Journal of Statistical Software

March 2011, Volume 39, Issue 4. *http://www.jstatsoft.org/*

Hidden Semi Markov Models for Multiple Observation Sequences: The mhsmm Package for R

Jared O'Connell
University of Oxford

Søren Højsgaard
Aarhus University

Abstract

This paper describes the R package **mhsmm** which implements estimation and prediction methods for hidden Markov and semi-Markov models for multiple observation sequences. Such techniques are of interest when observed data is thought to be dependent on some unobserved (or hidden) state. Hidden Markov models only allow a geometrically distributed sojourn time in a given state, while hidden semi-Markov models extend this by allowing an arbitrary sojourn distribution. We demonstrate the software with simulation examples and an application involving the modelling of the ovarian cycle of dairy cows.

Keywords: duration density, EM algorithm, hidden Markov model, R, sojourn time, Viterbi algorithm.

1. Introduction

The package **mhsmm** in the R system for statistical computing (R Development Core Team 2010) performs inference in multiple hidden Markov models and hidden semi-Markov models. A good overview of these models is given by Rabiner (1989). Efficient algorithms for parameter estimation are described by Guédon (2003). The models (and the **mhsmm** package) have been applied to oestrus detection in dairy cows (O'Connell, Tøgersen, Friggens, Løvendahl, and Højsgaard 2011).

The main features of the **mhsmm** package are as follows: Observations are allowed to be multivariate. Missing values are allowed. Observations must be recorded at equidistant times. The package is designed to allow the specification of custom emission distributions. It is possible to have multiple sequences of data. Parameter estimation is made using EM algorithms. Crucial parts of the code is written in C which makes estimation fast. The package is available from the Comprehensive R Archive Network at `http://CRAN.R-project.org/package=mhsmm`.

To our knowledge, there are two other software packages available for hidden semi-Markov

models: The first is the **AMAPmod** software (Godin and Guédon 2007), which is specifically for the exploration of plant architecture. Another R package for for hidden semi-Markov models is **hsmm** package, Bulla, Bulla, and Nenadic (2010). The **mhsmm** package is distinguished from **hsmm** in mainly two aspects: (1) **mhsmm** has the ability to estimate parameters for multiple observation sequences. (2) **mhsmm** is extensible because the user can create custom emission distributions.

The paper is organized as follows: Section 2 presents an example of a hidden Markov model based on simulated data. Section 3 goes more into the theory of the models and Section 4 contains various simulation examples each illustrating different aspects of the package. In Section 5 a real application on modelling the reproductive status of dairy cows is presented. Section 6 illustrates how to make user defined extensions. Finally Section 7 contains a discussion.

2. An introductory example

This example is based on simulation and illustrates hidden Markov models. A hidden Markov model can be described as follows (see Section 3 for more details): We consider a process evolving over discrete time points. Let $S = (S_t, t = 0, \ldots, T)$ denote a sequence of unobserved random variables, each with a finite state space $\{1, \ldots, J\}$, and let $X = (X_t, t = 1, \ldots, T)$ denote a corresponding set of observed random vectors. A hidden Markov model has the functional form

$$P(S, X) = P(S_0) \prod_{t=1}^{T} P(S_t|S_{t-1}) \prod_{t=1}^{T} P(X_t|S_t). \tag{1}$$

From (1) it follows that (1) the observables X are all conditionally independent given the latent variables S and (2) X_t depends on the latent variables S only through S_t in a hidden Markov model,

The term $P(S_0)$ is called the *initial distribution*, $P(S_t|S_{t-1})$ is the *transition distribution* and $P(X_t|S_t)$ is the *emission distribution*. In practice $P(S_0)$ is given as a vector π, $P(S_t|S_{t-1})$ is a transition matrix P (so the model is homogeneous because it is the same transition matrix for all t) while the emission distribution $P(X_t|S_t)$ (generically denoted by b) can be given in various different forms; see the examples below. Hence, a triple $\theta = (\pi, P, b)$ specifies a hidden Markov model.

In **mhsmm**, a HMM can be specified as:

```
R> J <- 3
R> initial <- rep(1/J, J)
R> P <- matrix(c(0.8, 0.5, 0.1, 0.05, 0.2, 0.5, 0.15, 0.3, 0.4), nrow = J)
R> b <- list(mu = c(-3, 0, 2), sigma = c(2, 1, 0.5))
R> model <- hmmspec(init = initial, trans = P, parms.emis = b,
+    dens.emis = dnorm.hsmm)
R> model

Hidden Markov Model specification:
J (number of states):
3
```

```
init:
[1] 0.3333333 0.3333333 0.3333333
transition:
     [,1] [,2] [,3]
[1,]  0.8 0.05 0.15
[2,]  0.5 0.20 0.30
[3,]  0.1 0.50 0.40
emission:
$mu
[1] -3  0  2

$sigma
[1] 2.0 1.0 0.5
```

The function `dnorm.hsmm` provides the density function for the emission distribution. The argument `rnorm.hsmm` is essentially a wrapper for the `rnorm()` function and takes the necessary specifcations from the model object. The specification of the emission distribution states that $X_t|S_t = s \sim N(\mu_s, \sigma_s^2)$. Notice that the elements of `sigma` are variances, not standard deviations. Section 6.2 shows an example of specifying a multivariate normal emission distribution.

We simulate data and plot the simulated data as follows (see Figure 1):

```
R> train <- simulate(model, nsim = 300, seed = 1234, rand.emis = rnorm.hsmm)
R> str(train)

List of 3
 $ s: int [1:300] 1 1 1 1 3 3 1 1 1 1 ...
 $ x: num [1:300] -3.533 -2.862 -0.682 -4.238 2.086 ...
 $ N: num 300
 - attr(*, "class")= chr "hsmm.data"

> plot(train, xlim = c(0, 100))
```

The parameters $\theta = (\pi, P, b)$ of the model are estimated using an EM algorithm (details and further references are given in Section 3) as follows. First we specify a starting value for the EM algorithm. This can be done with the `hmmspec()` function:

```
R> init0 <- rep(1/J, J)
R> P0 <- matrix(1/J, nrow = J, ncol = J)
R> b0 <- list(mu = c(-3, 1, 3), sigma = c(1, 1, 1))
R> startval <- hmmspec(init = init0, trans = P0, parms.emis = b0,
+    dens.emis = dnorm.hsmm)
```

The function `dnorm.hsmm` provides the density function for the emission distribution. The `hmmfit()` function implements the EM algorithm:

```
R> h1 = hmmfit(train, startval, mstep = mstep.norm)
R> plot(h1$loglik, type = "b", ylab = "Log-likelihood", xlab = "Iteration")
```

