

# MATH 545: INTRODUCTION TO TIME SERIES ANALYSIS

## WINTER 2025

### HIDDEN MARKOV MODELS

In a form of state space model, the state equation can be replaced by a Markov chain model: if  $\{X_t\}$  can take values on  $\{1, 2, \dots, S\}$  say, then for  $t > 1$

$$p(x_t | x_{1:(t-1)}) \equiv p(x_t | x_{t-1})$$

where

$$\Pr[X_t = l | X_{t-1} = k] = p_{kl}$$

The  $S \times S$  transition matrix  $\mathbf{P}$ , along with the (marginal) distribution of  $X_1$  ( $\Pr[X_1 = k], k = 1, \dots, S$ ) characterize the nature of the state evolution. Note that  $\mathbf{P}$  is a stochastic matrix whose rows sum to 1. Under mild conditions, the chain has stationary (or equilibrium) distribution,  $\pi$ , defined as the solution of

$$\pi = \pi \mathbf{P}$$

where  $\pi$  represents the ‘long-run’ behaviour of  $\{X_t\}$ .

All of the previous types of calculation

- filtering  $p(x_t | y_{1:t})$
- prediction  $p(x_t | y_{1:(t-1)})$
- smoothing  $p(x_t | y_{1:n})$
- parameter estimation

are required. The main difference is that the filtering, prediction and smoothing distributions are **discrete**, and therefore somewhat more straightforward to compute. The observation variable  $y_t$  can be either discrete or continuous.

**Example:** Suppose for  $t = 1, 2, \dots, n$

$$Y_t = \mu(x_t) + Z_t$$

and  $\{X_t\}$  follows a Markov chain with transition matrix

$$\mathbf{P} = \begin{bmatrix} 0.3 & 0.7 \\ 0.4 & 0.6 \end{bmatrix}$$

and the distribution of  $X_1$  is the stationary distribution,  $\pi$ , of the chain defined as the solution of

$$\pi = \pi \mathbf{P}$$

with  $\pi$  computed as

$$[1, 1]^\top \pi = \lim_{t \rightarrow \infty} \mathbf{P}^t = \begin{bmatrix} 4/11 & 7/11 \\ 4/11 & 7/11 \end{bmatrix}$$

