Using the seqHMM package for mixture hidden Markov models

(Version 1.0)

Satu Helske and Jouni Helske
Department of Mathematics and Statistics, University of Jyväskylä, Finland

November 25, 2015

Contents

1 Introduction

This vignette is supplementary material to the paper by **?**. It shows more detailed examples on how to use features of the `seqHMM` package in R (**?**). The package is designed for fitting hidden (or latent) Markov models (HMMs) and mixture hidden Markov models (MHMMs) for social sequence data and other categorical time series. It supports models for one or multiple subjects with one or multiple interdependent sequences (channels). External covariates can be added to explain cluster membership in MHMMs. The `seqHMM` package provides functions for evaluating and comparing models, as well as functions for easy plotting of multichannel sequences and hidden Markov models. Other restricted variants of the MHMM can also be estimated, e.g. latent class models, binomial and multinomial regression models, Markov models, and mixture Markov models.

Package is written mostly in C++, and it supports parallel processing via OpenMP. For improved numerical accuracy, the recursive algoritms are computed in log-space, which reduces under- and overflow problems in forward–backward algorithm even more than classical scaling approach (which is still an option in `seqHMM` for improved performance in well-behaving problems).

We start by showing how the package is installed and then prepare the example data before illustrating features of the package. We try to cover a wide range of examples to show different ways of using the functions. More examples and information are available in the documentation of the function (or data) by typing `?function_name` or equivalently `help(function_name)`.

2 Installation

The seqHMM package is currently available in Github: `https://github.com/helske/seqHMM`. It can be easily installed in R via the `devtools` package by writing the following lines in R.

```
install.packages("devtools")
library(devtools)
install_github("helske/seqHMM")
```

After the installation (only needed the first time) the package is loaded with the `library` function.

```
library(seqHMM)
```

3 Example data

The example data `biofam` comes with the `TraMineR` package (**?**) which is installed automatically with the `seqHMM` package. It is a sample of 2000 individuals born in 1909–1972, constructed from the Swiss Household Panel survey in 2002. The data set contains sequences of family life states from age 15 to 30 (in columns 10 to 25) and a series of covariates.

The states numbered from 0 to 7 are defined from the combination of five basic states, namely living with parents (parents), left home (left), married (marr), having children (child), and divorced:

0 = ”parents”
1 = ”left”
2 = ”married”
3 = ”left+marr”
4 = ”child”
5 = ”left+child”
6 = ”left+marr+child”
7 = ”divorced”.

For `seqHMM` functions, the data needs to be given as an `stslist` object. Such state sequence object is created with the `seqdef` function from the `TraMineR` package. It

has attributes such as color palette and alphabet, and it also has specific methods for plotting, summarizing, and printing.

We use two versions of the same data; one with the original single-channel sequences and another version, where the same data is converted into a multichannel version.

3.1 Single-channel data

We start by showing a glimpse of the original sequence data (in columns 10–25) and then create a state sequence object. We set the start at age 15 and give labels to states.

```
library(TraMineR)

## Warning:  package 'TraMineR' was built under R version 3.2.2

data(biofam)

head(biofam[, 10:25])

##      a15 a16 a17 a18 a19 a20 a21 a22 a23 a24 a25 a26 a27 a28 a29 a30
## 1167   0   0   0   0   0   0   0   0   0   3   6   6   6   6   6   6
## 514    0   1   1   1   1   1   1   1   1   1   1   3   6   6   6   6
## 1013   0   0   0   0   0   0   0   1   1   1   1   1   3   6   6   6
## 275    0   0   0   0   0   1   1   1   1   1   1   1   1   1   1   1
## 2580   0   0   0   0   0   1   1   1   1   1   1   1   1   6   6   6
## 773    0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0

biofam.seq <- seqdef(
  biofam[, 10:25], start = 15,
  labels = c("parent", "left", "married", "left+marr", "child",
             "left+child", "left+marr+child", "divorced"))
```