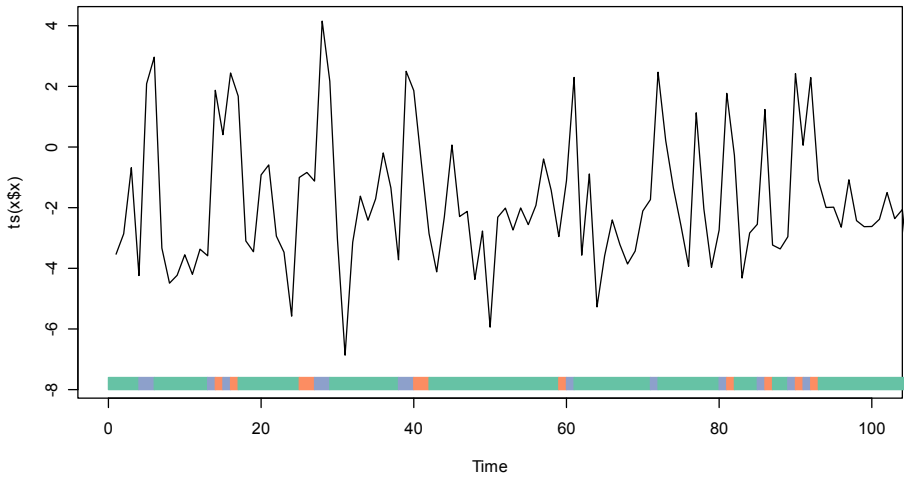

Figure 1: Simulated data from a hidden Markov model. The horizontal bar shows the different states while the curve shows the simulated values from the emission distribution. The colours correspond to states as follows: S_1=green, S_2=blue and S_3=orange.

Notice that the function `mstep.norm` provides the re-estimating formula for emission distribution (in this case, univariate Gaussian; see Section 3). We show how users can implement their own emission distributions in Section 6.

The estimated parameters are:

```
R> summary(h1)

init:
 1 0 0

transition:
      [,1]  [,2]  [,3]
[1,] 0.867 0.000 0.133
[2,] 0.477 0.210 0.313
[3,] 0.129 0.863 0.008

emission:
$mu
[1] -2.7728921  0.8137147  2.2085682

$sigma
[1] 2.253767 1.183886 0.499294
```

As a validation step, we simulate a test set of data from the original model and then try to reconstruct the state sequence using the `predict()` function:

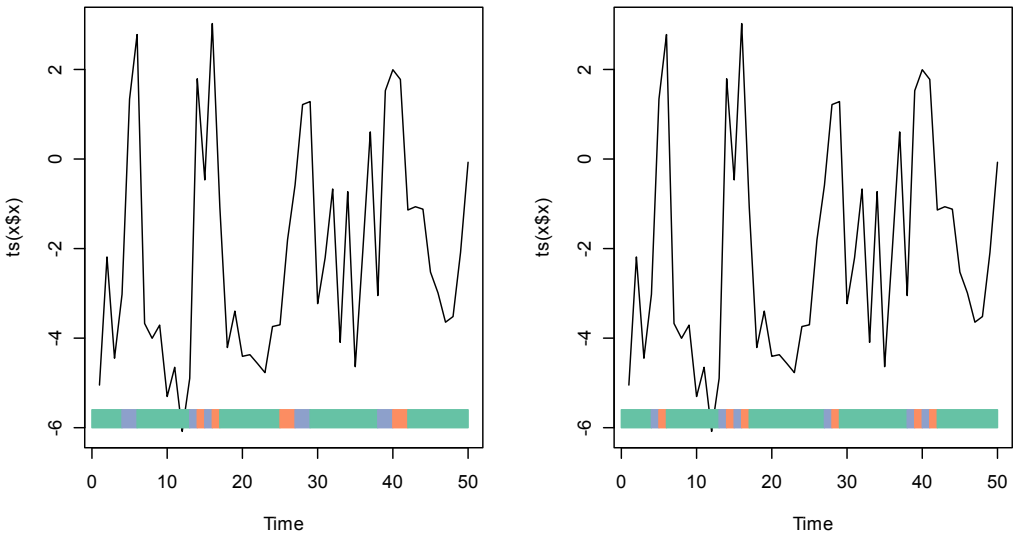

Figure 2: The simulated observation sequence and the true state sequence (left). The estimated state sequence from the Viterbi algorithm (right).

```
R> train2 <- simulate(model, nsim = 50, seed = 1234, rand.emis = rnorm.hsmm)
R> yhat <- predict(h1, train2)
R> mean(yhat$s != train2$s)
```

```
[1] 0.12
```

Plots of the simulated data along with the true and estimated state sequence can be seen in Figure 2.

The `predict()` returns a list in which the component named `s` contains the jointly most likely configuration of the states, which is found using a Viterbi algorithm, (Forney Jr 1973).

In some practical applications data consists of multiple sequences of observation. For example, in Section 5 we have multivariate data measured over time from several individual cows. The **mhsmm** package provides estimation and simulation routines for such data. For illustration, we generate three sequences of data, and fit the model with:

```
R> train = simulate(model, c(100, 20, 30), rand.emis = rnorm.hsmm)
R> h2 = hmmfit(train, startval, mstep = mstep.norm)
```

3. Theory of hidden Markov and semi-Markov models

This section contains a brief summary of Markov chains, hidden Markov and hidden semi-Markov models, or HMMs and HSMMs respectively. For a comprehensive introduction we refer to Rabiner (1989).

3.1. Discrete Markov chains

A discrete Markov chain is a random process (in discrete time) taking discrete values (states) from the state space S, that is, $S_t \in S = \{1, \ldots, J\}$ for $t = 1, 2, \ldots, T$. The process S_t is a Markov chain if it has the Markov property

$$P(S_{t+1} = s_{t+1}|S_0 = s_0, S_1 = s_1, \ldots, S_t = s_t) = P(S_{t+1} = s_{t+1}|S_t = s_t)$$

for any $s_0, s_1, \ldots, s_{t+1} \in \{1, \ldots, J\}$. Hence the state at any given time $t + 1$ depends on the previous states only through the state at time t.

Let $p_{ij} = P(S_{t+1} = j|S_t = i)$ with the properties $\sum_{j=1}^{J} p_{ij} = 1$ and $p_{ij} \geq 0$ denote the probability of jumping from state i at time t to state j at time $t + 1$. The matrix $P = (p_{ij})$ is then the transition matrix of the Markov chain. To fully specify the model we require the distribution of the initial state $\pi_i = P(S_0 = i)$.

The number of time steps spent in a given state is called the sojourn time. The probability of spending u consecutive time steps in state i under this model is

$$\begin{aligned} d_i(u) &= P(S_{t+u+1} \neq i, S_{t+u} = i, S_{t+u-1} = i, \ldots, S_{t+2} = i|S_{t+1} = i, S_t \neq i) \\ &= p_{ii}^{u-1}(1 - p_i). \end{aligned} \tag{2}$$

We call $d_i(u)$ the sojourn density. Hence the sojourn time is geometrically distributed for any Markov chain.

3.2. Hidden Markov models

Suppose we can only observe a variable X_t which is related to the state S_t but not the state itself. This situation is visualized in Figure 3. The conditional distribution of the observed variable X_t given the unobserved (or hidden) state S_t is referred to as the emission distribution. We refer to the parameters defining such a process as a *hidden Markov model*, henceforth referred to as an HMM. These models have been used for a variety of different applications, such as speech recognition (Rabiner 1989), weather modeling (Hughes, Guttorp, and Charles 1999) and DNA sequence analysis (Krogh, Mian, and Haussler 1994).