so  $\pi = (4/11, 7/11)$ . Suppose

$$\mu(x) = \begin{cases} \mu_1 & x = 1 \\ \mu_2 & x = 2 \end{cases}$$

and  $\{Z_t\} \sim \mathcal{N}(0, 1)$ , IID.

```

n<-500
S<-2
P<-matrix(c(3,4,7,6)/10,2,2)
mu<-c(2,-1)
Ptmp<-P
for(i in 1:100){Ptmp <- Ptmp %*% P}
Ptmp

##           [,1]      [,2]
## [1,] 0.3636364 0.6363636
## [2,] 0.3636364 0.6363636

Pi<-Ptmp[1,]
c(4/11,7/11)

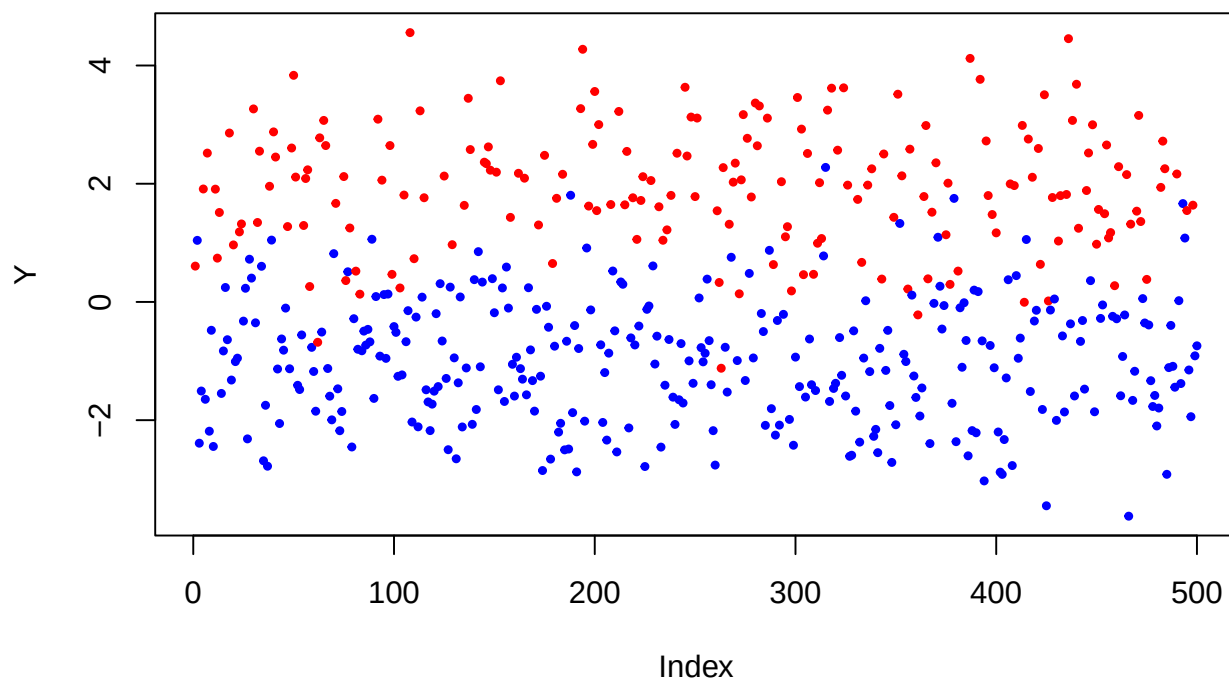
## [1] 0.3636364 0.6363636

set.seed(324)
X<-rep(0,n)
X[1]<-sample(c(1:S),size=1,prob=Pi)
for(i in 2:n){X[i]<-sample(c(1:S),size=1,prob=P[X[i-1],])}
X[1:20]

## [1] 1 2 2 2 1 2 1 2 2 2 1 1 1 2 2 2 2 1 2 1

Y<-rnorm(n,mu[X],1)
par(mar=c(4,4,2,2))
cvec<-c('red','blue')
plot(Y,pch=19,cex=0.5,col=cvec[X])

```



Observations from state 1 with  $\mu = 2$  plotted in red, observations from state 2 with  $\mu = -1$  plotted in blue.

## Key calculations:

- *Filtering*: By Bayes theorem

$$\begin{aligned}
 p(x_t|y_{1:t}) &= p(x_t|y_t, y_{1:(t-1)}) \\
 &= \frac{p(y_t|x_t, y_{1:(t-1)})p(x_t|y_{1:(t-1)})}{p(y_t|y_{1:(t-1)})} \\
 &= \frac{p(y_t|x_t)p(x_t|y_{1:(t-1)})}{\sum_{x=1}^S p(y_t|x)p(x|y_{1:(t-1)})} \\
 &\propto p(y_t|x_t)p(x_t|y_{1:(t-1)})
 \end{aligned}$$

This is a discrete distribution on  $\{1, \dots, S\}$  that we can compute exactly (at least numerically)

Note also that the denominator is the prediction distribution for  $y_t$  given  $y_{1:(t-1)}$

$$p(y_t|y_{1:(t-1)}) = \sum_{x=1}^S p(y_t|x)p(x|y_{1:(t-1)})$$

which can be used to compute the likelihood

$$p(y_1, \dots, y_n) = p(y_1)p(y_2|y_1)p(y_3|y_1, y_2) \dots p(y_n|y_1, y_2, \dots, y_{n-1})$$

- *Prediction*: By the theorem of total probability

$$p(x_{t+1}|y_{1:t}) = \sum_{x=1}^S p(x_{t+1}|x)p(x|y_{1:t})$$

- $p(x_{t+1}|x)$  is determined the  $x$ th row of  $\mathbf{P}$ , for each  $x = 1, 2, \dots, S$
- $p(x|y_{1:t})$  is the previous filtering distribution.