3.2 Multichannel data

For showing a more complex example with multichannel sequence data, we provide an R file which creates three separate sequences from the `biofam` data. The `biofam3c` object includes a list with three sequence data sets and a data frame with covariates. Note that the divorced state in the original data does not give information on children on residence, so the those are determined according to the preceeding states.

```
data(biofam3c)


marr.seq <- seqdef(biofam3c$married, start = 15,
                   alphabet = c("single", "married", "divorced"))
child.seq <- seqdef(biofam3c$children, start = 15,
                    alphabet = c("childless", "children"))
left.seq <- seqdef(biofam3c$left, start = 15,
                   alphabet = c("with parents", "left home"))
```

4 Plotting multichannel sequence data

The `TraMineR` package provides several plotting options for simple single-channel sequence data with the `seqplot` function, but there are no easy options for plotting multichannel data. In `seqHMM`, such data can be plotted in several ways. We start by choosing colors for the observed states in each channel.

```
attr(marr.seq, "cpal") <- c("violetred2", "darkgoldenrod2", "darkmagenta")
attr(child.seq, "cpal") <- c("darkseagreen1", "coral3")
attr(left.seq, "cpal") <- c("lightblue", "red3")
```

4.1 Stacked sequence plots

First we plot annual state distributions with the `ssplot` function (ssp for Stacked Sequence Plot). The `ssplot` function accepts two types of sequence plots available in the

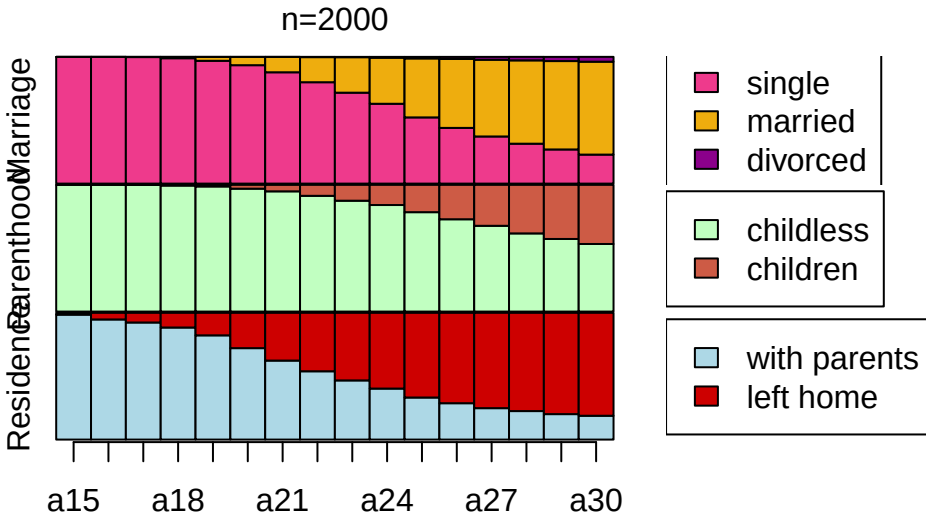


Figure 1: State distribution plots for multichannel data drawn with the `ssplot` function.

`seqplot` function from `TraMineR`: argument `type = "d"` gives state distributions and `type = "I"` shows sequence index plots. If the list objects are given a name, those are printed as labels; otherwise the number of the channel is plotted as a label. Another option is to use the `ylab` argument to change the labels. The number of sequences is automatically printed unless either the `title` or the `title.n` argument is set to `FALSE`. Figure 1 shows the default plot for three-channel data.

```
ssplot(list("Marriage" = marr.seq, "Parenthood" = child.seq,
       "Residence" = left.seq))
```

Another option is to define function arguments with the `ssp` function and then use previously saved arguments for plotting with a simple `plot` method. In figure 2 sequences are sorted according to the beginning of the marriage sequence (the first channel). Legends are omitted from the plot. The `ylab.pos` defines the positions of the channel labels.

```
ssp2 <- ssp(
  list(marr.seq, child.seq, left.seq),
  type = "I",title = "Sequence index plots",
```