In addition to the parameters π and P which defines a Markov chain, a HMM also requires an emission distribution to be defined, that is

$$b_i(x_t) = P(X_t = x_t|S_t = i).$$

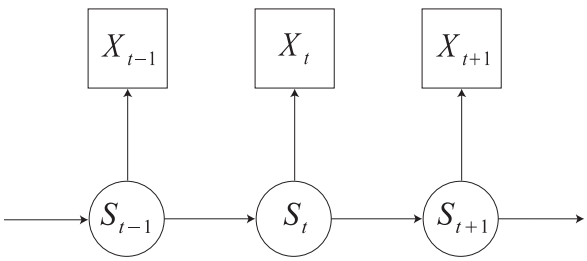

Figure 3: Visual representation of a hidden Markov process. X_t are some observed variables and S_t is the unobserved, hidden state.

For example, $b_i(x)$ may be a multivariate Gaussian distribution. As stated in Section 2, a HMM is hence specified by a triple $\theta = (\pi, P, b)$.

The Baum-Welch algorithm is the original procedure for estimating the parameters of a HMM (Baum, Petrie, Soules, and Weiss 1970). This technique was later grouped with a more general class of algorithms for incomplete data, named the expectation-maximization (EM) algorithm (Dempster, Laird, and Rubin 1977). We again point to Rabiner (1989) for a very clear overview.

3.3. Hidden semi-Markov models

In standard HMMs, the sojourn time is geometrically distributed (as shown by Equation (2)). In some real-world problems (see for example Section 5) this is an unrealistic and severe limitation because the probability of a state change depends on the time spent in the current state.

A possible solution to this issue is to explicitly estimate the duration density $d(u)$, producing what is referred as a *hidden semi-Markov model*, henceforth called an HSMM. Thus rather than having $d(u)$ defined by P as in (2) we model the $d(u)$ explicitely. Therefore, a HSMM is specified by a quadruple $\theta = (\pi, P, b, d)$.

Ferguson (1980) was the first to propose such models along with an algorithm to fit them, as Rabiner (1989) summarizes. Guédon (2003) developed a more efficient algorithm and a method to deal with right censoring which we have implemented.

The complete data likelihood of a HSMM is

$$P(X = x, S = s; \theta) = \pi_{s_1^*} d_{s_1^*}(u_1) \left\{ \prod_{r=2}^{R} p_{s_{r-1}^* s_r^*} d_{s_r^*}(u_r) \right\} p_{s_{R-1}^* s_R^*} D_{s_R^*}(u_R) \prod_{t=1}^{T} b_{s_t}(x_t), \tag{3}$$

where s_r^* is the rth visited state and u_r is the time spent in that state. Guédon proposed using the survivor function

$$D_i(u) = \sum_{v \geq u} d_i(v),$$

for the sojourn time in the last state so we do not have to assume the process is leaving a state immediately after time T. Using this survivor function has two advantages: It improves parameter estimation and, perhaps more importantly, it provides a more accurate prediction of the last state visited which is important for online applications where we wish to estimate the most recent state when monitoring a process.

As we have not observed the state sequence, maximising this likelihood constitutes an incomplete data problem. A local maximum can be found using the EM algorithm. We briefly outline the procedure below. The EM algorithm involves iterating over two steps until convergence. In the E-step, we calculate the expected complete data likelihood given the value of the parameters at iteration k and the observed data,

$$Q(\theta|\theta^{(k)}) = E[log(P(X = x, S = s; \theta))|X = x; \theta^{(k)}].$$

This term is typically broken down into a sum of terms involving subsets of the parameters. The M-step then involves choosing $\theta^{(k+1)}$ as the values that maximize $Q(\theta|\theta^{(k)})$. These steps are repeated until convergence.

3.4. The EM algorithm for hidden Markov models

A local maximum of the HMM likelihood (1) can be found via the EM algorithm through the following steps:

E-step: The E-step involves estimating two terms: (1) The probability of being in state i at time t given the observed sequence,

$$\gamma_t(i) = P(S_t = i|X = x; \theta), \tag{4}$$

and (2) the probability that the process left state i at time t and entered state j at $t+1$ given the observed sequence,

$$\xi_t(i,j) = P(S_t = i, S_{t+1} = j|X = x; \theta). \tag{5}$$

These values can be calculated via a dynamic programming method known as the forward-backward algorithm which has complexity $O(J^2T)$ as Rabiner (1989) discusses.

M-step: Based on (4) and (5) the initial transition probabilities are estimated as

$$\hat{\pi}'_i = \gamma_0(i) \text{ and } \hat{p}'_{ij} = \frac{\sum_{t=1}^{T-1} \xi_t(i,j)}{\sum_{t=1}^{T-1} \sum_{i \neq j} \xi_t(i,j)}. \tag{6}$$

Estimates for the parameters of the emission distribution are, of course, dependent on the choice of distribution. If we assume X_t are normally distributed given $S_t = i$, that is, $X_t|S_t = i \sim N(\mu_i, \sigma_i^2)$, then the parameters μ_i and σ_i^2 can be estimated as

$$\hat{\mu}_i = \frac{\sum_{t=1}^{T} \gamma_t(i) x_t}{\sum_{t=1}^{T} \gamma_t(i)} \text{ and } \hat{\sigma}_i = \frac{\sum_{t=1}^{T} \gamma_t(i)(x_t - \hat{\mu}_i)^2}{\sum_{t=1}^{T} \gamma_t(i)}. \tag{7}$$

Equations (6) and (7) are implemented in the `mstep.norm()` function in the **mhsmm** package.

The **mhsmm** package is extensible in that users can specify custom distributions. See Section 6 for examples.

3.5. The EM algorithm for hidden semi-Markov models

Parameter estimation for HSMMs is more complicated than for HMMs, both in terms of the mathematical description and in terms of the computational effort required. The EM algorithm for HSMMs is as follows:

E-step: Calculate the E-steps for HMMs as given in (4) and (5). Furthermore, we also need the expected number of times a process spends u time steps in state j,

$$\begin{aligned} \eta_{iu} &= P(S_u \neq i, S_{u-v} = i, v = 1, \ldots, u|X = x; \theta) \\ &\quad + \sum_{t=1}^{T} P(S_{t+u+1} \neq i, S_{t+u-v} = i, v = 0, \ldots, u-1, S_t \neq i|X = x; \theta). \end{aligned} \tag{8}$$

Guédon (2003) provides a version of the forward-backward algorithm for estimating (8) which is implemented in the **mhsmm** package. The algorithm has worst-case complexity $O(JT(J+T))$. However if we restrict the maximum possible sojourn time to a moderate value M this is reduced to $O(JT(J+M))$. For example, in one of the simulation examples of Section 4, we know sojourns of length greater than 500 are impossible for all practical purposes, so we set $M = 500$.