- *Smoothing*: By the theorem of total probability, we can compute *backwards*  $t = n, n-1, \dots, 1$ ,

$$p(x_t|y_{1:n}) = \sum_{x_{t+1}=1}^S p(x_t|x_{t+1}, y_{1:n})p(x_{t+1}|y_{1:n})$$

$p(x_{t+1}|y_{1:n})$  is the previous smoothing distribution. For the first term

$$\begin{aligned}
 p(x_t|x_{t+1}, y_{1:n}) &\equiv p(x_t|x_{t+1}, y_{1:t}) \\
 &= \frac{p(x_t|y_{1:t})p(x_{t+1}|x_t, y_{1:t})}{p(x_{t+1}|y_{1:t})} \\
 &= \frac{p(x_t|y_{1:t})p(x_{t+1}|x_t)}{p(x_{t+1}|y_{1:t})}
 \end{aligned}$$

which can be computed from the filtering and prediction distributions.

```

sig<-1
pX.filter<-pX.predict<-pX.smooth<-matrix(0,nrow=n,ncol=S)
pX.predict[1,]<-Pi
pX.filter[1,]<-dnorm(Y[1],mu,sig)*Pi
Norm<-sum(pX.filter[1,])
pX.filter[1,]<-pX.filter[1,]/Norm

LogLike<-log(Norm)

Xpred<-Xsmooth<-rep(0,n)
Xpred[1]<-which.max(pX.filter[1,])

for(i in 2:n){
  pX.predict[i,]<- colSums(P*pX.filter[i-1,])
  pX.filter[i,]<-dnorm(Y[i],mu,sig)*pX.predict[i,]
  Norm<-sum(pX.filter[i,])
  pX.filter[i,]<-pX.filter[i,]/Norm
  LogLike<-LogLike+log(Norm)
  Xpred[i]<-which.max(pX.filter[i,])
}
pX.smooth[n,]<-pX.filter[n,]
Xsmooth[n]<-which.max(pX.smooth[n,])
for(i in (n-1):1){
  pX.smooth[i,]<-pX.filter[i,] * (P %*% (pX.smooth[i+1,]/pX.predict[i+1,]))
  Xsmooth[i]<-which.max(pX.smooth[i,])
}

#P[X=1]
pmat<-cbind(Y[1:10],pX.filter[1:10,1],pX.smooth[1:10,1])
rownames(pmat)<-paste('t=',1:10)
colnames(pmat)<-c('Y','Filter','Smooth')
pmat

##           Y           Filter           Smooth
## t= 1    0.6061964 0.44003334328 0.39593926467
## t= 2    1.0420040 0.73754153563 0.76626364554
## t= 3   -2.3894067 0.00008326208 0.00009709526
## t= 4   -1.5069205 0.00161587544 0.00122472854
## t= 5    1.9107422 0.97867017268 0.98165627194
## t= 6   -1.6465883 0.00069082971 0.00051907782
## t= 7    2.5180991 0.99648972462 0.99698942321
## t= 8   -2.1864963 0.00013567457 0.00015604647
## t= 9   -0.4792575 0.03411544813 0.03957532770
## t= 10  -2.4447085 0.00009573548 0.00007270211

table(X,Xpred)

##      Xpred
## X      1      2
## 1 164   27
## 2   17  292

table(X,Xsmooth)

##      Xsmooth
## X      1      2
## 1 165   26
## 2   16  293

```

The HMM structure is commonly implemented when the observation is also *discrete*. Then the model is specified by

- *transition probabilities*

$$p_{kl} = \Pr[X_t = l | X_{t-1} = k]$$

with initial distribution  $\Pr[X_1 = k], k = 1, \dots, S$ .

- *emission probabilities*

$$e_k(m) = \Pr[Y_t = m | X_t = k]$$

$e_k(m)$  can be specified for a finite set ( $m \in \{1, 2, \dots, M\}$ ), or on the integers.

**The Viterbi Algorithm:** We seek the *most probable* state path, that is, the most probable sequence of states in light of the data:

$$\begin{aligned} x_{1:n}^* &= \arg \max_{x_{1:n}} p(x_{1:n}, y_{1:n}) \\ &= \arg \max_{x_{1:n}} p(x_{1:n} | y_{1:n}) \end{aligned}$$

- $p(x_{1:n}, y_{1:n})$  is the joint probability or joint likelihood
- $p(x_{1:n} | y_{1:n})$  is the posterior probability

The calculation is carried out using the *Viterbi* algorithm. The recursion proceeds as follows

(i) Initialize vector  $v_k(0) = \Pr[X_1 = k], k = 1, \dots, S$ .

