APPENDIX A

Examples of R code

In this Appendix we give examples of the **R** code used to perform the analyses presented in the book. We do not provide complete coverage of what we have done; only the case of a Poisson–HMM is covered fully, but we also illustrate how that code can be modified to cover two other models. Users are encouraged to experiment with the code and to write their own functions for additional models, starting with simple models, such as the binomial–HMM, and then progressing to more advanced models, e.g. multivariate models or HMMs with covariates. There is, however, other **R** software available which can implement many of our models, e.g. the packages `repeated` (Lindsey, 2008), `msm` (Jackson *et al.*, 2003), and `HiddenMarkov` (Harte, 2008).

We are aware that the code presented in this appendix can be improved. We have sometimes sacrificed efficiency or elegance in the interest of transparency. Our aim has been to give code that is easy to modify in order to deal with alternative models.

The time series is assumed to be stored as a vector `x` of length `n`, rather than the T which was used in the body of the text. This is to avoid overwriting the standard **R** abbreviation `T` for `TRUE`.

A.1 Fit a stationary Poisson–HMM by direct numerical maximization

Estimation by direct numerical maximization of the likelihood function is illustrated here by code for a stationary Poisson–HMM. The computation uses four functions, which appear in A.1.1– A.1.4:

- `pois.HMM.pn2pw` transforms the natural parameters (the vector of Poisson means $\boldsymbol{\lambda}$ and the t.p.m. $\boldsymbol{\Gamma}$) of an m-state Poisson–HMM into a vector of (unconstrained) working parameters;
- `pois.HMM.pw2pn` performs the inverse transformation;
- `pois.HMM.mllk` computes minus the log-likelihood of a Poisson–HMM for a given set of working parameters; and
- `pois.HMM.mle` estimates the parameters of the model using numerical minimization of minus the log-likelihood.

A.1.1 Transform natural parameters to working

```
pois.HMM.pn2pw <- function(m,lambda,gamma)
{
 tlambda <- log(lambda)
 tgamma  <- NULL
 if(m>1)
   {
   foo    <- log(gamma/diag(gamma))
   tgamma<- as.vector(foo[!diag(m)])
   }
 parvect <- c(tlambda,tgamma)
 parvect
}
```

The vector `parvect` which is returned contains the working parameters, starting with the entries $\log \lambda_i$ for $i = 1, 2, \ldots, \mathtt{m}$, and followed by τ_{ij} for $i, j = 1, 2, \ldots, \mathtt{m}$ and $i \neq j$. See Section 3.3.1 (pp. 47–49).

A.1.2 Transform working parameters to natural

```
pois.HMM.pw2pn <- function(m,parvect)
{
 epar   <- exp(parvect)
 lambda <- epar[1:m]
 gamma  <- diag(m)
 if(m>1)
   {
   gamma[!gamma] <- epar[(m+1):(m*m)]
   gamma         <- gamma/apply(gamma,1,sum)
   }
 delta  <- solve(t(diag(m)-gamma+1),rep(1,m))
 list(lambda=lambda,gamma=gamma,delta=delta)
}
```

The first `m` entries of `parvect`, the vector of working parameters, are used to compute λ_i, for $i = 1, 2, \ldots, \mathtt{m}$, and the remaining $\mathtt{m}(\mathtt{m}-1)$ entries to compute $\mathbf{\Gamma}$. The stationary distribution `delta` implied by the matrix `gamma` is also computed and returned; see Exercise 8(a) on p. 26.

A.1.3 Log-likelihood of a stationary Poisson–HMM

```
pois.HMM.mllk <- function(parvect,x,m,...)
{
 if(m==1) return(-sum(dpois(x,exp(parvect),log=TRUE)))
 n          <- length(x)
 pn         <- pois.HMM.pw2pn(m,parvect)
 allprobs   <- outer(x,pn$lambda,dpois)
 allprobs   <- ifelse(!is.na(allprobs),allprobs,1)
 lscale     <- 0
 foo        <- pn$delta
```

```
10  for (i in 1:n)
11    {
12    foo    <- foo%*%pn$gamma*allprobs[i,]
13    sumfoo <- sum(foo)
14    lscale <- lscale+log(sumfoo)
15    foo    <- foo/sumfoo
16    }
17  mllk       <- -lscale
18  mllk
19 }
```