M-step: Calculate the M-steps for HMMs as given in (6) and (7). In addition we also need to estimate the state duration density. Guédon provides derivations for $d_i(u)$ as a non-parametric probability mass function using (8) as

$$d_i(u) = \frac{\eta_{iu}}{\sum_v \eta_{iv}}$$

but then proposes an ad-hoc solution for using parametric distributions with η_{iu} which we have followed in **mhsmm**. One possibility is to use common discrete distributions with an additional shift parameter d that sets the minimum sojourn time ($d \geq 1$). For example, we may use the Poisson distribution with density,

$$d_j(u) = \frac{e^{\lambda}\lambda^{(u-d)}}{(u-d)!}.$$

We estimate $\bar{\lambda}_i = \sum_{v=1}^{T}(v-d)\eta_{iv}$ for all possible shift parameters, $d = 1, \ldots, \min(u : \eta_{iu} > 0)$, choosing the d which gives the maximum likelihood. Guédon states that this ad-hoc procedure works well in practice and we have found this to be the case in simulations. Such an approach is also possible for other common distributions.

Another possiblity is to assume that the sojourn times are Gamma distributed, that is, $U_r|S_r = i \sim \Gamma(a_i, b_i)$. For this case, we estimated the parameters as follows: The likelihood for the Gamma distribution can be maximized with respect to its parameters by solving,

$$\log(\hat{a}_i) - \psi(\hat{a}_i) = \log(\bar{u}_i) - \overline{\log u_i},$$

where $\psi()$ is the digamma function. We use

$$\bar{u}_i = \frac{\sum_u \eta_{iu} u}{\sum_u \eta_{iu}} \text{ and } \overline{\log u_i} = \frac{\sum_u \eta_{iu} \log(u)}{\sum_u \eta_{iu}}$$

and then solve the equation using Newton's method (Choi and Wette 1969). This methodology is implemented in the `gammafit()` function. The scale parameter is estimated as $\hat{b}_i = \bar{u}_i/\hat{a}_i$.

4. Further simulation examples

This section contains several simulation examples, each illustrating features of the package.

4.1. Shifted Poisson sojourn distribution

We simulate data from a HSMM with a shifted Poisson sojourn distribution and Gaussian emission distribution. First we create a model using `hsmmspec()`. Data simulated using this model is shown in Figure 4.

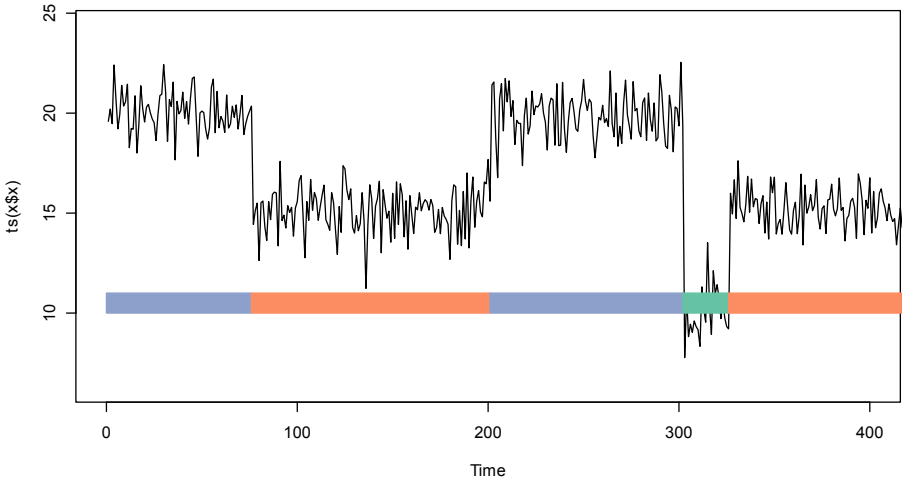

Figure 4: Simulated data from a hidden semi-Markov model with a shifted Poisson sojourn distribution and Gaussion emission distribution. That horizontal bar shows the different states while the curve shows the simulated values from the emission distribution.

```
R> J <- 3
R> init <- c(0, 0, 1)
R> P <- matrix(c(0, 0.1, 0.4, 0.5, 0, 0.6, 0.5, 0.9, 0), nrow = J)
R> B <- list(mu = c(10, 15, 20), sigma = c(2, 1, 1.5))
R> d <- list(lambda = c(10, 30, 60), shift = c(10, 100, 30),
+    type = "poisson")
R> model <- hsmmspec(init, P, parms.emis = B, sojourn = d,
+    dens.emis = dnorm.hsmm)
R> train <- simulate(model, nsim = 100, seed = 123456,
+    rand.emis = rnorm.hsmm)
```

We then estimate the parameters using some starting values (which can also be set using the `hsmmspec()` function):

```
R> start.pois <- hsmmspec(
+    init = rep(1/J, J),
+    transition = matrix(c(0, .5, .5, .5, 0, .5, .5, .5, 0), nrow = J),
+    parms.emis = list( mu=c(4, 12, 23), sigma = c(1, 1, 1)),
+    sojourn = list(lambda = c(9, 25, 40), shift = c(5, 95, 45),
+      type = "poisson"),
+    dens.emis = dnorm.hsmm)
R> M <- 500
R> h.poisson <- hsmmfit(train, start.pois, mstep = mstep.norm, M = M)
R> plot(h.poisson$loglik, type = "b", ylab = "Log-likelihood",
+    xlab = "Iteration")
R> summary(h.poisson)
```

```
Starting distribution =
[1] 0.0e+00 3.8e-16 1.0e+00

Transition matrix =
      [,1] [,2] [,3]
[1,] 0.000 0.65 0.35
[2,] 0.028 0.00 0.97
[3,] 0.465 0.53 0.00

Sojourn distribution parameters =
$lambda
[1]  6.59148 28.44452 54.97052

$shift
[1]  14 100  36

$type
[1] "poisson"


Emission distribution parameters =
$mu
[1]  9.97767 15.02050 20.00953

$sigma
[1] 1.742085 1.012044 1.496591

R> predicted <- predict(h.poisson, train)
R> table(train$s, predicted$s)

      1    2    3
  1 428    0    0
  2 491 4133    0
  3 323    0 3574

R> mean(predicted$s != train$s)

[1] 0.09095988
```

In this case we knew that the sojourn distribution was shifted Poisson and the EM algorithm has performed well in estimating the parameters (Figure 5, left).

4.2. Nonparametric sojourn distribution

Cases may arise where we do not know the sojourn distribution. We can estimate a non-parameteric sojourn distribution, perhaps as an initial step before deciding on a parametric distribution.

