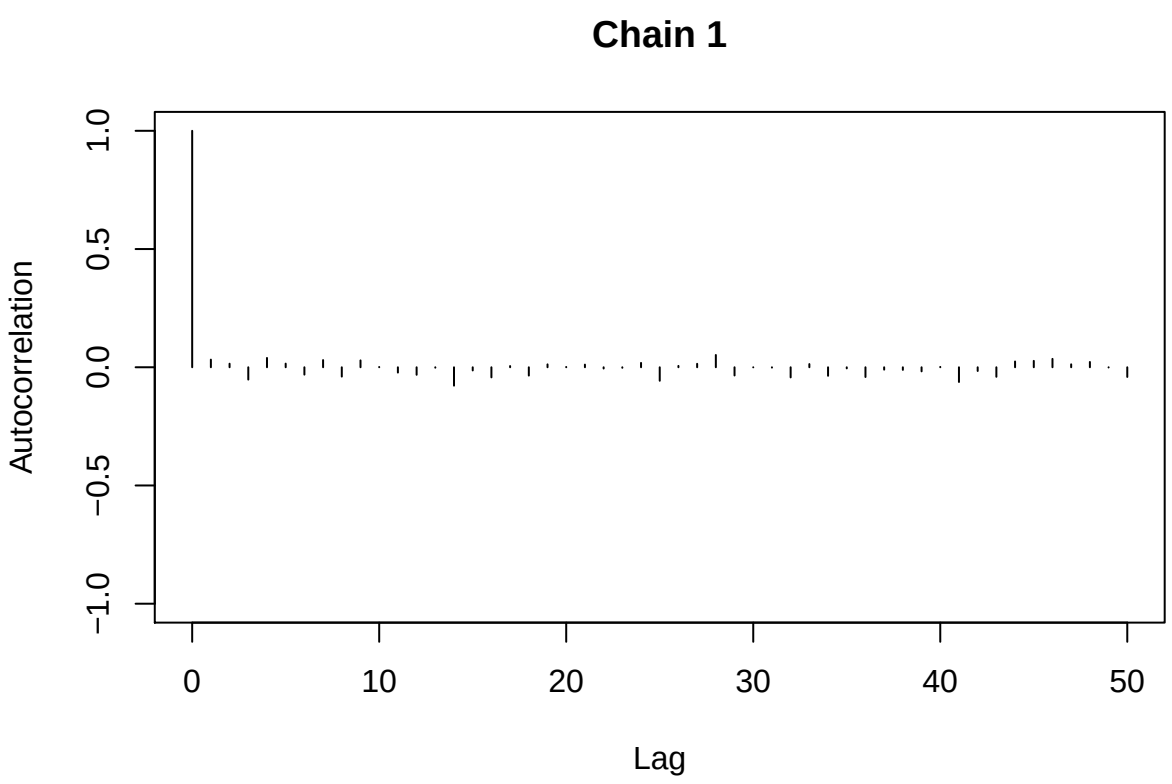
```
traceplot(L.cond.2, ylab="Log conditional density", main = "Chain 2")
```



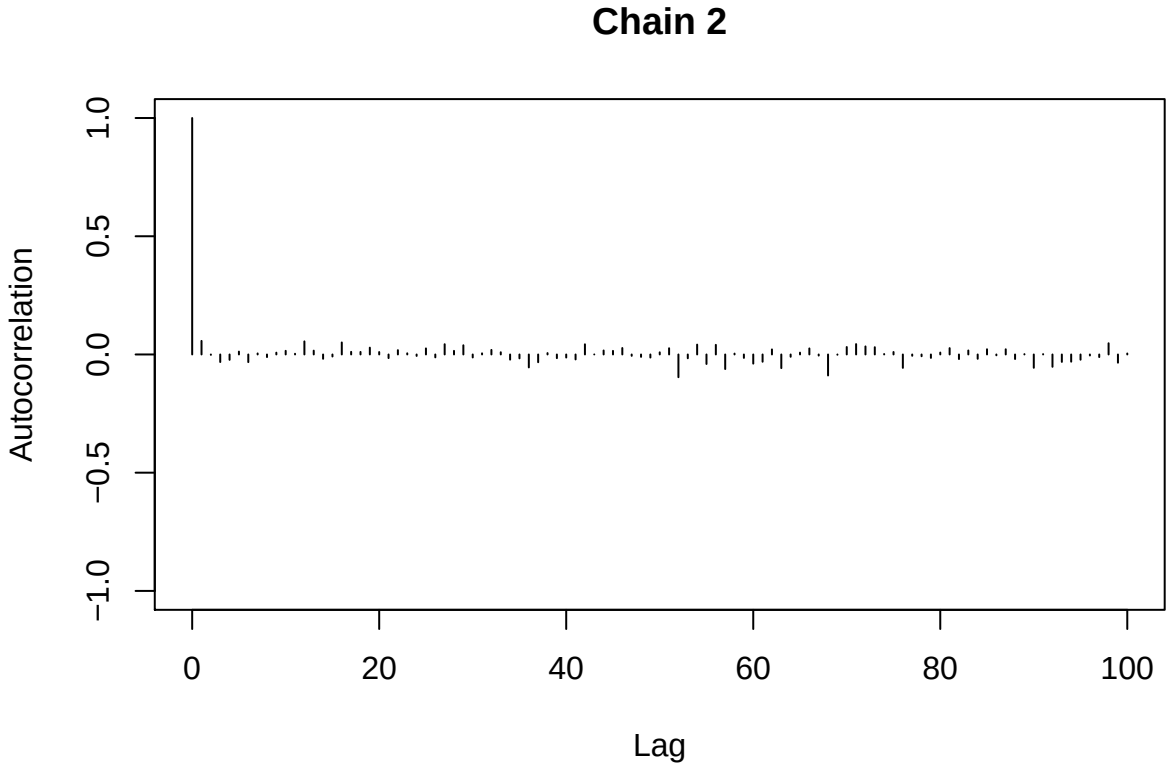

```
traceplot(L.cond.3, ylab="Log conditional density", main = "Chain 3")
```