This function computes minus the log-likelihood of an `m`-state model for a given vector `parvect` of working parameters and a given vector `x` of observations, some of which may be missing (`NA`). The natural parameters are extracted in line 5. Line 6 computes an $\mathtt{n} \times \mathtt{m}$ array of Poisson probabilities

$$e^{-\lambda_i}\lambda_i^{x_t}/x_t! \quad (i = 1, \ldots \mathtt{m}, \quad t = 1, \ldots, \mathtt{n}),$$

line 7 substitutes the value 1 for these probabilities in the case of missing data, and lines 8–17 implement the algorithm for the log-likelihood given on p. 47.

A.1.4 ML estimation of a stationary Poisson–HMM

```
1  pois.HMM.mle <- function(x,m,lambda0,gamma0,...)
2  {
3   parvect0  <- pois.HMM.pn2pw(m,lambda0,gamma0)
4   mod       <- nlm(pois.HMM.mllk,parvect0,x=x,m=m)
5   pn        <- pois.HMM.pw2pn(m,mod$estimate)
6   mllk      <- mod$minimum
7   np        <- length(parvect0)
8   AIC       <- 2*(mllk+np)
9   n         <- sum(!is.na(x))
10  BIC       <- 2*mllk+np*log(n)
11  list(lambda=pn$lambda,gamma=pn$gamma,delta=pn$delta,
12               code=pn$code,mllk=mllk,AIC=AIC,BIC=BIC)
13 }
```

This function accepts initial values `lambda0` and `gamma0` for the natural parameters $\boldsymbol{\lambda}$ and $\boldsymbol{\Gamma}$, converts these to the vector `parvect0` of working parameters, invokes the minimizer `nlm` to minimize –log-likelihood, and returns parameter estimates and values of selection criteria. The termination code (`code`), which indicates how `nlm` terminated, is also returned; see the **R** help for `nlm` for an explanation of the values of `code`, and for additional arguments that can be specified to tune the behaviour of `nlm`.

If a different minimizer, such as `optim`, is used instead of `nlm` it is necessary to modify the code listed above, to take account of differences in the calling sequence and the returned values.

A.2 More on Poisson–HMMs, including estimation by EM

A.2.1 Generate a realization of a Poisson–HMM

```
1  pois.HMM.generate_sample <-
2   function(n,m,lambda,gamma,delta=NULL)
3  {
4   if(is.null(delta))delta<-solve(t(diag(m)-gamma+1),rep(1,m))
5   mvect <- 1:m
6   state <- numeric(n)
7   state[1] <- sample(mvect,1,prob=delta)
8   for (i in 2:n)
9     state[i]<-sample(mvect,1,prob=gamma[state[i-1],])
10  x <- rpois(n,lambda=lambda[state])
11  x
12 }
```

This function generates a realization of length `n` of an `m`-state HMM with parameters `lambda` and `gamma`. If `delta` is not supplied, the stationary distribution is computed (line 4) and used as initial distribution. If `delta` is supplied, it is used as initial distribution.

A.2.2 Forward and backward probabilities

```
1  pois.HMM.lalphabeta<-function(x,m,lambda,gamma,delta=NULL)
2  {
3   if(is.null(delta))delta<-solve(t(diag(m)-gamma+1),rep(1,m))
4   n          <- length(x)
5   lalpha     <- lbeta<-matrix(NA,m,n)
6   allprobs   <- outer(x,lambda,dpois)
7   foo        <- delta*allprobs[1,]
8   sumfoo     <- sum(foo)
9   lscale     <- log(sumfoo)
10  foo        <- foo/sumfoo
11  lalpha[,1] <- log(foo)+lscale
12  for (i in 2:n)
13    {
14    foo        <- foo%*%gamma*allprobs[i,]
15    sumfoo     <- sum(foo)
16    lscale     <- lscale+log(sumfoo)
17    foo        <- foo/sumfoo
18    lalpha[,i] <- log(foo)+lscale
19    }
20  lbeta[,n]  <- rep(0,m)
21  foo        <- rep(1/m,m)
22  lscale     <- log(m)
23  for (i in (n-1):1)
24    {
25    foo        <- gamma%*%(allprobs[i+1,]*foo)
26    lbeta[,i]  <- log(foo)+lscale
27    sumfoo     <- sum(foo)
28    foo        <- foo/sumfoo
29    lscale     <- lscale+log(sumfoo)
30    }
31  list(la=lalpha,lb=lbeta)
32 }
```