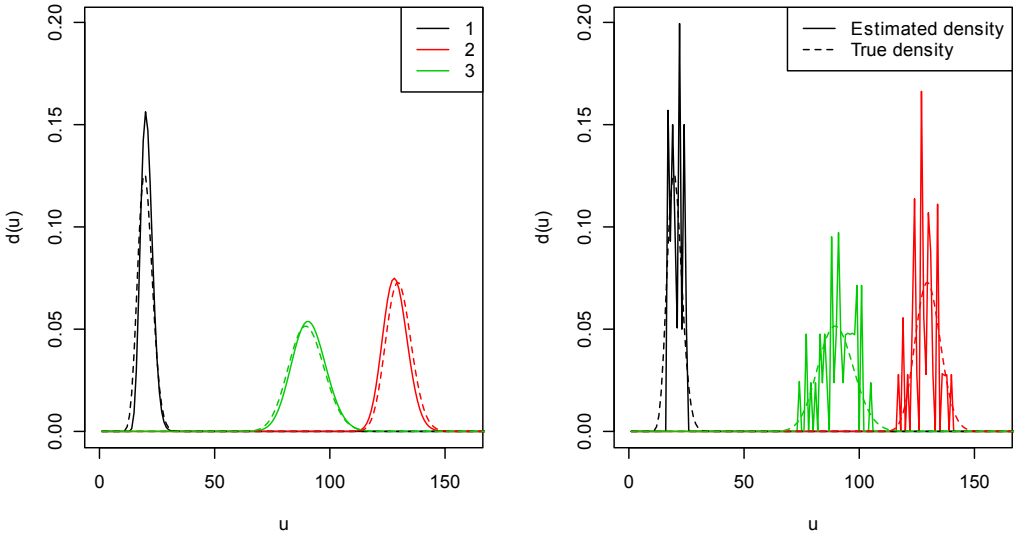

Figure 5: Theoretical and estimated sojourn densities for the parametric sojourn distribution model (left). Parametric theoretical and estimated non-parametric sojourn densities for the same simulated data set (right).

We can estimate a non-parametric sojourn distribution as in (3.5). The parameters for such a distribution are a $M \times J$ matrix, with entries (u, j) corresponding to $d_j(u)$. A good starting value to use is a uniform distribution covering the range of reasonable values for the sojourns. We can view the estimated non-parametric sojourn densities in Figure 5, right.

```
R> d <- cbind(dunif(1:M, 0, 50), dunif(1:M, 100, 175), dunif(1:M, 50, 130))
R> start.np <- hsmmspec(
+    init = rep(1/J, J),
+    transition = matrix(c(0, .5, .5, .5, 0, .5, .5, .5, 0), nrow = J),
+    parms.emis = list(mu = c(4, 12, 23), sigma = c(1, 1, 1)),
+    sojourn = list(d = d, type = "nonparametric"),
+    dens.emis = dnorm.hsmm)
R> h.np <- hsmmfit(train, start.np, mstep = mstep.norm, M = M,
+    graphical = TRUE)
```

5. Detecting reproductive status of dairy cows

5.1. The ovarian cycle in cattle

The ovarian cycle in cattle takes approximately 21 days (but with a large variation) and can be divided into four stages, which can be further grouped into two longer stages (see Table 1). The use of days is an insufficiently granular timescale for the oestrus stage, as oestrus lasts

Time	Stage	Meta-phase	Activity	Progesterone
Day 0	oestrus	follicular	high	low
Days 1-4	metoestrus		normal	
Days 5-18	dioestrus	luteal	normal	high
Days 18-20	pro-oestrus		normal	

Table 1: Table displaying the ovarian stages of a cow and the related variables. Ovulation occurs within the first day of metoestrus.

between 6 and 30 hours (Ball and Peters 2004). Ovulation occurs on the day following oestrus, so identifying oestrus allows a farm manager to know when to artificially inseminate a cow.

Following ovulation a structure called the corpus luteum forms in the ovary. The corpus luteum produces the hormone progesterone and remains in the ovary until a few days prior to the next ovulation (the luteal phase), at which time it degenerates rapidly. That is, progesterone is high during the luteal phase and low during the follicular phase. Progesterone directly reflects the biological processes occurring in the ovary and is a very useful indicator of reproductive status. Progesterone can be automatically measured from milk samples, with milking occurring every 6 - 20 hours (in robotic milking systems). A typical progesterone profile after calving can be viewed in Figure 6 (bottom).

In the period leading up to ovulation, the cow will try to attract the attention of a bull by standing to be mounted, mounting other cows and being mounted by other cows. This is sometimes referred to as standing heat and is traditionally how a stockman would identify a cow that is about to ovulate, allowing them to proceed with artificial insemination or bring the cow to a bull. This behaviour leads to an increase in the number of counts on a pedometer the cow is wearing, and so can be exploited for automated detection of oestrus. Having a

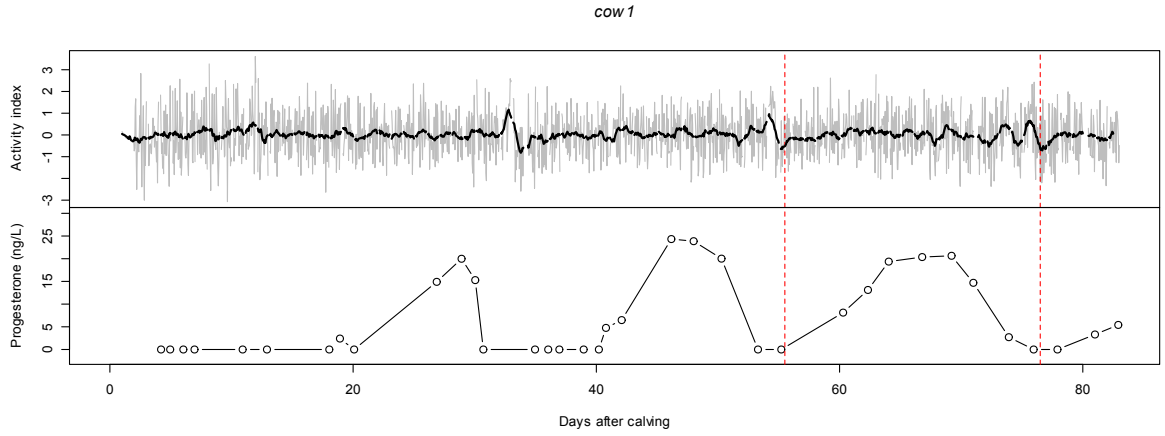

Figure 6: The activity index from a pedometer against time since calving (grey), the black line is as 24 hour centered moving average (top). The progesterone concentrations over the same time and cow. The dashed vertical lines are times when an artificial insemination occurred (an indicator of oestrus). Note that these occur after a drop in progesterone and brief spike in activity levels (bottom).

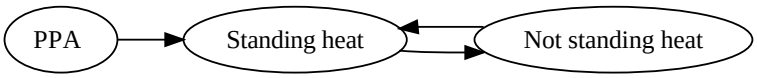

Figure 7: The underlying states for a univariate activity model.

stockman manually detect oestrus in farms with hundreds of cows is expensive and inaccurate, so automated systems are of great interest. These spikes in activity correspond with the drop in progesterone and can be seen in Figure 6 (top).

