Assessed practical Computational Statistics: Investigating the copy number variants in DNA using a Hidden Markov Model (HMM)

Practical Number: P535

June 10, 2021

Abstract

A Hidden Markov model (HMM) is used to analyze the hybridization intensity (signal) in order to infer on the copy number variants (CNVs). It is shown that the choice of emission density can impact whether outliers are assigned to the correct hidden state or not. The marginal maximum a posterior probability is used to decide from which hidden states the observed signals come from. In the first HMMs that are implemented, the transition matrix is assumed to be known. The EM algorithm is then implemented to estimate the transition matrix, which turns out to be close to the one that was used at the start.

Keywords: Copy number variants; DNA; Hidden Markov Models; Hybridization;

1 Introduction

Copy number variants (CNVs) are segments of DNA that are present in variable copy number relative to the reference genome for that organism. For a given organism there is a typical number of homologous copies expected to be found in any genomic region. However, various genetic mechanisms can lead to the loss or gain of portions of DNA reducing or increasing the number of copies of genes, with implications in many diseases. One can infer on the number of copies by observing a signal (hybridisation intensity) created using microarrays on the targeted genomic region. Indeed, the signal is proportional to the number of copies of that region present in the sample which allows to indirectly measure DNA copy number. In this report, we are interested in segmenting an observed sequence signal into homogeneous regions of constant signal intensity and then to classify these segments into 3 groups: normal number (state 1), higher number (state 2) and lower number (state 3) of copies. Figure 1 below shows the sample data that we are considering in this report.

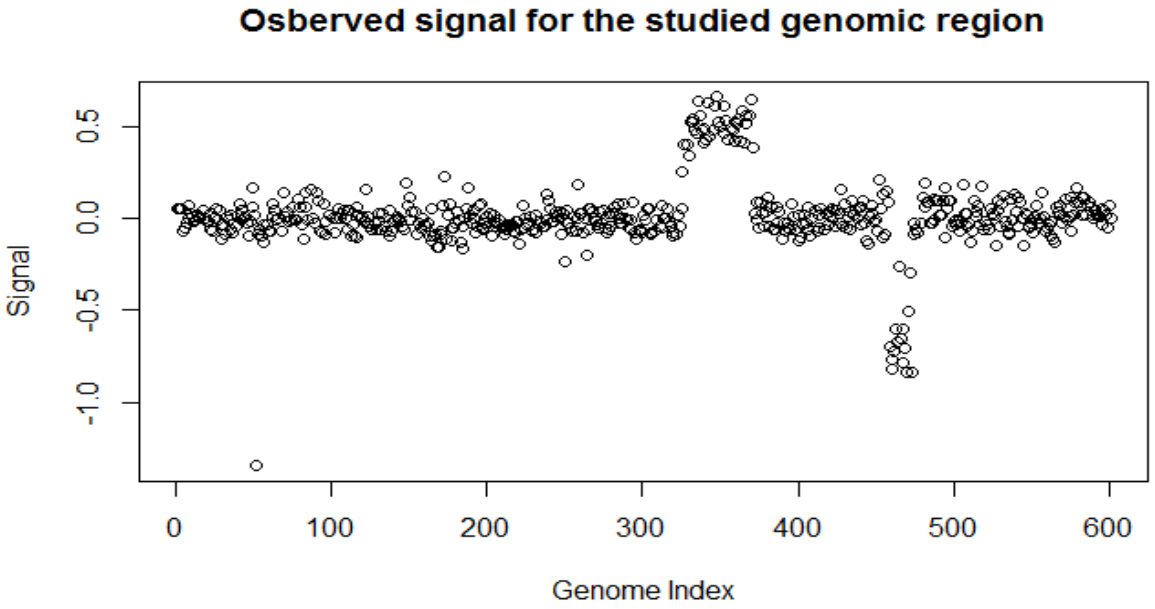

Figure 1: Hybridisation intensity (signal) for the studied region.