This function computes the *logarithms* of the forward and backward probabilities as defined by Equations (4.1) and (4.2) on p. 60, in the form of `m` × `n` matrices. The initial distribution `delta` is handled as in A.2.1. To reduce the risk of underflow, scaling is applied, in the same way as in A.1.3.

A.2.3 EM estimation of a Poisson–HMM

```
pois.HMM.EM <- function(x,m,lambda,gamma,delta,
                        maxiter=1000,tol=1e-6,...)
{
 lambda.next    <- lambda
 gamma.next     <- gamma
 delta.next     <- delta
 for (iter in 1:maxiter)
   {
   lallprobs    <- outer(x,lambda,dpois,log=TRUE)
   fb  <-  pois.HMM.lalphabeta(x,m,lambda,gamma,delta=delta)
   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)/
                     sum(exp(la[j,]+lb[j,]-llk))
   }
   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(lambda-lambda.next)) +
                 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(lambda=lambda,gamma=gamma,delta=delta,
           mllk=-llk,AIC=AIC,BIC=BIC))
     }
   lambda     <- lambda.next
   gamma      <- gamma.next
   delta      <- delta.next
   }
 print(paste("No convergence after",maxiter,"iterations"))
 NA
}
```

This function implements the EM algorithm as described in Sections 4.2.2 and 4.2.3, with the initial values `lambda`, `gamma` and `delta` for the natural parameters $\boldsymbol{\lambda}$, $\boldsymbol{\Gamma}$ and $\boldsymbol{\delta}$. In each iteration the logs of the forward and backward probabilities are computed (lines 10–12); the forward and backward probabilities themselves would tend to underflow. The log-likelihood l, which is needed in both E and M steps, is computed as follows:

$$l = \log\left(\sum_{i=1}^{m} \alpha_n(i)\right) = c + \log\left(\sum_{i=1}^{m} \exp\Big(\log(\alpha_n(i)) - c\Big)\right),$$

where c is chosen in such a way as to reduce the chances of underflow in the exponentiation. The convergence criterion `crit` used above is the sum of absolute values of the changes in the parameters in one iteration, and could be replaced by some other criterion chosen by the user.

A.2.4 Viterbi algorithm

```
1  pois.HMM.viterbi<-function(x,m,lambda,gamma,delta=NULL,...)
2  {
3   if(is.null(delta))delta<-solve(t(diag(m)-gamma+1),rep(1,m))
4   n          <- length(x)
5   poisprobs  <- outer(x,lambda,dpois)
6   xi         <- matrix(0,n,m)
7   foo        <- delta*poisprobs[1,]
8   xi[1,]     <- foo/sum(foo)
9   for (i in 2:n)
10    {
11    foo     <- apply(xi[i-1,]*gamma,2,max)*poisprobs[i,]
12    xi[i,] <- foo/sum(foo)
13    }
14  iv<-numeric(n)
15  iv[n]      <-which.max(xi[n,])
16  for (i in (n-1):1)
17    iv[i] <- which.max(gamma[,iv[i+1]]*xi[i,])
18  iv
19 }
```

This function computes the most likely sequence of states, given the parameters and the observations, as described on p. 84: see Equations (5.9)–(5.11). The initial distribution `delta` is again handled as in A.2.1.

