Chapter 4 - Solutions to exercises

Exercises: 5,6

Exercise 5

The third term of equation (4.12)

$$\sum_{j=1}^{m}\sum_{t=1}^{T} u_j(t)\log \Pr(X_t = x_t|C_t = j) \tag{4.12}$$

in Zucchini09 only depends on the state dependent parameters (here with a slightly different notation). Thus, in estimating the state dependent parameters the maximum of this term must be found.

a)

In the case of a binomial-HMM we have

$$\Pr(X_t = x_t|C_t = j) = \binom{n_t}{x_t} p_j^{x_t}(1-p_j)^{n_t-x_t}.$$

To find the value of p_j which maximizes (4.12) we take derivative of $\log \Pr(X_t = x_t|C_t = j)$ with respect to p_j:

$$\begin{aligned}
&= \frac{\partial}{\partial p_j}\left[\log \Pr(X_t = x_t|C_t = j)\right] \\
&= \frac{\partial}{\partial p_j}\log\left[\binom{n_t}{x_t} p_j^{x_t}(1-p_j)^{n_t-x_t}\right] \\
&= \frac{\partial}{\partial p_j}\left[x_t\log p_j + (n_t - x_t)\log(1-p_j)\right] \\
&= \frac{x_t}{p_j} - \frac{n_t - x_t}{1-p_j} \\
&= \frac{x - n_t p_j}{p_j(1-p_j)}.
\end{aligned}$$

Inserting into (4.12) and equating to zero we get

$$
\begin{aligned}
0 &= \sum_{t=1}^{T} u_j(t) \frac{x - n_t p_j}{p_j(1-p_j)} \\
0 &= \sum_{t=1}^{T} u_j(t) x_t - p_j \sum_{t=1}^{T} u_j(t) n_t \\
p_j &= \frac{\sum_{t=1}^{T} u_j(t) x_t}{\sum_{t=1}^{T} u_j(t) n_t}.
\end{aligned}
$$

b)

In the case of an exponential-HMM we have

$$p(x_t|C_t = j) = \lambda_j e^{-\lambda_j x_t},$$

note that for a continuous random variable we use the probability *density* function instead of the probability mass function. Differentiating gives

$$
\begin{aligned}
&= \frac{\partial}{\partial \lambda_j} \left[\log(\lambda_j e^{-\lambda_j x_t}) \right] \\
&= \frac{\partial}{\partial \lambda_j} \left[\log \lambda_j - \lambda_j x_t) \right] \\
&= \frac{1}{\lambda_j} - x_t.
\end{aligned}
$$

Inserting into (4.12) and equating to zero we get

$$
\begin{aligned}
0 &= \sum_{t=1}^{T} u_j(t) \left(\frac{1}{\lambda_j} - x_t \right) \\
0 &= \sum_{t=1}^{T} u_j(t) \frac{1}{\lambda_j} - \sum_{t=1}^{T} u_j(t) x_t \\
\lambda_j &= \frac{\sum_{t=1}^{T} u_j(t)}{\sum_{t=1}^{T} u_j(t) x_t}.
\end{aligned}
$$