One can see from Figure 1 that it is reasonable to classify the signal into the aforementioned groups. Note that there is an odd signal value around index 50, we will want this signal to be classified into the normal number group.
In the first part, we will fit a hidden Markov model to the data using normal emission densities for the 3 hidden states and compute the marginal maximum a posteriori for each observation. We will show that the model misclassifies the outlier into the lower number group and we will then consider a t-distribution emission for state 1 instead. In the second part, we will implement the EM algorithm in order to compute the estimate of the transition matrix. The models are implemented in R and the code is provided in Appendix A.

2 Hidden Markov Model

2.1 Forward and backward recursions

To segment the dataset, we consider a three-state homogeneous hidden Markov model $(X_1, Y_1, ..., X_n, Y_n)$, where X_i are the hidden states that we want to infer and Y_i are the measured signals, $i = 1, ..., T$. The state space is $\chi = 1, 2, 3$ where 1 corresponds to a normal copy number, 2 to an increased copy number, and 3 to a reduced copy number, and the observation space is $Y = \mathbb{R}$.

We are interested in computing the marginal maximum a posteriori estimate of the state for each index, which is given by

$$\hat{x}_t = \arg\max_{k=1,2,3} P(X_t = k | y_{1:T}). \tag{1}$$

We thus want to compute the smoothing probability $P(X_t = k|y_{1:T})$ for all t and all possible states (1,2,3). The smoothing probability is given by

$$\frac{\alpha_t(x_t)\beta_t(x_t)}{\sum_{x\in\chi} \alpha_t(x)\beta_t(x)}, \tag{2}$$

where x_t can be equal to 1,2 or 3, $\alpha_t(x_t) = P(x_t, y_{1:t})$ and $\beta_t(x_t) = P(x_t|y_{1:T})$. The expressions $\alpha_t(x_t)$ and $\beta_t(x_t)$ can be obtained by forward and backward recursions, respectively (see R code in Appendix A), where

$$\alpha_t(x_t) = P(y_t|x_t) \sum_{x-1\in\chi} P(x_t|x_{t-1})\alpha_{t-1}(x_{t-1}), \quad \alpha_0(x_0) = P(x_0), \;\; t = 1, ..., T \tag{3}$$

and

$$\beta_{t-1}(x_{t-1}) = P(y_t|x_t) \sum_{x\in\chi} P(y_t|x_t)P(x_t|x_{t-1})\beta_t(x_t), \quad \beta_T(x_T) = 1, \;\; t = T, ..., 2. \tag{4}$$

In equations (3) and (4), $P(y_t|x_t)$ is called the emission density, $P(x_t|x_{t-1})$ is the transition matrix and $\alpha_0(x_0) = P(x_0)$ is the initial state probability. In our case, the emission densities are normal with $g_1(y) = \mathrm{N}(y; 0, 0.05^2)$, $g_2(y) = \mathrm{N}(y; 0, 5, 0.08^2)$, $g_2(y) = \mathrm{N}(y; -0, 5, 0.08^2)$. The transition matrix $A = P(x_t|x_{t-1})$ is given by

$$A = \begin{bmatrix} 0.99 & 0.005 & 0.005 \\ 0.005 & 0.99 & 0.005 \\ 0.005 & 0.005 & 0.99 \end{bmatrix}$$

and the initial state probability is uniform, so $\alpha_0(x_0) = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$.

2.2 Marginal maximum a posteriori

We are now in position to compute the marginal maximum a posteriori states for each observation. Figure 2 shows the resulting marginal maximum a posteriori hidden states.