A.2.5 Conditional state probabilities

```
1  pois.HMM.state_probs <-
2   function(x,m,lambda,gamma,delta=NULL,...)
3  {
4   if(is.null(delta))delta<-solve(t(diag(m)-gamma+1),rep(1,m))
5   n           <- length(x)
```

```
 fb          <- pois.HMM.lalphabeta(x,m,lambda,gamma,
                    delta=delta)
 la          <- fb$la
 lb          <- fb$lb
 c           <- max(la[,n])
 llk         <- c+log(sum(exp(la[,n]-c)))
 stateprobs  <- matrix(NA,ncol=n,nrow=m)
 for (i in 1:n) stateprobs[,i]<-exp(la[,i]+lb[,i]-llk)
 stateprobs
}
```

This function computes the probability $\Pr(C_t = i \mid \mathbf{X}^{(\mathtt{n})})$ for $t = 1, \ldots \mathtt{n}$, $i = 1, \ldots, \mathtt{m}$, by means of Equation (5.6) on p. 81. As in A.2.3, the logs of the forward probabilities are shifted before exponentiation by a quantity `c` chosen to reduce the chances of underflow; see line 11.

A.2.6 Local decoding

```
pois.HMM.local_decoding <-
 function(x,m,lambda,gamma,delta=NULL,...)
{
 stateprobs <-
    pois.HMM.state_probs(x,m,lambda,gamma,delta=delta)
 ild <- rep(NA,n)
 for (i in 1:n) ild[i]<-which.max(stateprobs[,i])
 ild
}
```

This function performs local decoding, i.e. determines for each time t the most likely state, as specified by Equation (5.7) on p. 81.

A.2.7 State prediction

```
pois.HMM.state_prediction <-
 function(x,m,lambda,gamma,delta=NULL,H=1,...)
{
 if(is.null(delta))delta<-solve(t(diag(m)-gamma+1),rep(1,m))
 n           <- length(x)
 fb          <- pois.HMM.lalphabeta(x,m,
                    lambda,gamma,delta=delta)
 la          <- fb$la
 c           <- max(la[,n])
 llk         <- c+log(sum(exp(la[,n]-c)))
 statepreds  <- matrix(NA,ncol=H,nrow=m)
 foo1        <- exp(la[,n]-llk)
 foo2        <- diag(m)
 for (i in 1:H)
   {
   foo2            <- foo2%*%gamma
   statepreds[,i] <- foo1%*%foo2
   }
 statepreds
}
```

This function computes the probability $\Pr(C_t = i \mid \mathbf{X}^{(\mathtt{n})})$ for $t = \mathtt{n}+1, \ldots, \mathtt{n}+\mathtt{H}$ and $i = 1, \ldots, \mathtt{m}$, by means of Equation (5.12) on p. 86. As in A.2.3, the logs of the forward probabilities are shifted before exponentiation by a quantity `c` chosen to reduce the chances of underflow; see line 10.

A.2.8 Forecast distributions

```
1  pois.HMM.forecast <- function(x,m,lambda,gamma,
2         delta=NULL,xrange=NULL,H=1,...)
3  {
4   if(is.null(delta))
5         delta<-solve(t(diag(m)-gamma+1),rep(1,m))
6   if(is.null(xrange))
7         xrange<-qpois(0.001,min(lambda)):
8                 qpois(0.999,max(lambda))
9   n         <- length(x)
10  allprobs  <- outer(x,lambda,dpois)
11  allprobs  <- ifelse(!is.na(allprobs),allprobs,1)
12  foo       <- delta*allprobs[1,]
13  sumfoo    <- sum(foo)
14  lscale    <- log(sumfoo)
15  foo       <- foo/sumfoo
16  for (i in 2:n)
17    {
18    foo     <- foo%*%gamma*allprobs[i,]
19    sumfoo  <- sum(foo)
20    lscale  <- lscale+log(sumfoo)
21    foo     <- foo/sumfoo
22    }
23  xi        <- matrix(NA,nrow=m,ncol=H)
24  for (i in 1:H)
25    {
26    foo     <- foo%*%gamma
27    xi[,i]  <- foo
28    }
29  allprobs  <- outer(xrange,lambda,dpois)
30  fdists    <- allprobs%*%xi[,1:H]
31  list(xrange=xrange,fdists=fdists)
32 }
```

This function uses Equation (5.4) on p. 77 to compute the forecast distributions $\Pr(X_{\mathtt{n}+h} = x \mid \mathbf{X}^{(\mathtt{n})})$ for $h = 1, \ldots, \mathtt{H}$ and a range of x values. This range can be specified via `xrange`, but if not, a suitably wide range is used: see lines 7 and 8.