(ii) Recursion: for  $t = 1, \dots, n$ , compute

$$\begin{aligned} v_l(t) &= e_l(y_t) \max_k (v_k(t-1) p_{kl}) \\ point_t(l) &= \arg \max_k (v_k(t-1) p_{kl}) \end{aligned}$$

for each  $l = 1, 2, \dots, S$

(iii) Termination

$$\begin{aligned} p(y_{1:n}, x_{1:n}^*) &= \max_k (v_k(n)) \\ x_n^* &= \arg \max_k (v_k(n)) \end{aligned}$$

(iv) Trace back the optimal path

$$x_{t-1}^* = point_t(x_t^*) \quad t = n, n-1, \dots, 2.$$

```
V<-Point<-matrix(0,ncol=n,nrow=S)
v0<-rep(1,S)/S
for(l in 1:S){
  V[l,1]<-dnorm(Y[1],mu[1],1)*max(P[l,]*v0)
  Point[l,1]<-which.max(P[l,]*v0)
}
Const<-sum(V[,1])
V[,1]<-V[,1]/Const
Const.run<-log(Const)
```

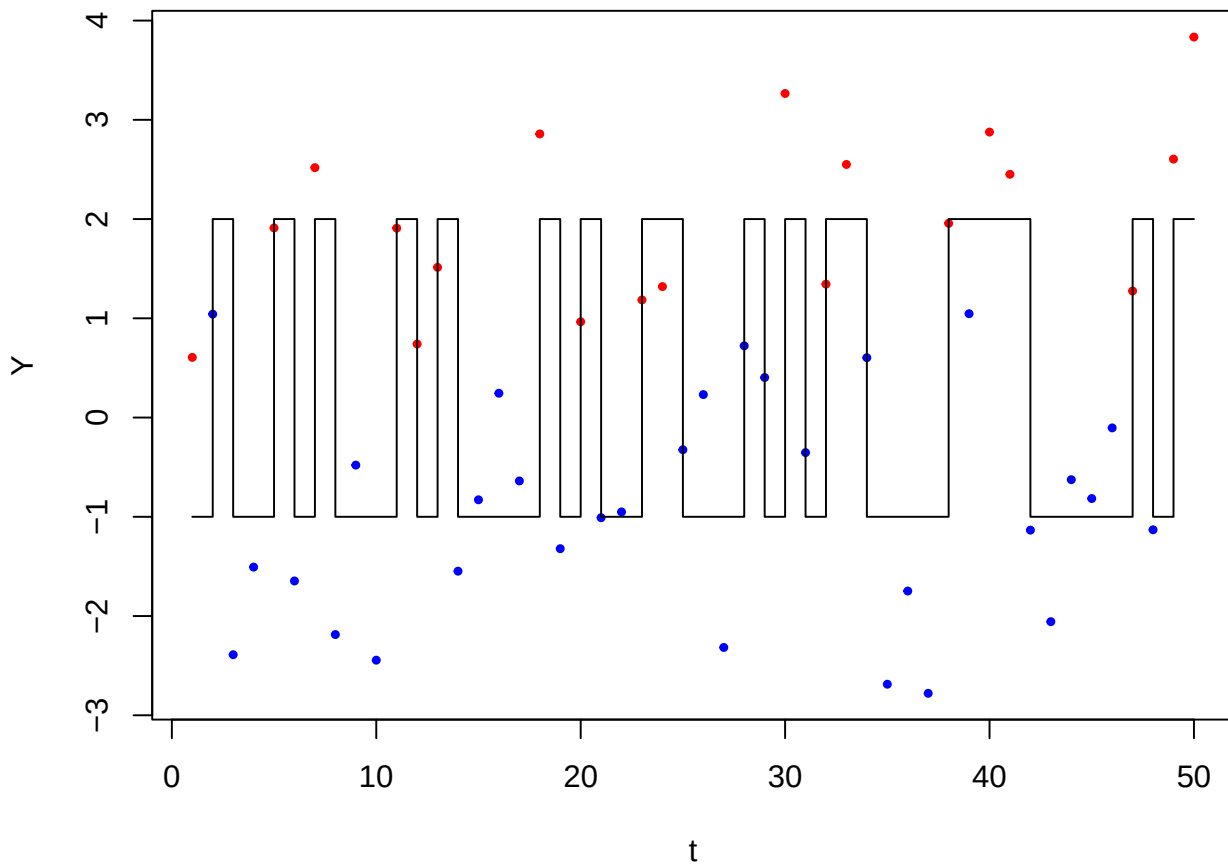
```

for(ti in 2:n){
  vp<-V[, (ti-1)]
  for(l in 1:S){
    V[l,ti]<-dnorm(Y[ti],mu[l],1)*max(P[l,] * vp)
    Point[l,ti]<-which.max(P[l,] * vp)
  }
  Const<-sum(V[,ti])
  V[,ti]<-V[,ti]/Const

  Const.run<-Const.run+log(Const)
}

Xstar<-rep(0,n)
Xstar[n]<-which.max(V[,n])
for(ti in n:2){
  Xstar[ti-1]<-Point[Xstar[ti],ti]
}
par(mar=c(4,4,2,2))
plot(Y[1:50],pch=19,cex=0.5,xlab="t",ylab="Y",col=cvec[X[1:50]])
lines(c(1:50),mu[Xstar[1:50]],type="s")