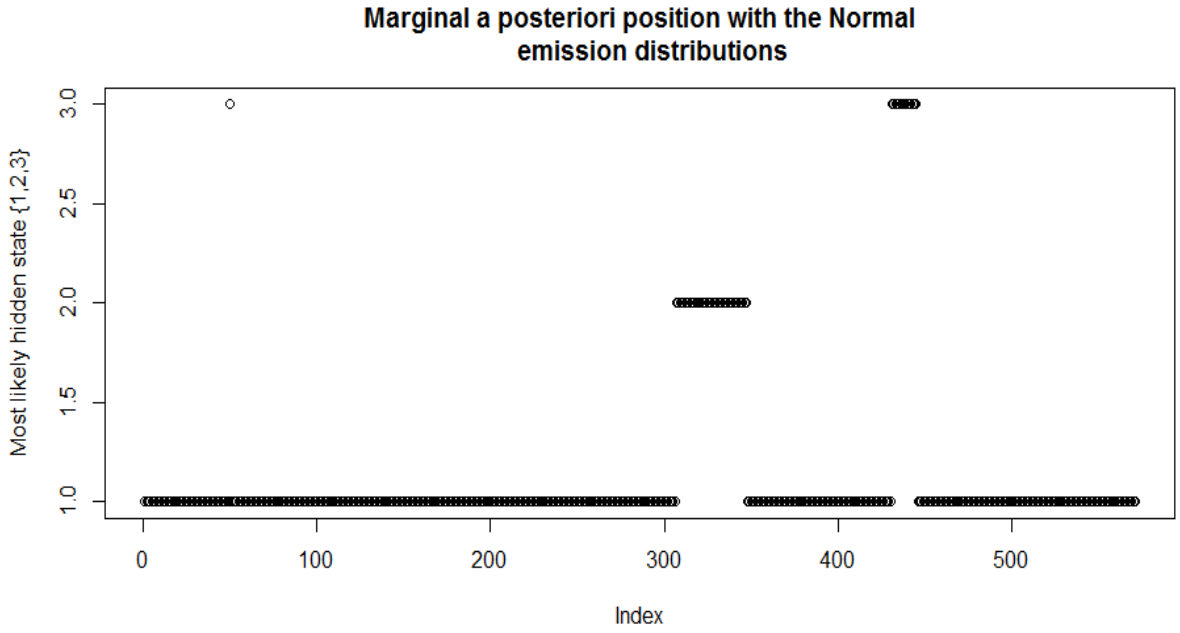

Figure 2: Marginal maximum a posteriori states with normal emission densities.

We can clearly see that the outlier is put into group 3 while it should be put into group 1. We can conclude that our model is not performing well in detecting and classifying the outliers in the correct group. This problem can be solved using a non-standardized t-distribution for the emission density of hidden state 1 instead of the normal density that was used previously. Thus, we now have

$$g_1(y) = \frac{\Gamma(\frac{5}{2})}{0.1s\sqrt{(\pi)}} \left(1 + \frac{y^2}{0.01}\right)^{\frac{5}{2}}.$$

Computing the marginal maximum a posteriori in (1) with this new emission density, we obtain the following segmentation of the data as shown in Figure 3.

