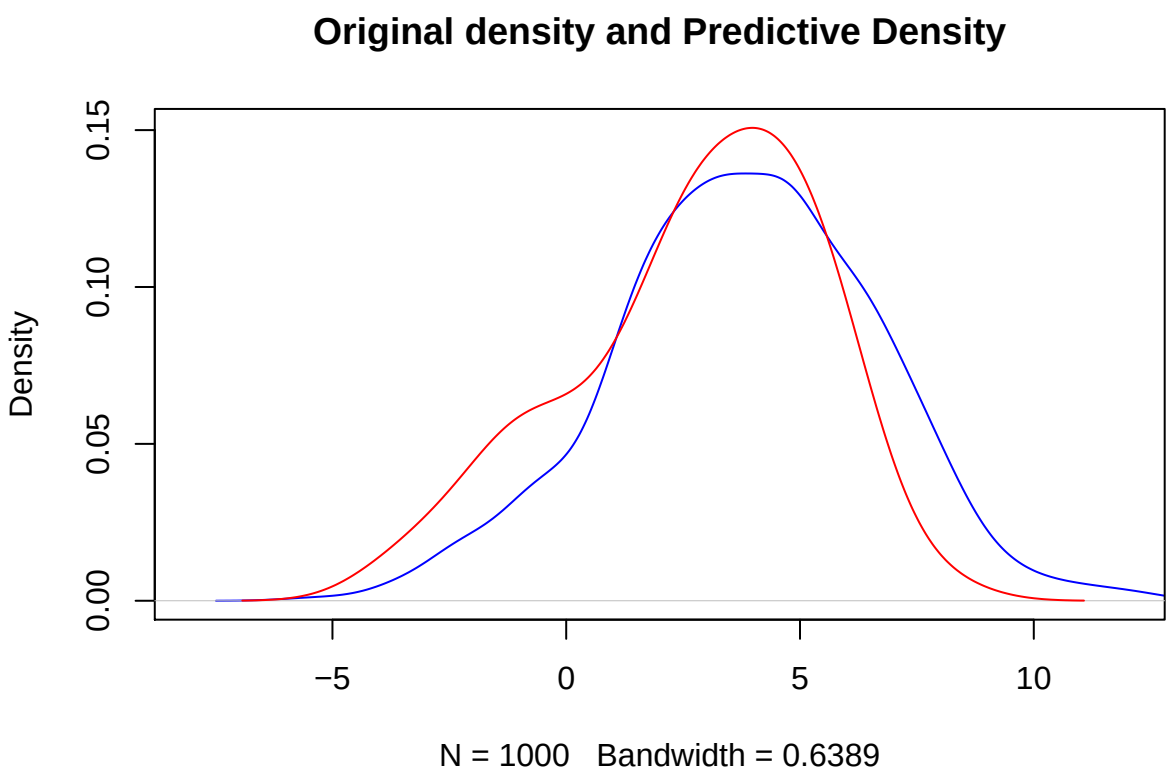
Monte Carlo Methods HW 5 Output

Kylie Taylor

4/18/2019




```
#generated the data
set.seed(90)
N = 100
K <- rep(0,N)
for(h in 1:N){
  if(runif(1)<(1/3)){
    K[h] <- rnorm(1, 0, 2)
  }else{
    K[h] <- rnorm(1, 4, 2)
  }
}

#possibly change bandwidth of density

#set aside memory to store into
t = 10000
w <- lam <- mu1 <- mu2 <- n1 <- n2 <- R <- E1 <- E2 <- A12 <- A21 <- rep(0,t)
d1 <- d2 <- F<- dis <- rep(0,N)
M <- E1 <- E2 <- A12 <- A21 <- rep(0, t)
burnin <- 0.1*t
```

```
#set initial values
lam[1] <- 2
mu1[1] <- 2
mu2[1] <- 3
w[1] <- 1
M[1] <- R[1] <- 1
n1[1] <- n2[1] <- 50
d1[1] <- d2[1] <- 0.5
e1 <- e2 <- 1


for(i in 2:t){
  # MH step for finding M's
    currentx = M[i-1]