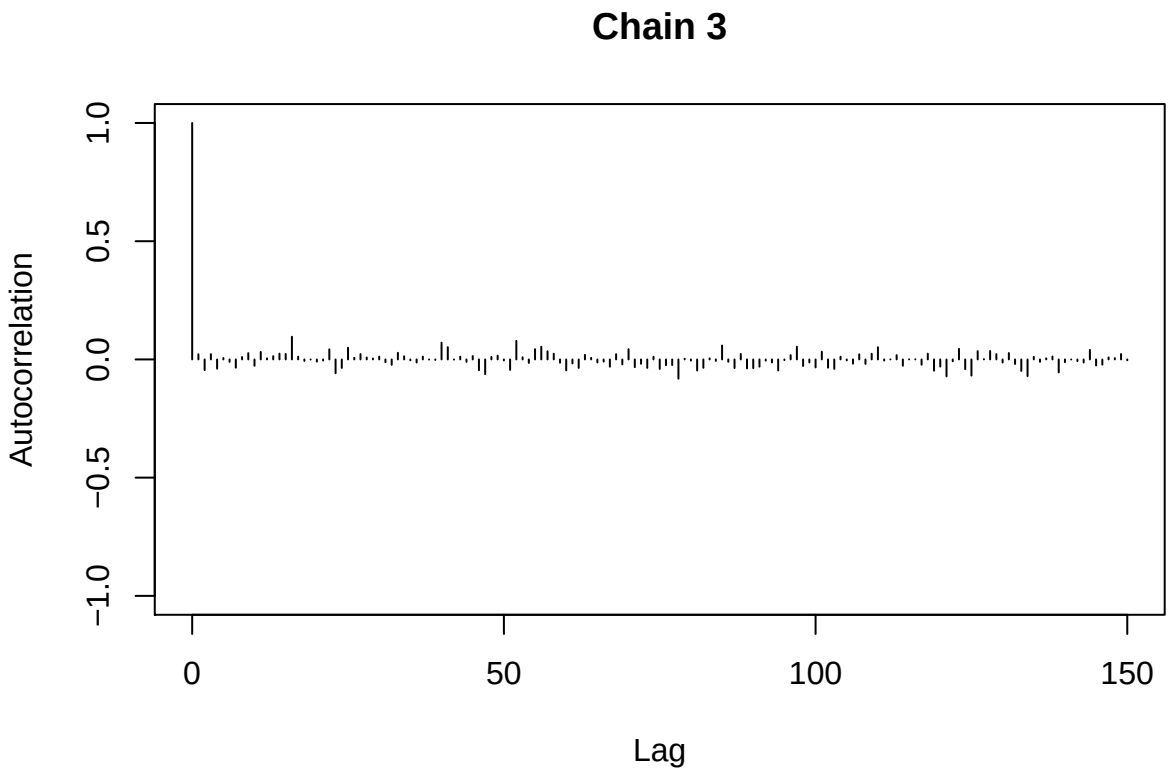
```r
# Plot autocorr
autocorr.plot(L.cond.1, 50, main = "Chain 1")
```




```r
autocorr.plot(L.cond.2, 100, main = "Chain 2")
```



```r
autocorr.plot(L.cond.3, 150, main = "Chain 3")
```




```r
# compute gelman-rubin-brooks
gelman.diag(list(Xdraws.1, Xdraws.2, Xdraws.3))
```

```
## Potential scale reduction factors:
##
##       Point est. Upper C.I.
##  [1,]       1.00       1.00
##  [2,]       1.01       1.01
##  [3,]       1.01       1.01
##  [4,]       1.04       1.05
##  [5,]       1.00       1.00
##  [6,]       1.00       1.00
##  [7,]       1.00       1.00
##  [8,]       1.00       1.00
##  [9,]       1.00       1.00
##
## Multivariate psrf
##
## 1
```

(c).

```r
# MCMC exstimates for P(Xi|Y)
mcmc.estimates <- function(Xdraws){
```

```
  n <- nrow(Xdraws)
  p.X.est <- matrix(NA, 9, 2)
  for(i in 1:9){
    p.X.est[i, 1] <- sum((Xdraws[,i] == 1))/n # P(Xi=1|Y)
    p.X.est[i, 2] <- sum((Xdraws[,i] == -1))/n # P(Xi=-1|Y)
  }
  return(p.X.est)
}

mcmc.est <- mcmc.estimates(Xdraws.1)
print(mcmc.est)
```

```
##        [,1]  [,2]
##  [1,] 0.993 0.007
##  [2,] 0.986 0.014
##  [3,] 0.982 0.018
##  [4,] 0.976 0.024
##  [5,] 0.210 0.790
##  [6,] 0.110 0.890
##  [7,] 0.483 0.517
##  [8,] 0.967 0.033
##  [9,] 0.353 0.647
```

```
# Entropy
sum(L.cond.1) / 1000
```

```
## [1] 3.557758
```