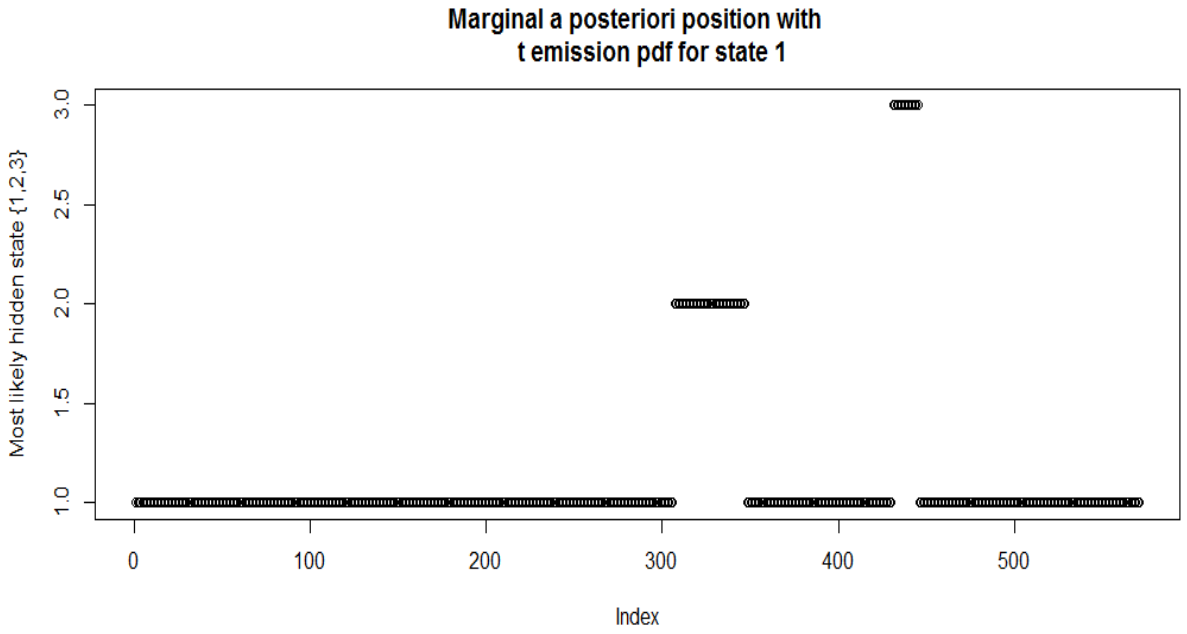

Figure 3: Marginal maximum a posteriori states with the new emission density for hidden state 1.

We can see now that the outlier is classified in the group with a normal number of copies, as it should be. This is because the t-distribution has a heavier tail and can capture outliers. One would like to have a method in order to decide which of the two models is better. The reason we switched to using a t-distribution was to get rid of the outlier. In that sense, and in that particular genome segment, since all the marginal maximum a posteriori hidden states are the same, except for the outlier, one could say that model 2 is better. A first way of assessing the models would be to simply count the number of outliers that are correctly classified in each model. However, to do so one would have to know which points are outliers. This could be done by saying that if we have $x_i = j$ and $x_{i-1} = x_{i+1} = m$, where $m \neq j$, then it would make sense biologically (as it seems that the change in copy number happens on an interval rather than in an isolated instance) to treat x_i as an outlier which was wrongly classified. Furthermore, one model may give more sensible results than the other even if the outputted hidden states are exactly the same. More generally, assuming that the change in copy number does happen on an interval, one could use nonparametric methods to test which model is best. For each model, each observation is assigned a probability vector corresponding to how likely it is that it comes from a given hidden state. Let p_{im}, $m = 1, ...M$ and $i = 1, ..., T$ be the probability vector associated to observation i for model m. For a certain model, after having calculated all the vectors of probabilities, one could take p_i as unknown, and p_j for all $j \neq i$ equal to what was calculated, and take either the moving average of the neighboring indices of i (i.e, taking the average of the vectors of probabilities) or take a weighted average (e.g, kernel estimator), to estimate p_i. Then the process would be repeated for all $i = 1, ..., T$ and compare each of these to the originally computed p_i and use some overall loss measure (quadratic loss for example). We would like to have the smallest loss over all the indices i. Therefore, we could look at both the number of outliers correctly classified and at the total loss for each model in order to decide which model is best.

3 Estimating the transition matrix

In the first part, we computed the marginal maximum a posteriori using the known transition matrix A. Now, we want to start with an initial transition matrix to find an estimate $\hat{A}$ using the expectation-maximization (EM) algorithm. The E-step is as follows:

$$Q(A, A^*) = \boldsymbol{E}[\log P(X_{0:T}, y_{1:T}; A)|y_{1:T}, A^*] = \sum_{i,j\in\chi} \boldsymbol{E}[N_{i,j}|y_{1:T}, A^*] \log A_{i,j} + C, \quad (5)$$

where C is a constant that does not depend on A and

$$N_{i,j} = \sum_{t=1}^{T} \boldsymbol{I}(X_t = j, X_{t-1} = i).$$

Thus, we have

$$\boldsymbol{E}[N_{i,j}|y_{1:T}, A^*] = \sum_{t=1}^{T} P(X_t = j, X_{t-1} = i||y_{1:T}, A^*),$$

where

$$P(X_t = j, X_{t-1} = i||y_{1:T}, A^*) \propto \alpha_{t-1}(x_{t-1})P(y_t|x_t)P(x_t|x_{t-1})\beta_t(x_t).$$