```



**The Forward Algorithm:** The probability of an observed sequence that yields the likelihood is given by

$$p(y_{1:n}) = \sum_{x_{1:n}} p(y_{1:n}, x_{1:n})$$

However, the direct computation of this sum is not in general possible, so a recursive calculation is needed. Note first that for  $1 \leq t \leq n$

$$p(y_{1:t}) = \sum_{x_t=1}^S p(x_t, y_{1:t})$$

and we may focus on computing the terms in the sum

$$f_{x_t}(t) = p(x_t, y_{1:t})$$

for  $x_t = 1, 2, \dots, S$ . Secondly, notice that

$$\begin{aligned} p(x_t, y_{1:t}) &= \sum_{x_{t-1}=1}^S p(x_{t-1}, x_t, y_{1:t}) && \text{total probability} \\ &= \sum_{x_{t-1}=1}^S p(y_t | x_t, x_{t-1}, y_{1:(t-1)}) p(x_t | x_{t-1}, y_{1:(t-1)}) p(x_{t-1}, y_{1:(t-1)}) && \text{chain rule} \\ &= \sum_{x_{t-1}=1}^S p(y_t | x_t) p(x_t | x_{t-1}) p(x_{t-1}, y_{1:(t-1)}) && \text{conditional independence} \\ &= p(y_t | x_t) \sum_{x_{t-1}=1}^S p(x_t | x_{t-1}) p(x_{t-1}, y_{1:(t-1)}) \end{aligned}$$

In this expression,  $p(y_t | x_t)$  is the observation/emission probability,  $p(x_t | x_{t-1})$  is the transition probability, and  $p(x_{t-1}, y_{1:(t-1)})$  is the joint probability from the previous recursion step which we denoted  $f_{x_{t-1}}(t-1)$ . Therefore, we simply need to track the values of  $f_x(t)$  for all  $x$  and  $t$ , with updates using the recursive formula. To set up the recursion, we need

$$f_{x_1}(1) = p(x_1, y_1) = p(y_1 | x_1) p(x_1)$$

which we can compute from the observation/emission probability and the probability distribution for  $X_1$ , which we have assumed to be  $\pi$ .

(i) Initialize vector  $f_k(1) = e_k(y_1) \pi_k$  for  $k = 1, 2, \dots, S$

(ii) Recursion: for  $t = 2, \dots, n$ , and  $l = 1, 2, \dots, S$ ,

$$f_l(t) = e_l(y_t) \sum_{k=1}^S f_k(t-1) p_{kl}$$

(iii) Final calculation:

$$p(y_{1:n}) = \sum_{k=1}^S f_k(n).$$

Note: in this calculation, when  $n$  is large,  $p(y_{1:n})$  is likely to be very small, so it is preferable to work on the log scale – however, we must compute the forward quantities on the original scale. We may scale by (for example)

$$m_t = \max_k f_k(t)$$

at each step to prevent numerical underflow.

```
fmat<-matrix(0,nrow=n,ncol=S)
logm<-rep(0,n)
fmat[1,<-dnorm(Y[1],mu,sig)*Pi
logm[1]<-log(max(fmat[1,]))
fmat[1,<-fmat[1,]/max(fmat[1,])
for(i in 2:n){
  fmat[i,<-dnorm(Y[i],mu,sig)*(P %*% fmat[i-1,])
  logm[i]<-log(max(fmat[i,]))
  fmat[i,<-fmat[i,]/max(fmat[i,])
}
log(sum(fmat[n,]))+sum(logm)

## [1] -963.4339

LogLike

## [1] -963.2597
```

We also seek the posterior (smoothing) probabilities

$$\begin{aligned} p(X_t = k | y_{1:n}) &= \frac{p(X_t = k, y_{1:n})}{p(y_{1:n})} \\ &= \frac{p(y_1, \dots, y_t, X_t = k) p(y_{t+1}, \dots, y_n | X_t = k)}{p(y_{1:n})} \end{aligned}$$

for  $k = 1, 2, \dots, S$ . Direct computation is not possible, so a recursive calculation is needed, after completing the Forward Algorithm.