5.2. Analysis

We provide a simplified version of an analysis performed in O'Connell *et al.* (2011) using HSMMs to model reproductive data from dairy cows. Since the period of the ovarian cycle is irregular, a more conventional time series approach such as ARIMA is not suitable. We may hypothesize that the stages given in Table 1 are suitable states for a hidden Markov model but we know the states will not have geometrically distributed sojourn times. This is because the reproductive states of cows are not a memoryless process, since a follicular stage is likely to occur after 18 days in a luteal stage. Hence, a HSMM may be a suitable model for this data.

The dataset `reprocows` contains time-series data from seven cows with two measured variables, progesterone and the activity index derived in O'Connell *et al.* (2011). In addition, the dataset `reproai` contains days artifical insemination occurred for each cow and the dataset `reproppa` contains post-partum anoestrus lengths (in days) for 73 cows. We can use these auxilliary data sets for model validation and calculating start values, respectively.

```
R> data("reproai")
R> data("reprocows")
R> data("reproppa")
```

We fit a HSMM to the activity data, using the states;

$$S^{Activity} = \{\textit{post-partum anoestrus}, \textit{standing heat}, \textit{not standing heat}\}$$

(Figure 7). Validation for such a model is difficult, but we have two indicators:

- progesterone must be low for oestrus (and hence standing heat) to have occurred
- the artificial insemination at the end of each series was known to result in pregnancy

We begin by defining the model in Figure 7 and setting reasonable starting values for the emission distribution.

```
R> J <- 3
R> init <- c(1, 0, 0)
R> trans <- matrix(c(0, 0, 0, 1, 0, 1, 0, 1, 0), nrow = J)
R> emis <- list(mu = c(0, 2.5, 0), sigma = c(1, 1, 1))
```

We must also put the data into a `hsmm.data` object. To do this we need to calculate the length of the sequence from each individual cow (`N` here).

```
R> N <- as.numeric(table(reprocows$id))
R> train <- list(x = reprocows$activity, N = N)
R> class(train) <- "hsmm.data"
```

We need starting values for the sojourn distribution. We have found the Gamma distribution works well for these data. We can use the `reproppa` data set to estimate parameters for the initial states.

```
R> tmp <- gammafit(reproppa * 24)
```

As we are unsure of the parameters for the other two states, we just crudely use a uniform distribution with a reasonable range. The software will use this $d_j(u)$ for the initial E-step and then calculate Gamma parameters on the M-step. This ad-hoc procedure has been found to work well in simulation and in this practical application.

```
R> M <- max(N)
R> d <- cbind(dgamma(1:M, shape = tmp$shape, scale = tmp$scale),
+    dunif(1:M, 4, 30), dunif(1:M, 15 * 24, 40 * 24))
```

Finally, we create a `hsmmspec` object and fit the model with `hsmm`

```
R> startval <- hsmmspec(init, trans, emis, list(d = d, type = "gamma"),
+    dens.emis = dnorm.hsmm)
R> h.activity <- hsmmfit(train, startval, mstep = mstep.norm, maxit = 10,
+    M = M, lock.transition = TRUE)
```

We can view the predicted states for the first cow in Figure 8, the predicted states are consistent with the biological processes of the cow. As more formal validation, we compare the artifical insemination times with the timings of predicted standing heat (and therefore oestrus).

```
R> yhat <- predict(h.activity,train)$s
R> last.heat.hour <- cumsum(rle(yhat)$lengths)[rle(yhat)$values == 2]
R> cows.validation <- reprocows[last.heat.hour,]
R> dif <- list()
R> for(i in 1:nrow(reproai)) {
+    for(j in reproai$days.from.calving[reproai$id == i])
+      dif[[paste(i,j)]] <- j -
+        subset(cows.validation, id == i)$days.from.calving
+ }
R> dif <- unlist(dif)
R> dif <- dif[abs(dif) < 15]
R> plot(density(dif), xlab = "Standing heat time - AI time", main = "")
R> rug(jitter(dif))
R> dif
```

```
    1 55.52     1 76.53     2 72.53     2 97.54     3 55.52     3 77.53
 0.79166667  0.62500000  0.62500000  0.62500000  0.04166667  0.58333333
    4 53.52     4 73.53     5 43.52     5 45.52      6 32.5     7 51.52
 0.25000000  0.25000000 -1.66666667  0.33333333  0.25000000  0.00000000
```

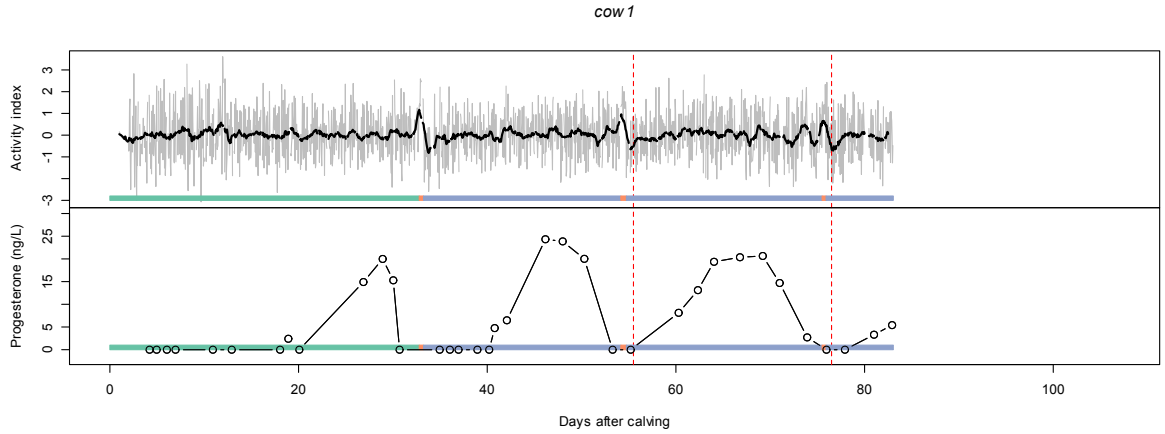

Figure 8: The cow data with the predicted states via the Viterbi algorithm. Notice that oestrus is predicted around the time progesterone drops and just prior to artificial insemination, that is, the model is consistent with biological knowledge.

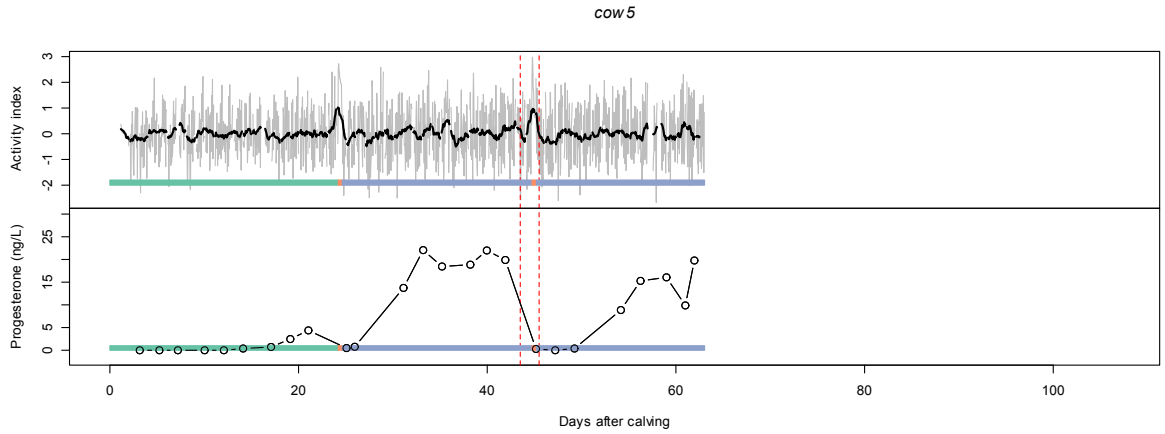

Figure 9: The cow data with the predicted states via the Viterbi algorithm.