Once we have calculated the E-step we can maximize the expression with respect to $A_{i,j}$ in the M-step. We want to compute

$$A_{i,j}^{(k)} = \text{argmax}_{A_{i,j}} Q(A_{i,j}, A_{i.j}^{(k-1)}). \quad (6)$$

We then iterate and perform in turn the E and M steps (at each iteration the log-likelihood increases) until a certain threshold is met and we stop. We use the R code given in subsection 5.5 in Appendix A to recursively estimate the transition matrix. The initial transition matrix is taken to be uniform with probability $\frac{1}{3}$ to either stay in the same state, move up or move down. One has to make sure that the initial matrix is indeed a valid transition matrix. After running the algorithm we find that the estimated matrix $\hat{A}$ after 5 iterations is (rounded at 3 decimals)

$$\hat{A} = \begin{bmatrix} 0.996 & 0.002 & 0.002 \\ 0.025 & 0.975 & 0.000 \\ 0.067 & 0.000 & 0.933 \end{bmatrix}$$

.

We notice that we almost recover the matrix that we used in the first part with large probabilities in the diagonal and probabilities close to 0 off the diagonals. When we perform the whole analysis with this estimated matrix, we find the same results that when we used the known matrix.

4 Conclusions

We used a homogeneous hidden Markov model to analyze the hybridization intensity observed in a given segment of the genome. We showed that changing one of our emission density made it possible to capture an outlier in the data and proposed a method to decide which HMM is more suitable to the data. After which, we estimated the transition matrix using the EM algorithm, starting with a uniform initial transition matrix. After re-running the whole analysis with the new estimated matrix, the same results were found. A disadvantage of using HMMs is that it may not be obvious how to choose the emission densities (which were assumed to be known) and we may not know the initial state probabilities either. In which case, we may have to also estimate these extra parameters, which can be much more computationally demanding (although we only have 3 hidden states here). Furthermore, using a homogeneous transition matrix may be over simplifying the problem, even though we obtain good results on this particular dataset. Finally, the Markov property of the HMMs may not be appropriate for this type of data since state X_t not only depends on the previous state but also on the ones before.

5 Appendix A

5.1 Loading packages and data

```
# Load the libraries

library(lattice)
library(Rcpp)
library(grid)
library(ggplot2)
library(penalized)
library(parallel)
library(latticeExtra)
library(snow)
library(doSNOW)

# Get the data

source("https://bioconductor.org/biocLite.R")
biocLite("DNAcopy")
library("DNAcopy")

# Few plots of the data

data(coriell)
y.all = coriell$Coriell.05296 # y.all is the whole signal
plot(y.all,xlab="Genome Index", ylab="Signal")

y = y.all[900:1500]
plot(y, xlab="Genome Index", ylab="Signal",
main="Osberved signal for the studied genomic region")
```

5.2 Exercise 1 a)

```
# Forward alpha recursion and beta backward recursion

alpha_recursion = function(y, mu, A, m, s)
{
  K = length(mu)
  T = length(y)
  alpha = matrix(0, nrow=T,ncol=K)
  for (j in 1:K) alpha[1,j] = dnorm(y[1],m[j],s[j]) *sum(A[,j]* mu)
  for (t in 2:T) for (j in 1:K) alpha[t,j] =
  dnorm(y[t],m[j],s[j])*sum(A[,j]* alpha[t-1,])
  return(alpha)
}

beta_recursion = function(y, mu, A, m, s)
{
```

```
  K = length(mu)
  T = length(y)
  beta = matrix(0, nrow=T,ncol=K)
  for (j in 1:K) beta[T,j] = 1
  for (t in T:2) for (i in 1:K) beta[t-1,i] = sum(c(dnorm(y[t],m[1],s[1]),
  dnorm(y[t],m[2],s[2]),dnorm(y[t],m[3],s[3]))*A[i,]* beta[t,])
  return(beta)
}

mu <- c(1/3,1/3,1/3)
A <- matrix(0.005,nrow=3,ncol=3)+diag(3)*(0.99-0.005)
m <- c(0,0.5,-0.5)
s <- c(0.05,0.08,0.08)
y <- y[!is.na(y)]
```