A.2.9 Conditional distribution of one observation given the rest

```
1  pois.HMM.conditionals <-
2   function(x,m,lambda,gamma,delta=NULL,xrange=NULL,...)
3  {
```

```
4  if(is.null(delta))
5    delta  <- solve(t(diag(m)-gamma+1),rep(1,m))
6  if(is.null(xrange))
7    xrange <-qpois(0.001,min(lambda)):
8              qpois(0.999,max(lambda))
9  n       <- length(x)
10 fb      <- pois.HMM.lalphabeta(x,m,lambda,gamma,delta=delta)
11 la      <- fb$la
12 lb      <- fb$lb
13 la      <- cbind(log(delta),la)
14 lafact  <- apply(la,2,max)
15 lbfact  <- apply(lb,2,max)
16 w       <- matrix(NA,ncol=n,nrow=m)
17 for (i in 1:n)
18   {
19   foo   <- (exp(la[,i]-lafact[i])%*%gamma)*
20               exp(lb[,i]-lbfact[i])
21   w[,i] <- foo/sum(foo)
22   }
23 allprobs <- outer(xrange,lambda,dpois)
24 cdists   <- allprobs%*%w
25 list(xrange=xrange,cdists=cdists)
26 }
```

This function computes $\Pr(X_t = x \mid \mathbf{X}^{(-t)})$ for a range of x values and $t = 1, \ldots,$ `n` via Equation (5.3) on p. 77. The range can be specified via `xrange`, but if not, a suitably wide range is used. Here also the logs of the forward and the backward probabilities are shifted before exponentiation by a quantity chosen to reduce the chances of underflow; see lines 19 and 20.

A.2.10 Ordinary pseudo-residuals

```
1  pois.HMM.pseudo_residuals <-
2   function(x,m,lambda,gamma, delta=NULL,...)
3  {
4   if(is.null(delta))delta<-solve(t(diag(m)-gamma+1),rep(1,m))
5   n         <- length(x)
6   cdists    <- pois.HMM.conditionals(x,m,lambda, gamma,
7                      delta=delta,xrange=0:max(x))$cdists
8   cumdists  <- rbind(rep(0,n),apply(cdists,2,cumsum))
9   ul <- uh  <- rep(NA,n)
10  for (i in 1:n)
11    {
12    ul[i]   <- cumdists[x[i]+1,i]
13    uh[i]   <- cumdists[x[i]+2,i]
14    }
15  um        <- 0.5*(ul+uh)
16  npsr      <- qnorm(rbind(ul,um,uh))
17  npsr
18 }
```

This function computes, for each t from 1 to `n`, the ordinary normal pseudo-residuals as described in Section 6.2.2. These are returned in an $\texttt{n} \times 3$ matrix in which the columns give (in order) the lower, mid- and upper pseudo-residuals.

A.3 HMM with bivariate normal state-dependent distributions

This section presents the code for fitting an `m`-state HMM with bivariate normal state-dependent distributions, by means of the discrete likelihood. It is assumed that the observations are available as intervals: `x1` and `x2` are $\texttt{n} \times 2$ matrices of lower and upper bounds for the observations. (The values `-inf` and `inf` are permitted. Note that missing values are thereby allowed for.) The natural parameters are the means `mu` (an $\texttt{m} \times 2$ matrix), the standard deviations `sigma` (an $\texttt{m} \times 2$ matrix), the vector `corr` of `m` correlations, and the t.p.m. `gamma`.

There are in all $\texttt{m}^2 + 4\texttt{m}$ parameters to be estimated, and each evaluation of the likelihood requires `mn` bivariate-normal probabilities of a rectangle: see line 19 in A.3.3. This code must therefore be expected to be slow.

An example of the use of this code appears in Section 10.4.

A.3.1 Transform natural parameters to working

```
1  bivnorm.HMM.pn2pw <-
2   function(mu,sigma,corr,gamma,m)
3  {
4   tsigma <- log(sigma)
5   tcorr<-log((1+corr)/(1-corr))
6   tgamma <- NULL
7   if(m>1)
8     {
9     foo    <- log(gamma/diag(gamma))
10    tgamma <- as.vector(foo[!diag(m)])
11    }
12  parvect <- c(as.vector(mu),as.vector(tsigma),
13               tcorr,tgamma)
14  parvect
15 }
```

The working parameters are assembled in the vector `parvect` in the order: `mu`, `sigma`, `corr`, `gamma`. The means are untransformed, the s.d.s are log-transformed, the correlations are transformed from $(-1, 1)$ to $\mathbb{R}$ by $\rho \mapsto \log((1+\rho)/(1-\rho))$, and the t.p.m. transformed in the usual way, i.e. as in Section 3.3.1 (pp. 47–49).