All the artificial inseminations occurred within a day after the last predicted standing heat period except one (Figure 10, left), and this is consistent with the information in cattle reproduction texts, e.g. Ball and Peters (2004). If we view the "missed" insemination (Figure 9), we can see that two inseminations occurred around this oestrus, and that the first one was likely premature. We can view the estimated sojourn distributions in Figure 10, right.

A more rigorous analysis and validation of a larger version of this data set is presented in O'Connell *et al.* (2011).

6. User-defined extensions

Cases may arise where we wish to simulate or estimate a model whose emission distribution is not provided in the `mhsmm` packages. If users can provide

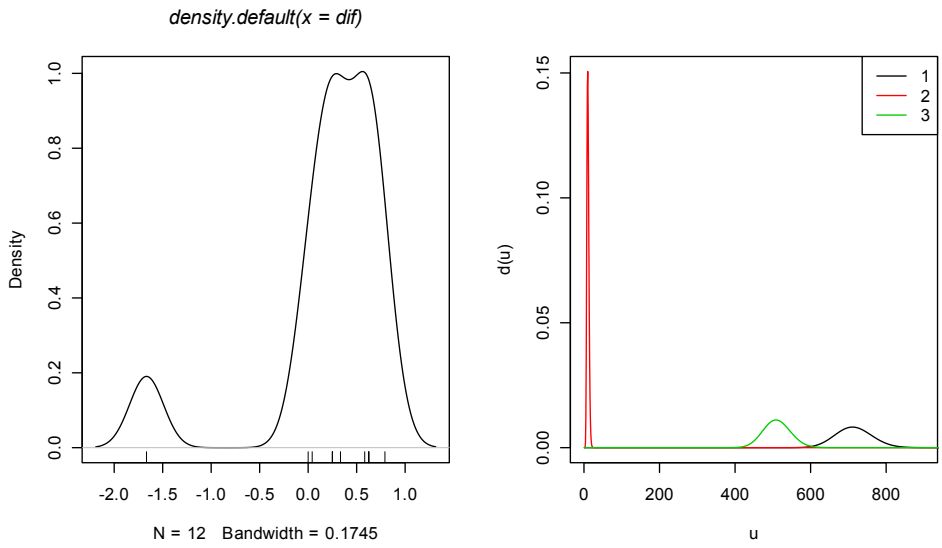

Figure 10: Kernel density estimate of the differences between last hour of predicted heat and artifical insemination times (left). The estimated sojourn distributions for each state (right).

- a function for generating values from the emission distribution
- a density function for the emission distribution
- implement a M-step for the emission distribution (see Section 3)

then they may use their own custom emission distributions. We provide two brief examples, we show how to do this with a HMM that has Poisson emission distribution and a HSMM with a Multivariate normal emission distribution.

6.1. Poisson emission distribution

We assume X_t are Poisson distributed given $S = i$, that is, $X_t|S_t = i \sim P(\lambda_i)$. Then λ_i can be reestimated via the equation,

$$\hat{\lambda}_i = \frac{\sum_{t=1}^{T} \gamma_t(i) x_t}{\sum_{t=1}^{T} \gamma_t(i)}.$$

We implement this in the function `mstep.pois`. Functions for the emission M-step take two arguments, `x`, the vector or dataframe of observed data and `wt` which is a $T \times K$ matrix representing the values $\gamma_t(j)$ in (4). The return value is a list corresponding to the `$emission` slot in a `hsmmspec` or `hmmspec` object.

```
R> mstep.pois <- function(x, wt)
+   list(lambda = apply(wt, 2, function(w) weighted.mean(x, w)))
```

Functions to simulate from and for calculating densities can be defined as:

```
R> rpois.hsmm <- function(j, model) rpois(1, model$parms.emission$lambda[j])
R> dpois.hsmm <- function(x, j, model)
+    dpois(x, model$parms.emission$lambda[j])
```

The `rpois.hsmm` function simulates data for state `j` from `model`, where `model` is a `hmmspec` or `hsmmspec` class. This class contains an emission list which should have the relevant parameters of the emission distribution. Similarly, `dpois.hsmm` calculates the densities for the observation vector (or data.frame for multivariate data) `x`, given state `j` for `model`.

```
R> J <- 2
R> init <- rep(1/J, J)
R> P <- matrix(c(.3, .2, .7, .8), nrow = J)
R> B <- list(lambda = c(5,10))
R> model <- hmmspec(init, P, parms.emis = B, dens.emis = dpois.hsmm)
R> train <- simulate(model, 1000, seed = 1, rand.emis = rpois.hsmm)
R> start.val <- hmmspec(init = rep(1/J, J),
+    trans = matrix(1/J, nrow = J, ncol = J),
+    parms.emis = list(lambda = c(2, 14)),
+    dens.emis = dpois.hsmm)
R> h1 <- hmmfit(train, start.val, mstep = mstep.pois)
R> summary(h1)


init:
 1 0

transition:
      [,1]  [,2]
[1,] 0.403 0.597
[2,] 0.213 0.787

emission:
$lambda
[1]  5.205141 10.129875
```

6.2. Multivariate normal emission distribution

In the case of multivariate data with p variables, the functions will expect the data `x` to be a matrix or dataframe of dimension $\sum_{T_n}^{N} \times p$ (for N sequences of length T_n). First we define a M-step.

```
R> mstep.mvnorm <- function(x, wt) {
+    emission <- list(mu = list(), sigma = list())
+    for(i in 1:ncol(x)) {
+      tmp <- cov.wt(x, wt[, i])
+      emission$mu[[i]] <- tmp$center
+      emission$sigma[[i]] <- tmp$cov
+    }
+    emission
+  }
```

Notice here we have used lists to hold the parameter values of each state rather than a vector. Users can design custom emission distributions. We then provide functions to generate from the distribution and calculate the density:

```
R> rmvnorm.hsmm <- function(j, model) rmvnorm(1,
+    mean = model$parms.emission$mu[[j]],
+    sigma = model$parms.emission$sigma[[j]])
R> dmvnorm.hsmm <- function(x, j, model) dmvnorm(x,
+    mean = model$parms.emission$mu[[j]],
+    sigma = model$parms.emission$sigma[[j]])
```