5.3 Exercise 1 b)

```
alpha <- alpha_recursion(y=y, mu=mu, A=A, m=m, s=s)
beta <- beta_recursion(y=y, mu=mu, A=A, m=m, s=s)
normalisation <- 0
numerator <- matrix(0, nrow = length(y), ncol = length(mu))
marginal_posteriori <- 0

# Compute the normalization term

for (t in 1:length(y))
{
  normalisation[t] <- sum(alpha[t,]*beta[t,])
}

# We find that all the normalisation values have the same value for the computer
# (this is the highest number the computer can assimilate)
# So we do not divide by the normalisation.

# Compute the numerator of the smoothing probability

for (t in 1:length(y))
{
  for (k in 1:length(mu))
  {
   numerator[t,k] <- (alpha[t,k]*beta[t,k])
  }
  marginal_posteriori[t] <- which.max(numerator[t,])
}

# Plot the marginal a posteriori states at all the observations

Nstate_1 <- length(which(marginal_posteriori==1))
Nstate_2 <- length(which(marginal_posteriori==2))
```

```
Nstate_3 <- length(which(marginal_posteriori==3))

# Plot the marginal maximum a posteriori states

plot(marginal_posteriori, ylab='Most likely hidden state {1,2,3}',
main='Marginal a posteriori position with the Normal
    emission distributions')
```

5.4 Exercise 2 part 1

```
# Define the new vector of emission densities
# with the t distribution for state 1

pdf <- function(tt,j,y)
{
  if(j==1) {return((gamma(5/2)/(sqrt(pi)*0.1))*(1+(y[tt]^2)/(0.01))^(-5/2))}
  if(j==2) {return(dnorm(y[tt],0.5,0.08))}
  if(j==3) {return(dnorm(y[tt],-0.5,0.08))}
}

# alpha recursion

alpha_recursion = function(y, mu, A)
{
  K = length(mu)
  T = length(y)
  alpha = matrix(0, nrow=T,ncol=K)
  for (j in 1:K) alpha[1,j] = pdf(tt=1,j=j,y=y) *sum(A[,j]* mu)
  for (t in 2:T) for (j in 1:K) alpha[t,j] =
  pdf(tt=t,j=j,y=y)*sum(A[,j]* alpha[t-1,])
  return(alpha)
}

# beta recursion
beta_recursion = function(y, mu, A)
{
  K = length(mu)
  T = length(y)
  beta = matrix(0, nrow=T,ncol=K)
  for (j in 1:K) beta[T,j] = 1
  for (t in T:2) for (i in 1:K) beta[t-1,i] =
  sum(c((gamma(5/2)/(sqrt(pi)*0.1))*(1+
  (y[t]^2)/(0.01))^(-5/2),dnorm(y[t],m[2],s[2]),
  dnorm(y[t],m[3],s[3]))*A[i,]* beta[t,])
  return(beta)
}


mu <- c(1/3,1/3,1/3)
```

```r
A <- matrix(0.005,nrow=3,ncol=3)+diag(3)*(0.99-0.005)
m <- c(0,0.5,-0.5)
s <- c(0.05,0.08,0.08)
y <- y[!is.na(y)]

alpha <- alpha_recursion(y=y, mu=mu, A=A)
beta <- beta_recursion(y=y, mu=mu, A=A)
normalisation <- 0
numerator <- matrix(0, nrow = length(y), ncol = length(mu))
marginal_posteriori <- 0

for (t in 1:length(y))
{
  normalisation[t] <- sum(alpha[t,]*beta[t,])
}

# We find that all the normalisation values have the
# same value for the computer.
# So we do not divide by the normalisation.

for (t in 1:length(y))
{
  for (k in 1:length(mu))
  {
    numerator[t,k] <- (alpha[t,k]*beta[t,k])
  }
  marginal_posteriori[t] <- which.max(numerator[t,])
}

# Plot the marginal maximum a posteriori states

plot(marginal_posteriori, ylab='Most likely hidden state {1,2,3}',
main='Marginal a posteriori position with
     t emission pdf for state 1')

# Get the counts of the number of times the hidden states occur

NNstate_1 <- length(which(marginal_posteriori==1))
NNstate_2 <- length(which(marginal_posteriori==2))
NNstate_3 <- length(which(marginal_posteriori==3))

# Compare the counts for the 2 models

c(Nstate_1,Nstate_2,Nstate_3)
c(NNstate_1,NNstate_2,NNstate_3)
```