**The Backward Algorithm:** The backward algorithm works in reverse order through the data:

- (i) Initialize element  $b_k(n) = f_k(n)$  for  $k = 1, \dots, S$ .
- (ii) Backwards recursion: for  $t = n - 1, \dots, 1$  compute

$$b_k(t) = \sum_{l=1}^S p_{kl} e_l(y_{t+1}) b_l(t+1)$$

for  $k = 1, 2, \dots, S$ .

```
bmat<-matrix(0,nrow=n,ncol=S)
logbm<-rep(0,n)
bmat[n,<-fmat[n,]
logbm[n]<-log(max(bmat[n,]))
bmat[n,<-bmat[n,]/max(bmat[n,])
for(i in (n-1):1){
  bmat[i,<-dnorm(Y[i],mu,sig)*(t(P) %*% bmat[i+1,])
  logbm[i]<-log(max(bmat[i,]))
  bmat[i,<-bmat[i,]/max(bmat[i,])
}
```

The forward and backward probabilities,  $f_k(t)$  and  $b_k(t)$ , are in fact the probabilities

$$\begin{aligned} f_k(t) &= p(y_1, \dots, y_t, X_t = k) \\ b_k(t) &= p(y_{t+1}, \dots, y_n | X_t = k) \end{aligned}$$

Thus we have

$$p(X_t = k, y_{1:n}) = f_k(t)b_k(t)$$

and hence, for  $k = 1, \dots, S$ ,

$$p(X_t = k | y_{1:n}) = \frac{f_k(t)b_k(t)}{p(y_{1:n})}$$

**Estimation:** The algorithms above are all expressed in terms of known transition and emission probabilities. These probabilities can be pre-specified, estimated from observed sequences, or estimated during the state sequence reconstruction.

If the state sequence is *known*, let  $n_{kl}$  be the number of transitions from state  $k$  to state  $l$ , and let  $r_k(b)$  be the number of times character  $b$  is emitted from state  $k$ . Then we may estimate the transition and emission probabilities as follows

$$\hat{p}_{kl} = \frac{n_{kl}}{\sum_{l'} n_{kl'}} \quad \hat{e}_k(b) = \frac{r_k(b)}{\sum_{b'} r_k(b')} \quad (1)$$

If the state sequence is *not known*, we must reconstruct the state sequence and perform estimation concurrently. This is achieved using the *Baum-Welch* algorithm, which is a special case of the *Expectation-Maximization* (EM) algorithm.

**The Baum-Welch Algorithm:** The Baum-Welch algorithm estimates  $n_{kl}$  and  $r_k(b)$  as the *expected* number of transitions and emissions, which are computed using the forward and backward algorithms. The probability of a transition  $k \rightarrow l$  at time  $t$  is

$$p(X_t = k, X_{t+1} = l | y_{1:n}) = \frac{f_k(t) p_{kl} e_l(y_{t+1}) b_l(t+1)}{p(y_{1:n})}$$

Suppose that  $N$  replicate observed sequences

$$y_{1:n}^1, \dots, y_{1:n}^N$$

are available. Then the expected number of times this transition will be observed (in light of the data) is

$$n_{kl} = \sum_{i=1}^N \left\{ \frac{\sum_t f_k^i(t) a_{kl} e_l(y_{t+1}^i) b_l^i(t+1)}{p(y_{1:n}^i)} \right\}$$

In this formula

- $y_{1:n}^i$  is the  $i^{th}$  observed sequence
- $f_k^i(t)$  is the forward probability from the forward algorithm implemented for sequence  $i$
- $b_l^i(t)$  is the backward probability from the backward algorithm implemented for sequence  $i$

Similarly,

$$r_k(b) = \sum_{i=1}^N \left\{ \frac{\sum_{t \in \mathcal{B}_i} f_k^i(t) b_l^i(t)}{p(y_{1:n}^i)} \right\}$$

where  $\mathcal{B}_i$  is the set of indices of locations in replicate sequence  $i$  where character  $b$  is emitted.