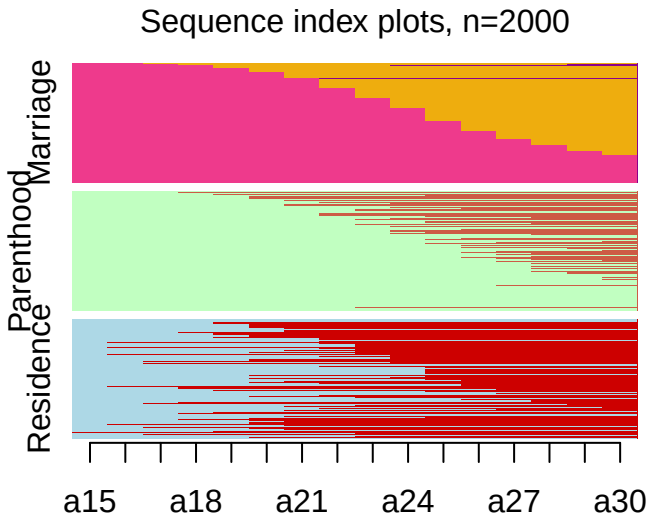


Figure 2: Sequence index plots for multichannel data defined with the `ssp` function and plotted with the `plot` method.

```
  sortv = "from.start", sort.channel = 1,
  withlegend = FALSE, ylab.pos = c(1, 1.5, 1),
  ylab = c("Marriage", "Parenthood", "Residence"))
plot(ssp2)
```

4.2 Plotting sequence data in a grid

We can also plot several `ssp` objects together. At first we define each `ssp` plot and then use the `gridplot` function to arrange them in a grid. Here we plot state distributions and sequence plots for women and men in a grid with two columns and three rows, of which the bottom row is reserved for the legends. The number of rows and/or columns is determined automatically, if one or both are omitted from the call. The plots are positioned column-wise by default, but here we change to row-wise plotting order.

We start by defining the first plot, state distributions for women. We remove borders from the bars, and omit legends and the number of sequences from the title. The other

plots are easy to define with the `update` function – the user only needs to type the arguments that are different to the first call.

```
# Preparing plots for women's state distributions
ssp_f_d <- ssp(
  x = list(marr.seq[biofam$sex == "woman",],
           child.seq[biofam$sex == "woman",],
           left.seq[biofam$sex == "woman",]),
  plots = "obs", type = "d", border = NA, withlegend = FALSE,
  title = "State distributions for women", title.n = FALSE,
  ylab = c("Marriage", "Parenthood", "Residence"),
  xlab = "Age", xtlab = 15:30)

# Same plot, but sequences instead of state distributions
ssp_f_I <- update(
  ssp_f_d, type = "I", sortv = "mds.obs",
  title = "Sequences for women", title.n = TRUE)

# State distributions with men's data
ssp_m_d <- update(
  ssp_f_d, title = "State distributions for men",
  x = list(marr.seq[biofam$sex == "man",],
           child.seq[biofam$sex == "man",],
           left.seq[biofam$sex == "man",]))

# Men's sequences
ssp_m_I <- update(
  ssp_m_d, type = "I", sortv = "mds.obs",
  title = "Sequences for men", title.n = TRUE)

gridplot(
```

```
  list(ssp_f_d, ssp_f_I, ssp_m_d, ssp_m_I), ncol = 2,
  byrow = TRUE, row.prop=c(0.42, 0.42, 0.16))
```

The `gridplot` function is able to create legends automatically using the states given in the first `ssp` object. If the objects contain different data with different states, legends can be included in the `ssp` objects and omitted from the `gridplot`. Figure 4 illustrates such plot using both the original `biofam` data and the three-channel version.

```
ssp_d <- ssp(
  biofam.seq, border = NA, title.n = FALSE,
  title = "State distributions (original states)",
  yaxis = TRUE, ylab = FALSE,
  withlegend = "bottom", ncol.legend = 2)

ssp_I <- ssp(
  list("Marriage" = marr.seq, "Parenthood" = child.seq,
       "Residence" = left.seq),
  type = "I", sortv = "from.end", title = "Sequences",
  xaxis = FALSE, legend.prop = 0.5)

gridplot(
  list(ssp_d, ssp_I), ncol = 2, withlegend = FALSE, col.prop = c(0.5, 0.5))
```