Exercise 6

For the normal distribution use the results on p.67 in Zucchini09 to get the modified functions:

```
# Chapter 4, R-code for exercise 6, mwp 31/1-2011
statdist <- function(gamma){
  m = dim(gamma)[1]
  matrix(1,1,m) %*% solve(diag(1,m) - gamma + matrix(1,m,m))
}
norm.HMM.generate_sample <-
 function(n,m,mu,sigma,gamma,delta=NULL)
{
 if(is.null(delta))delta<-solve(t(diag(m)-gamma+1),rep(1,m))
 mvect <- 1:m
 state <- numeric(n)
 state[1] <- sample(mvect,1,prob=delta)
 for (i in 2:n)
   state[i]<-sample(mvect,1,prob=gamma[state[i-1],])
 x <- rnorm(n,mu[state],sigma[state])
 x
}
norm.HMM.lalphabeta<-function(x,m,mu,sigma,gamma,delta=NULL)
{
 if(is.null(delta))delta<-solve(t(diag(m)-gamma+1),rep(1,m))
 n          <- length(x)
 lalpha     <- lbeta<-matrix(NA,m,n)
# allprobs   <- outer(x,lambda,dpois)                    # <- OLD
 allprobs   <- matrix(NA,n,m)                            # <- NEW
 for(i in 1:m){ allprobs[,i] <- dnorm(x,mu[i],sigma[i])}  # <- NEW
 foo        <- delta*allprobs[1,]
 sumfoo     <- sum(foo)
 lscale     <- log(sumfoo)
 foo        <- foo/sumfoo
 lalpha[,1] <- log(foo)+lscale
 for (i in 2:n)
   {
   foo        <- foo%*%gamma*allprobs[i,]
   sumfoo     <- sum(foo)
   lscale     <- lscale+log(sumfoo)
   foo        <- foo/sumfoo
   lalpha[,i] <- log(foo)+lscale
   }
 lbeta[,n]  <- rep(0,m)
 foo        <- rep(1/m,m)
 lscale     <- log(m)
 for (i in (n-1):1)
   {
   foo        <- gamma%*%(allprobs[i+1,]*foo)
   lbeta[,i]  <- log(foo)+lscale
   sumfoo     <- sum(foo)
   foo        <- foo/sumfoo
   lscale     <- lscale+log(sumfoo)
   }
```

```
 list(la=lalpha,lb=lbeta)
}

norm.HMM.EM <- function(x,m,mu,sigma,gamma,delta,maxiter=1000,tol=1e-6,...)
{
 n              <- length(x)
# lambda.next    <- lambda                                          # <- OLD
 mu.next        <- mu                                              # <- NEW
 sigma.next     <- sigma                                           # <- NEW
 gamma.next     <- gamma
 delta.next     <- delta
 for (iter in 1:maxiter)
   {
#   lallprobs    <- outer(x,lambda,dpois,log=TRUE)                  # <- OLD
   lallprobs <- matrix(NA,n,m)                                     # <- NEW
   for(i in 1:m){ lallprobs[,i] <- log(dnorm(x,mu[i],sigma[i]))}   # <- NEW
#  fb  <-  pois.HMM.lalphabeta(x,m,lambda,gamma,delta=delta)       # <- OLD
   fb  <-  norm.HMM.lalphabeta(x,m,mu,sigma,gamma,delta=delta)     # <- NEW
   la  <-  fb$la
   lb  <-  fb$lb
   c   <-  max(la[,n])
   llk <- c+log(sum(exp(la[,n]-c)))
   for (j in 1:m)
   {
     for (k in 1:m)
     {
       gamma.next[j,k] <- gamma[j,k]*sum(exp(la[j,1:(n-1)]+
                          lallprobs[2:n,k]+lb[k,2:n]-llk))
     }
#   lambda.next[j] <- sum(exp(la[j,]+lb[j,]-llk)*x)/          # <- OLD
#                     sum(exp(la[j,]+lb[j,]-llk))             # <- OLD
     mu.next[j]    <- sum(exp(la[j,]+lb[j,]-llk)*x)/          # <- NEW
                      sum(exp(la[j,]+lb[j,]-llk))             # <- NEW
     sigma.next[j] <- sqrt(sum(exp(la[j,]+lb[j,]-llk)         # <- NEW
                               *(x-mu.next[j])^2)/            # <- NEW
                      sum(exp(la[j,]+lb[j,]-llk)))            # <- NEW
   }
   gamma.next <- gamma.next/apply(gamma.next,1,sum)
   delta.next <- exp(la[,1]+lb[,1]-llk)
   delta.next <- delta.next/sum(delta.next)
   crit       <- sum(abs(mu-mu.next)) +                        # <- NEW
                 sum(abs(sigma-sigma.next)) +                   # <- NEW
              # sum(abs(lambda-lambda.next)) +                  # <- OLD
                 sum(abs(gamma-gamma.next)) +
                 sum(abs(delta-delta.next))
   if(crit<tol)
     {
     np     <- m*m+m-1
     AIC    <- -2*(llk-np)
     BIC    <- -2*llk+np*log(n)
     return(list(mu=mu,sigma=sigma,gamma=gamma,delta=delta,
           mllk=-llk,AIC=AIC,BIC=BIC))
     }
   mu         <- mu.next      # <- NEW
```

```
   sigma      <- sigma.next   # <- NEW
#  lambda     <- lambda.next  # <- OLD
   gamma      <- gamma.next
   delta      <- delta.next
   }
 print(paste("No convergence after",maxiter,"iterations"))
 NA
}

# case m=2
m= 2
mu = c(-3,3)
sigma = c(1,2)
gamma= rbind(c(0.9,0.1),c(0.1,0.9))
delta= statdist(gamma)
n=200

# Generate synthetic data
x <- norm.HMM.generate_sample(n,m,mu,sigma,gamma,delta)

# Fit normal-HMM with the EM algorith,
res <- norm.HMM.EM(x,m,mu,sigma,gamma,delta)
```

Note that the new functions are tested by simulating data and then reestimating the parameters. The output of the above script is

```
> res
$mu
[1] -3.145358  3.252831
$sigma
[1] 0.943199 2.016094
$gamma
          [,1]      [,2]
[1,] 0.9024413 0.0975587
[2,] 0.1324828 0.8675172
$delta
[1] 7.191168e-237  1.000000e+00
$mllk
[1] 405.9208
$AIC
[1] 821.8417
$BIC
[1] 838.3333
```

The output provides strong evidence that the functions work as hoped since the estimated parameters are close to the true parameters used to generate the data.

Implementing the above functions for the **binomial** distribution use the results of exercise 5.a while assuming that x_t and n_t are observed.

Implementing the above functions for the **exponential** distribution use the results of exercise 5.b while assuming that x_t is observed.