- (i) Initialize parameter values to some arbitrary values
- (ii) Recursion: Set  $n_{kl}^{(0)}, r_k^{(0)}(b)$  to be some small count value, then for  $i = 1, \dots, N$ : calculate  $f_k^i(t)$  using the Forward Algorithm, and  $b_k^i(t)$  using the Backward Algorithm, then set

$$n_{kl}^{(i)} = n_{kl}^{(i-1)} + \frac{1}{p(y_{1:n}^i)} \left\{ \sum_{t=1}^n f_k^i(t) p_{kl} e_l(y_{t+1}^i) b_l^i(t+1) \right\}$$

$$r_k^{(i)}(b) = r_k^{(i-1)}(b) + \frac{1}{p(y_{1:n}^i)} \left\{ \sum_{t \in \mathcal{B}_i} f_k^i(t) b_k^i(t) \right\}$$

- (iii) Estimate  $p_{kl}$  and  $e_k(b)$  using (1), and compute the likelihood

$$\prod_{i=1}^N p(y_{1:n}^i)$$

- (iv) Terminate the recursion if the computed likelihood value is virtually unchanged compared to the previous iteration.

Note that most of the above calculations carry through to the continuous emission case, that is if the emission probabilities

$$e_k(m) = \Pr[Y_t = m | X_t = k]$$

are replaced by densities

$$e_k(y) = p(y | X_t = k)$$

The forward and backward quantities become unnormalized joint densities

To avoid underflow in the computation, the computations can either be computed on the log scale, or with continual rescaling of the calculation. For example, in the Viterbi algorithm, the computation of the  $v_k(t)$  quantities will underflow quickly without rescaling, but rescaling (for example) by

$$c_t = \sum_{k=1}^S v_k(t)$$

will prevent this. The constants  $c_t$  must be stored for future use; if the forward and backward probabilities are computed after rescaling, then this rescaling must be undone when, for example,  $p(y_{1:n})$  is computed.

**Direct estimation using filtering:** As for the Kalman Filter, we can use the filtering approach to compute the likelihood, and perform maximum likelihood estimation.

```
ml.HMM<-function(parv,Yv,Sv){
  P<-matrix(parv[1:(Sv*(Sv-1))],nrow=Sv,byrow=T)
  P<-cbind(P,0)
  P<-exp(P)
  P<-t(apply(P,1,function(x){return(x/sum(x))}))
  mu<-parv[(Sv*(Sv-1)+1):(Sv*Sv)]
  sig<-exp(parv[(Sv*Sv)+1])
  n<-length(Yv)
  Ptmp<-P;for(i in 1:100){Ptmp<-Ptmp%*%P}
  Pi<-Ptmp[1,]
  pXf<-pXp<-matrix(0,nrow=n,ncol=Sv)
  pXp[1,]<-Pi
```

```

pXf[1,]<-dnorm(Yv[1],mu,sig)*Pi
Norm<-sum(pXf[1,])
pXf[1,]<-pXf[1,]/Norm
LogLike<-log(Norm)
for(i in 2:n){
  pXp[i,]<- colSums(P*pXf[i-1,])
  pXf[i,]<-dnorm(Yv[i],mu,sig)*pXp[i,]
  Norm<-sum(pXf[i,])
  pXf[i,]<-pXf[i,]/Norm
  LogLike<-LogLike+log(Norm)
}
return(LogLike)
}

S^2+1

## [1] 5

xp<-c(0,0,2,0,0)
xfit<-optim(xp,ml.HMM,Yv=Y,Sv=S,control=list(fnscale=-1),method='BFGS')
par.max<-xfit$par

P.hat<-matrix(par.max[1:(S*(S-1))],nrow=S,byrow=T)
P.hat<-cbind(P.hat,0)
P.hat<-exp(P.hat)
P.hat<-t(apply(P.hat,1,function(x){return(x/sum(x))}))
mu.hat<-par.max[(S*(S-1)+1):(S*S)]
sig.hat<-exp(par.max[(S*S)+1])

P.hat

##           [,1]      [,2]
## [1,] 0.3275179 0.6724821
## [2,] 0.4163523 0.5836477

mu.hat

## [1] 1.964978 -1.017225

sig.hat

## [1] 1.018478

```