    proposedx = (M[i-1] %% 2) + 1
      ### fix all this shit!!
    for(q in 2:N){
      e1 <- w[i-1]*dnorm(K[q-1], mu1[i-1], 1/sqrt(lam[i-1])) + (1-w[i-1])*dnorm(K[q-1], mu2[i-1], 1/sqr
      e2 <- dnorm(K[q-1], mu1[i-1], 1/sqrt(lam[i-1]))*e2
      }

      #A12[i] = (dnorm(rnorm(1,0,100), 0, 100)*e1)/(dnorm(rnorm(1,0,100), mu1[i-1], 100)*e2)
      #A21[i] = (dnorm(mu2[i-1], mu1[i-1], 100)*e2)/(dnorm(mu2[i-1], 0, 100)*e1)     ###make sure A12 m
      A12[i] = 1
      A21[i] =1

    if(currentx < proposedx){

        if(runif(1)<A12[i]){
            M[i] = proposedx  # accept move with probabily min(1,A)

            for(q in 1:N){
              norm <- (w[1]*dnorm(K[q], mu1[1], sqrt(1/lam[1]))) + ((1-w[1])*dnorm(K[q], mu2[1], sqrt(1/
              d1[q] <- (w[1]*dnorm(K[q], mu1[1], sqrt(1/lam[1])))/norm
              d2[q] <- ((1-w[1])*dnorm(K[q], mu2[1], sqrt(1/lam[1])))/norm
              dis[q] <- ifelse(d1[q] >= d2[q], 1, 0)
            }

            n1[i] <- sum(dis)
            n2[i] <- N - n1[i]

            w[i] <- rbeta(1, n1[i] +0.5, n2[i] +0.5)

            DX <- cbind(dis, K)

            DX1 <- DX[DX[,1] == 1,]
            mu1[i] <- rnorm(1, ((lam[i-1]/2)*sum(DX1[,2]))/(0.01 + n1[i]*lam[i-1]/2),  (1/(0.01 + n1[i]*

            DX2 <- DX[DX[,1] == 0,]
            mu2[i] <- rnorm(1, ((lam[i-1]/2)*sum(DX2[,2]))/(0.01 + n2[i]*lam[i-1]/2),  (1/(0.01 + n2[i]*

            lam[i] <- rgamma(1, 0.5 +N/2, 0.5 + 0.5*(sum(DX1[,2] - mu1[i])^2 + sum(DX2[,2] - mu2[i])^2))
```

```
            if(runif(1)<w[i]){
              R[i] = rnorm(1, mu1[i], sqrt(1/lam[i]))
            } else {
              R[i] = rnorm(1, mu2[i], sqrt(1/lam[i]))
            }

        }else{
            M[i] = currentx                 # otherwise "reject" move, and stay where we are

            mu1[i] <- rnorm(1, ((lam[i-1]/2)*sum(K))/(0.01 + N*lam[i-1]/2),  (1/(0.01 + N*lam[i-1]/2)))
            lam[i] <- rgamma(1, 0.5+(N/2), rate = (0.5 + 0.5* sum((K-mu1[i-1])^2)))
            R[i] <- rnorm(1, mu1[i], sqrt(lam[i]))
          }
    }else{

        if(runif(1) < A21[i]){
            M[i] = proposedx  # accept move with probabily min(1,A)

            mu1[i] <- rnorm(1, ((lam[i-1]/2)*sum(K))/(0.01 + N*lam[i-1]/2),  (1/(0.01 + N*lam[i-1]/2)))
            lam[i] <- rgamma(1, 0.5+(N/2), rate = (0.5 + 0.5* sum((K-mu1[i-1])^2)))
            R[i] <- rnorm(1, mu1[i], sqrt(lam[i]))

        }else{
            M[i] = currentx                 # otherwise "reject" move, and stay where we are

            for(q in 1:N){
              norm <- (w[i-1]*dnorm(K[q], mu1[i-1], sqrt(1/lam[i-1]))) + ((1-w[i-1])*dnorm(K[q], mu2[i-
              d1[q] <- (w[i-1]*dnorm(K[q], mu1[i-1], sqrt(1/lam[i-1])))/norm
              d2[q] <- ((1-w[i-1])*dnorm(K[q], mu2[i-1], sqrt(1/lam[i-1])))/norm
              dis[q] <- ifelse(d1[q] >= d2[q], 1, 0)
            }

            n1[i] <- sum(dis)
            n2[i] <- N - n1[i]


            w[i] <- rbeta(1, n1[i] +0.5, n2[i] +0.5)

            DX <- cbind(dis, K)

            DX1 <- DX[DX[,1] == 1,]
            mu1[i] <- rnorm(1, ((lam[i-1]/2)*sum(DX1[,2]))/(0.01 + n1[i]*lam[i-1]/2),  (1/(0.01 + n1[i]*

            DX2 <- DX[DX[,1] == 0,]
            mu2[i] <- rnorm(1, ((lam[i-1]/2)*sum(DX2[,2]))/(0.01 + n2[i]*lam[i-1]/2),  (1/(0.01 + n2[i]*

            lam[i] <- rgamma(1, 0.5 +N/2, 0.5 + 0.5*(sum(DX1[,2] - mu1[i])^2 + sum(DX2[,2] - mu2[i])^2))

            if(runif(1)<w[i]){
              R[i] = rnorm(1, mu1[i], sqrt(1/lam[i]))
            } else {
              R[i] = rnorm(1, mu2[i], sqrt(1/lam[i]))
            }
```

```
            }

        }

}
```