A.3.2 Transform working parameters to natural

```
bivnorm.HMM.pw2pn <- function(parvect,m)
{
 mu    <- matrix(parvect[1:(2*m)],m,2)
 sigma <- matrix(exp(parvect[(2*m+1):(4*m)]),m,2)
 temp<-exp(parvect[(4*m+1):(5*m)])
 corr<-(temp-1)/(temp+1)
 gamma <- diag(m)
 if(m>1)
   {
   gamma[!gamma]<-exp(parvect[(5*m+1):(m*(m+4))])
   gamma<-gamma/apply(gamma,1,sum)
   }
 delta<-solve(t(diag(m)-gamma+1),rep(1,m))
 list(mu=mu,sigma=sigma,corr=corr,gamma=gamma,
      delta=delta)
}
```

A.3.3 Discrete log-likelihood

```
bivnorm.HMM.mllk<-function(parvect,x1,x2,m,...)
{
 n        <- dim(x1)[1]
 p        <- bivnorm.HMM.pw2pn(parvect,m)
 foo      <- p$delta
 covs     <- array(NA,c(2,2,m))
 for (j in 1:m)
   {
   covs[,,j] <- diag(p$sigma[j,])%*%
                matrix(c(1, p$corr[j],p$corr[j],1),2,2)%*%
                diag(p$sigma[j,])
   }
 P        <- rep(NA,m)
 lscale   <- 0
 for (i in 1:n)
   {
   for (j in 1:m)
     {
     P[j] <- pmvnorm(lower=c(x1[i,1],x2[i,1]),
                     upper=c(x1[i,2],x2[i,2]),
                     mean=p$mu[j,], sigma=covs[,,j])
     }
   foo    <- foo%*%p$gamma*P
   sumfoo <- sum(foo)
   lscale <- lscale+log(sumfoo)
   foo    <- foo/sumfoo
   }
 mllk     <- -lscale
 mllk
}
```

Lines 7–12 assemble the covariance matrices in the $2 \times 2 \times$ `m` array `covs`. Lines 14–28 compute minus the log-likelihood. The package `mvtnorm` is needed by this function, for the function `pmvnorm`.

A.3.4 MLEs of the parameters

```
bivnorm.HMM.mle<-
 function(x1,x2,m,mu0,sigma0,corr0,gamma0,...)
{
 n      <- dim(x1)[1]
 start  <- bivnorm.HMM.pn2pw(mu0,sigma0,corr0,gamma0,m)
 mod    <- nlm(bivnorm.HMM.mllk,p=start,x1=x1,x2=x2,m=m,
           steptol = 1e-4,iterlim = 10000)
 mllk   <- mod$minimum
 code   <- mod$code
 p      <- bivnorm.HMM.pw2pn(mod$estimate,m)
 np     <- m*(m+4)
 AIC    <- 2*(mllk+np)
 BIC    <- 2*mllk+np*log(n)
 list(mu=p$mu,sigma=p$sigma,corr=p$corr,
             gamma=p$gamma,delta=p$delta,code=code,
             mllk=mllk,AIC=AIC,BIC=BIC)
}
```

A.4 Fitting a categorical HMM by constrained optimization

As a final illustration of the variations of models and estimation techniques that are possible, we describe here how the constrained optimizer `constrOptim` can be used to fit a categorical HMM, i.e. a model of the kind that is discussed in Section 8.4.2. We assume that the underlying stationary Markov chain has m states, and that associated with each state i there are q probabilities adding to 1: $\sum_{j=1}^{q} \pi_{ij} = 1$. There are m^2 transition probabilities to be estimated, and mq state-dependent probabilities π_{ij}, a total of $m(m+q)$ parameters. The constraints (apart from nonnegativity of all the parameters) are in two groups:

$$\sum_{j=1}^{m} \gamma_{ij} = 1 \quad \text{and} \quad \sum_{j=1}^{q} \pi_{ij} = 1, \quad \text{for } i = 1, 2, \ldots m.$$