5.5 Exercise 2 part 2

```
# Compute the probabilities to be used in the EM algorithm

proba_xt_xt_1 <- function(y, alpha, beta, mu, A){
  K = length(mu)
  T = length(y)
  proba_xt_xt_1 = matrix(0, nrow = 3*K, ncol = T)
  for (t in 2:T){
    if(is.na(pdf(tt=t,j=1,y=y))){
      for (j in 1:K)  proba_xt_xt_1[j, t] = alpha[t-1,1]*A[1,j]*beta[t,j]
      for (j in (K+1):(2*K)) proba_xt_xt_1[j, t] = alpha[t-1,2]*A[2,j-
      K]*beta[t,j-K]
      for (j in (2*K+1):(3*K)) proba_xt_xt_1[j, t] = alpha[t-1,3]*A[3,j-
      2*K]*beta[t,j-2*K]
    }
    else{
      for (j in 1:K)  proba_xt_xt_1[j, t] = alpha[t-
      1,1]*pdf(tt=t,j=j,y=y)*A[1,j]*beta[t,j]
      for (j in (K+1):(2*K)) proba_xt_xt_1[j, t] = alpha[t-
      1,2]*pdf(tt=t,j=(j-K),y=y)*A[2,j-K]*beta[t,j-K]
      for (j in (2*K+1):(3*K)) proba_xt_xt_1[j, t] = alpha[t-
      1,3]*pdf(tt=t,j=(j-2*K),y=y)*A[3,j-2*K]*beta[t,j-2*K]
    }
  }
  return(proba_xt_xt_1)
}

# EM algorithm

EM_algorithm <- function(y, mu, epsilon, A_start){
  K = length(mu)
  T = length(y)
  A_hat = A_start
  alpha = alpha_recursion(y, mu, A_hat)
  beta = beta_recursion(y, mu, A_hat)
  proba_xt_xt_1 = proba_xt_xt_1(y, alpha, beta, mu, A_hat)
  N = rowSums(proba_xt_xt_1)
  A_hat = matrix(0, nrow = 3, ncol = 3)
  for (j in 1:K) for (i in 1:K) A_hat[i, j] = N[i+3*(j-1)]/(N[i] + N[i+3]
  + N[i+6])
  likelihood = sum(alpha[T,])
  print(likelihood)
  print(A_hat)
  n=1

  repeat{
    alpha = alpha_recursion(y, mu, A_hat)
    beta = beta_recursion(y, mu, A_hat)
```

```
    proba_xt_xt_1 = proba_xt_xt_1(y, alpha, beta, mu, A_hat)
    N = rowSums(proba_xt_xt_1)
    A_hat = matrix(0, nrow = 3, ncol = 3)
    for (j in 1:K) for (i in 1:K) A_hat[i, j] = N[i+3*(j-1)]/(N[i] +
    N[i+3] + N[i+6])
    new.likelihood = sum(alpha[T,])
    print(new.likelihood)
    print(A_hat)

    if(abs(new.likelihood - likelihood) < likelihood*epsilon){
      return(A_hat)
      break
    }
    else{
      likelihood = new.likelihood
    }
  }
}

# pick a starting matrix and use the EM to estimate the transition matrix
# (make sure that it is a valid transition matrix)

AA <- matrix(0.5,nrow=3,ncol=3)

EM_algorithm(y=y, mu=mu, epsilon=0.001, A_start=AA)
```