4.3 Converting multichannel to single-channel data

The `mc_to_sc_data` function converts multichannel sequence data to single channel data. At each time point of each individual, the states in each channel are combined into one. Note that here the number of combined observations (10 states) is larger than in the original data (8 states), because we have split the original divorced state into three.

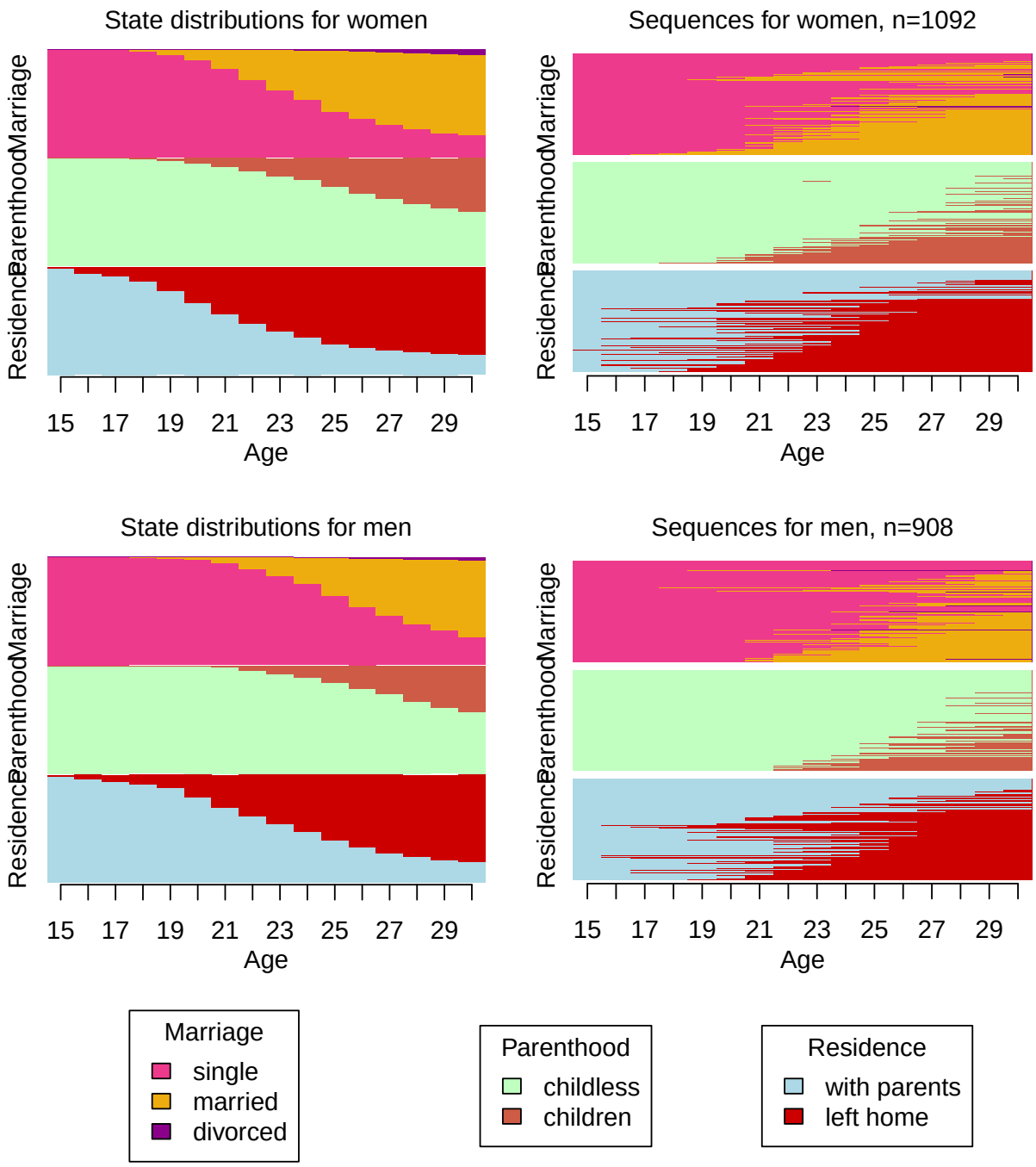


Figure 3: Plotting several `ssp` plots in a grid with the `gridplot` function.

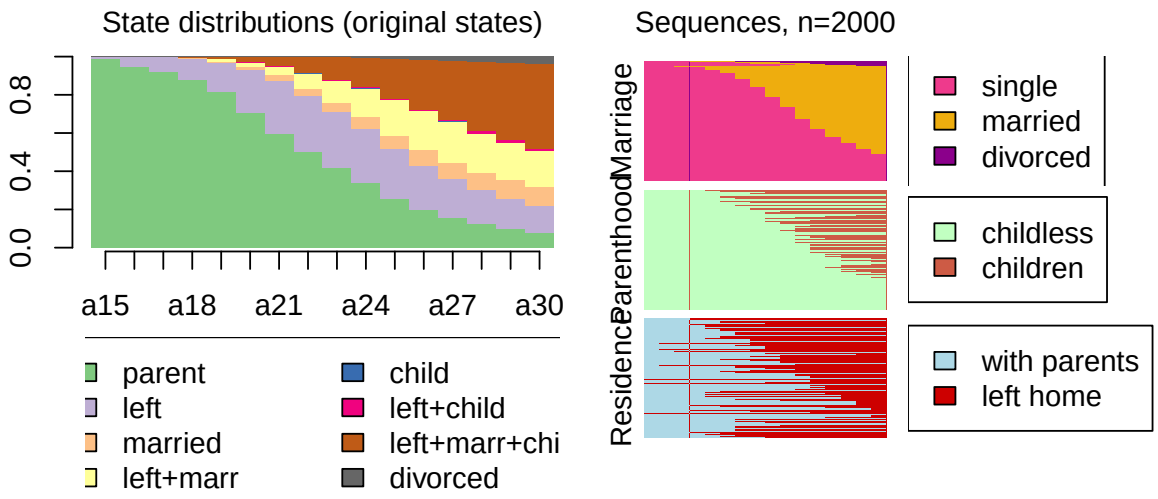


Figure 4: Two different sequence data sets drawn with the `gridplot` function.

Also single-channel data can be plotted with the `ssplot` and `gridplot` functions. Figure 5 illustrates the state distributions. We give more space for the legend to fit the labels, and set new labels for the x axis.

```
sc_data <- mc_to_sc_data(list(marr.seq, child.seq, left.seq))


ssplot(
  sc_data, type = "d", ylab = "Proportion", yaxis = TRUE,
  xtlab = 15:30, xlab = "Age", title = "Combined states",
  legend.prop = 0.4)
```

5 Hidden Markov models

Hidden Markov models consists of *observed states*, which are regarded as probabilistic functions of *hidden states*. These hidden states cannot be observed directly, but only through the sequence(s) of observations, since they emit the observations on varying probabilities. A discrete first order hidden Markov model for a single sequence can be characterized by the following:

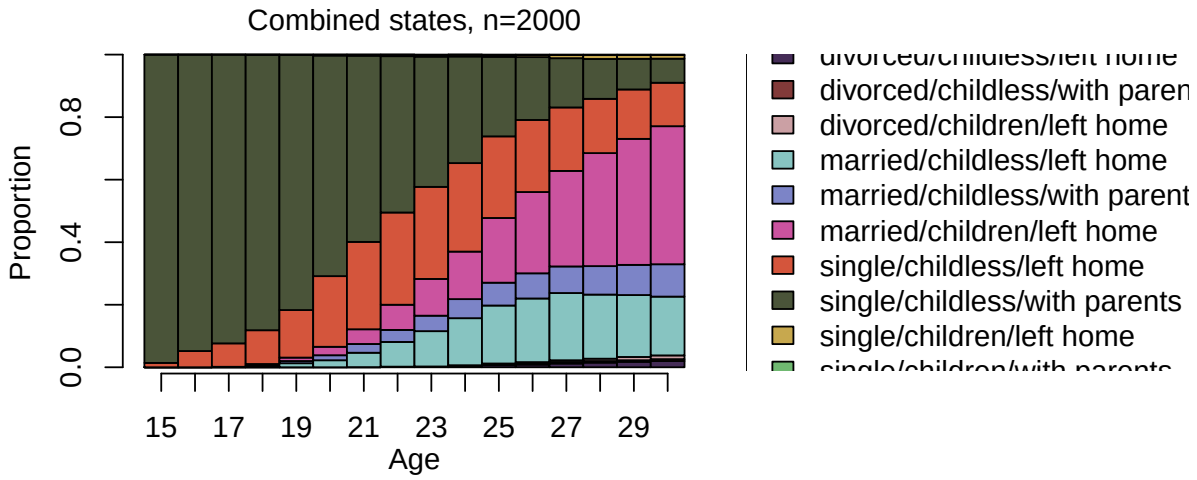


Figure 5: Three-channel `biofam3c` data converted to single-channel data.

- *Observed state sequence* $\mathbf{y} = y_1, y_2, \ldots, y_T$ with observed states $m \in \{1, 2, \ldots, M\}$.
- *Hidden state sequence* $\mathbf{z} = z_1, z_2, \ldots, z_T$ with hidden states $s \in \{1, 2, \ldots, S\}$.
- *Transition matrix* $A = \{a_{sr}\}$ where a_{sr} is the probability of moving from the hidden state s at time $t-1$ to the hidden state r at time t:

  $$a_{sr} = P(z_t = r | z_{t-1} = s); \quad s, r = 1, \ldots, S.$$

  We only consider homogeneous HMMs where the transition probabilities a_{sr} are constant over time.

- *Emission matrix* $B = \{b_s(m)\}$ where $b_s(m)$ is the probability of the hidden state s emitting the observed state m:

  $$b_s(m) = P(y_t = m | z_t = s); \quad s = 1, \ldots, S; m = 1, \ldots, M.$$

- *Initial probabilities* $\pi = \{\pi_s\}$, where π_s is the probability of starting from the hidden state s:

  $$\pi_s = P(z_1 = s); \quad s = 1, \ldots, S.$$

The (first order) Markov assumption states that the hidden state transition probability at time t only depends on the hidden state at the previous time point $t-1$:

$$P(z_t | z_{t-1}, \ldots, z_1) = P(z_t | z_{t-1}). \tag{1}$$

Also, the observation at time t is only dependent on the current hidden state, not on previous hidden states or observations:

$$P(y_t|y_{t-1},\ldots,y_1,z_t,\ldots,z_1) = P(y_t|z_t). \tag{2}$$

Generalization to case with multiple sequences and multiple channels is straightforward. Sequences $\mathbf{y}_i, i = 1,\ldots,N$ are assumed to be identically distributed but mutually independent given the states, each individual having their own hidden state sequence. In multichannel case, for each individual i there are C parallel sequences. The model has one transition matrix A but separate emission matrices $B_1,\ldots,B_C$ for each channel. We assume that the observed states in different channels at a given time point t are independent of each other given the hidden state at t, i.e. $P(\mathbf{y}_{it}|z_{it}) = P(y_{it1}|z_{it})\cdots P(y_{itC}|z_{it})$.

Before a HMM can be fitted, the user must define model specifications: the number of hidden states and initial values for parameters for emission, transition, and initial probabilities. Zero values in starting parameters are treated as fixed during the estimation.