However, it is necessary to restructure these $2m$ constraints slightly in order to use `constrOptim`. We rewrite them as

$$\sum_{j=1}^{m-1} (-\gamma_{ij}) \geq -1 \quad \text{and} \quad \sum_{j=1}^{q-1} (-\pi_{ij}) \geq -1, \quad \text{for } i = 1, 2, \ldots m.$$

In this formulation there are $(m^2 - m) + (mq - m)$ parameters, all subject to nonnegativity constraints, and subject to a further $2m$ constraints; in all, $m(m+q)$ constraints must be supplied to `constrOptim`.

A model which we have checked by this means is the two-state model for categorized wind direction, described in Section 12.2.1. Our experience in this and other applications leads us to conclude provisionally

that `constrOptim` can be very slow compared to transformation and unconstrained maximization by `nlm`. The **R** help for `constrOptim` states that it is likely to be slow if the optimum is on a boundary, and indeed the optimum is on a boundary in the wind-direction application and many of our other models. The resulting parameter estimates differ very little from those supplied by `nlm`.

Note that, if $q = 2$, the code presented here can be used to fit models to binary time series, although more efficient functions can be written for that specific case.

A.4.1 Log-likelihood

```
1  cat.HMM.nllk <- function(parvect,x,m,q,...)
2  {
3   n      <- length(x)
4   gamma < -matrix(0,m,m)
5   pr     <- matrix(0,m,q)
6   for (i in 1:m)
7     {
8     gamma[i,1:(m-1)] <-
9       parvect[((i-1)*(m-1)+1):(i*(m-1))]
10    gamma[i,m]<-1-sum(gamma[i,1:(m-1)])
11    pr[i,1:(q-1)] <-
12      parvect[((i-1)*(q-1)+m*(m-1)+1):
13                      (i*(q-1)+m*(m-1))]
14    pr[i,q] <- 1-sum(pr[i,1:(q-1)])
15    }
16  delta  <- solve(t(diag(m)-gamma+1),rep(1,m))
17  lscale <- 0
18  foo    <- delta
19  for (i in 1:n)
20    {
21    foo    <- foo%*%gamma*pr[,x[i]]
22    sumfoo <- sum(foo)
23    lscale <- lscale+log(sumfoo)
24    foo    <- foo/sumfoo
25    }
26  nllk <- -lscale
27  nllk
28  }
```

In lines 4–15 the vector of working parameters `parvect` is unpacked into two matrices of natural parameters, `gamma` and `pr`. The stationary distribution `delta` is computed in line 16, and then minus log-likelihood `nllk` is computed in lines 17–26. The usual scaling method is applied in order to reduce the risk of numerical underflow.

A.4.2 MLEs of the parameters

```
1  cat.HMM.mle <- function(x,m,gamma0,pr0,...)
2  {
3   q         <- ncol(pr0)
4   parvect0  <- c(as.vector(t(gamma0[,-m])),
5                  as.vector(t(pr0[,-q])))
6   np        <- m*(m+q-2)
7   u1        <- diag(np)
8   u2 <- u3 <- matrix(0,m,np)
9   for (i in 1:m)
10    {
11    u2[i,((i-1)*(m-1)+1):(i*(m-1))] <- rep(-1,m-1)
12    u3[i,(m*(m-1)+(i-1)*(q-1)+1):(m*(m-1)+i*(q-1))]
13           <- rep(-1,q-1)
14    }
15  ui   <- rbind(u1,u2,u3)
16  ci   <- c(rep(0,np),rep(-1,2*m))
17  mod <- constrOptim(parvect0,cat.HMM.nllk,grad=NULL,ui=ui,
18             ci=ci,mu=1e-07,method="Nelder-Mead",x=x,m=m,q=q)
19  mod
20 }
```

In lines 6–16 we set up the constraints needed as input to `constrOptim`, which then returns details of the fitted model. The object `mod` contains (among other things) the vector of working parameters `mod$par`: see the **R** help for `constrOptim`.