In this example, we will simulate from a two state HSMM with Gamma distributed sojourn times and the multivariate Gaussion distribution we have just defined. We will generate multiple observations sequences.

```
R> J <- 2
R> init <- c(1, 0)
R> P <- matrix(c(0, 1, 1, 0), nrow = J)
R> B <- list(mu = list(c(2, 3), c(3, 4)),
+    sigma = list(matrix(c(4, 2, 2, 3), ncol = J), diag(J)))
R> d <- list(shape = c(10, 25), scale = c(2, 2), type = "gamma")
R> model <- hsmmspec(init, P, parms.emis = B, sojourn = d,
+    dens.emis = dmvnorm.hsmm)
R> train <- simulate(model, c(10, 12, 13, 14), seed = 123,
+    rand.emis = rmvnorm.hsmm)
R> plot(train)
```

Now we create some perturbed starting values and try to recreate the true model. Note that in the two state case, the embedded Markov chain must be cyclical for a HSMM. Suppose that we do not have reasonable starting values for the Gamma sojourn distribution and use a uniform distribution with a reasonable range of values as the initial $d_j(u)$.

```
R> init0 <- rep(1/J, J)
R> B0 <- list(mu = list(c(1, 2), c(2, 3)),
+    sigma = list(matrix(c(3, 1.5, 1.5, 2.5), ncol = J), diag(J) * 1.5))
R> M <- 200
R> d0 <- cbind(dunif(1:M, 1, 50), dunif(1:M, 20, 100))
R> startval <- hsmmspec(init0, P, parms.emis = B0,
+    sojourn=list(d = d0, type = "gamma"), dens.emis = dmvnorm.hsmm)
R> hmv <- hsmmfit(train, startval, mstep = mstep.mvnorm, M = 200)
R> summary(hmv)

Starting distribution =
[1] 1 0

Transition matrix =
     [,1] [,2]
```

```
[1,]    0    1
[2,]    1    0

Sojourn distribution parameters =
$shape
[1] 13.49753 37.71941

$scale
[1] 1.456604 1.357940

$type
[1] "gamma"


Emission distribution parameters =
$mu
$mu[[1]]
[1] 1.905447 2.892303

$mu[[2]]
[1] 3.022554 4.022453


$sigma
$sigma[[1]]
         [,1]     [,2]
[1,] 4.150780 2.151244
[2,] 2.151244 2.995578

$sigma[[2]]
              [,1]          [,2]
[1,]  9.974962e-01 -5.556017e-05
[2,] -5.556017e-05  1.008613e+00
```

7. Summary and perspectives

In this paper we have presented the **mhsmm** package for R through several examples. We have also outlined the theory behind the hidden Markov and hidden semi-Markov models and we have described the estimation algorithms in some detail. In particular, we have shown that **mhsmm** is extensible as we have tried to facilitate the design and use of custom emission distributions. These features come perhaps at the cost of some simplicity and ease of use.

The creation the **mhsmm** package was motivated by the work on detecting reproductive status of dairy cows, (O'Connell *et al.* 2011) where two indicators of oestrus were used for estimating the reproductive status of cows, (see also Section 5). Problems of this type are commen, for example, in modern highly efficient farming because modern sensor technology

allows for frequent online measurement of many indicators on a large group of animals. As an additional example, Højsgaard and Friggens (2010) consider estimating the degree of mastitis for dairy cows from a panel of three indicators (measured with different intensities). It is an ongoing activity to apply hidden semi-Markov models for monitoring the mastitis status of a cow. The **mhsmm** package allows for missing values among the observables and this facility allows different sampling intensities to be handled in a natural way.

Acknowledgments

This study was part of the BIOSENS project funded by the Danish Ministry of Food, Agriculture and Fisheries and the Danish Cattle Industry via Finance Committee Cattle.

References

Ball PJH, Peters AR (2004). *Reproduction in Cattle*. 3rd edition. Blackwell Publishing.

Baum LE, Petrie T, Soules G, Weiss N (1970). "A Maximization Technique Occurring in the Statistical Analysis of Probabilistic Functions of Markov Chains." *The Annals of Mathematical Statistics*, **41**(1), 164–171.

Bulla J, Bulla I, Nenadic O (2010). "**hsmm** – An R Package for Analyzing Hidden Semi-Markov Models." *Computational Statistics & Data Analysis*, **54**(3), 611–619.

Choi SC, Wette R (1969). "Maximum Likelihood Estimation of the Parameters of the Gamma Distribution and Their Bias." *Technometrics*, **11**(4), 683–690.

Dempster AP, Laird NM, Rubin DB (1977). "Maximum Likelihood from Incomplete Data via the EM Algorithm." *Journal of the Royal Statistical Society B*, **39**(1), 1–38.

Ferguson JD (1980). "Hidden Markov Analysis: An Introduction." In *Hidden Markov Models for Speech*. Institute for Defense Analyses, Princeton.

Forney Jr GD (1973). "The Viterbi Algorithm." *Proceedings of the IEEE*, **61**(3), 268–278.

Godin C, Guédon Y (2007). "**AMAPmod** Version 1.8 Reference Manual." URL http://amap.cirad.fr/amapmod/refermanual18/partHome.html.

Guédon Y (2003). "Estimating Hidden Semi-Markov Chains from Discrete Sequences." *Journal of Computational and Graphical Statistics*, **12**(3), 604–639.

Højsgaard S, Friggens NC (2010). "Quantifying Degree of Mastitis from Common Trends in a Panel of Indicators for Mastitis in Dairy Cows." *Journal of Dairy Science*, **93**(2), 582–592.

Hughes JP, Guttorp P, Charles SP (1999). "A Non-Homogeneous Hidden Markov Model for Precipitation Occurrence." *Journal of the Royal Statistical Society C*, **48**(1), 15–30.

Krogh A, Mian IS, Haussler D (1994). "A Hidden Markov Model that Finds Genes in E. coli DNA." *Nucleic Acids Research*, **22**(22), 4768–4778.

O'Connell J, Tøgersen FA, Friggens NC, Løvendahl P, Højsgaard S (2011). "Combining Cattle Activity and Progesterone Measurements Using Hidden Semi-Markov Models." *Journal of Agricultural, Biological and Ecological Statistics*. doi:10.1007/s13253-010-0033-7. Forthcoming.

Rabiner LR (1989). "A Tutorial on Hidden Markov Models and Selected Applications in Speech Recognition." *Proceedings of the IEEE*, **77**(2), 257–286.

R Development Core Team (2010). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org/.

Affiliation:

Jared O'Connell
Wellcome Trust Centre for Human Genetics
University of Oxford
Roosevelt Drive
Oxford, OX3 7BN, United Kingdom
E-mail: jared@well.ox.ac.uk

Søren Højsgaard
Department of Genetics and Biotechnology
Faculty of Agricultural Sciences
Aarhus University
8830 Tjele, Denmark
E-mail: sorenh@agrsci.dk
URL: http://gbi.agrsci.dk/~sorenh/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume 39, Issue 4 *Submitted:* 2009-03-02
March 2011 *Accepted:* 2010-09-29