We start by showing how HMMs are fitted for the simple single-channel data and then move on to a more complex example with three channels.

5.1 HMMs for single-channel data

Our goal here is to fit a model where hidden states describe a more general life stage, during which individuals are more likely to be in certain observable states. We expect that the life stages are somehow related to age, so constructing starting values from observed state frequences by age group seems like an option worth a try. We fit a model with four hidden state using age groups 15–18, 19–23, 24–27, and 28–30. Amount of 0.1 is added to each value in case of zero-frequencies in some categories (at this point we do not want to fix any parameters to zero) and each row is divided with its sum so that the row sums equal to 1.

```r
# Starting values for the emission matrix
sc_emiss <- matrix(NA, nrow = 4, ncol = 8)
sc_emiss[1,] <- seqstatf(biofam.seq[, 1:4])[, 2] + 0.1
sc_emiss[2,] <- seqstatf(biofam.seq[, 5:9])[, 2] + 0.1
sc_emiss[3,] <- seqstatf(biofam.seq[, 10:13])[, 2] + 0.1
sc_emiss[4,] <- seqstatf(biofam.seq[, 14:16])[, 2] + 0.1
sc_emiss <- sc_emiss / rowSums(sc_emiss)

# Starting values for the transition matrix
sc_trans <- matrix(
  c(0.80, 0.10, 0.05, 0.05,
    0.05, 0.80, 0.10, 0.05,
    0.05, 0.05, 0.80, 0.10,
    0.05, 0.05, 0.10, 0.80),
  nrow = 4, ncol = 4, byrow = TRUE)

# Starting values for initial state probabilities
sc_inits <- c(0.9, 0.07, 0.02, 0.01)
```

The model is initialized with the `build_hmm` function. It checks that the data and matrices are of the right form and creates an object of class `hmm`.

```r
# Building a hidden Markov model with starting values
sc_model <- build_hmm(
  observations = biofam.seq, transition_probs = sc_trans,
  emission_probs = sc_emiss, initial_probs = sc_inits)
```

The `hmm` object can then be fitted with the `fit_model` function. The fitting function provides three estimation steps: 1) EM algorithm, 2) global optimization, and 3) local optimization. Any combination of these steps can be chosen but their order cannot be modified. The results from a former step are used as starting values in a latter.

```
sc_fit <- fit_model(sc_model)
```

By default, the `fit_model`function starts with a maximum of 100 iterations of the EM step, skips the global step, and finishes with the L-BFGS algorithm as a local estimator. For direct numerical estimation (steps 2 and 3), any optimization method available in the `nloptr` package (**??**) can be chosen.

If the global step is chosen, by default it uses the multilevel single-linkage method (MLSL) (**??**). It draws multiple random starting values and performs local optimization from each starting point. The LDS modification uses low-discrepancy sequences instead of random numbers as starting points and should improve the convergence rate (**?**).

There are some theoretical guarantees that the MLSL method shoud find all local optima in a finite number of local optimizations. Of course, it might not always succeed in a reasonable time. Also, it requires setting boundaries for the parameter space, which is not always straightforward. In `seqHMM` the transition, emission, and initial probabilities are estimated using unconstrained reparameterization using softmax function (a generalization of the logistic function), but good boundaries are essential for efficient use of the MLSL algorithm. If the boundaries are too strict, the global optimum cannot be found; if too wide, the probability of finding the global optimum is decreased. The `fit_model` function uses starting values or results from the preciding estimation step to adjust the boundaries. The EM algorithm can help in setting good boundaries, but in some cases it can also lead to worse results. For finding the best solution, it is advisable to try a couple of different settings; e.g. only EM, EM followed by MLSL, a couple of EM iterations followed by MLSL, and only MLSL.

It is also possible to automatically run the EM algorithm multiple times by randomizing starting values. By default, both transition and emission probabilities are modified by adding a random value from the $N(0, 0.25)$ distribution. Although not done by default, this method seems to perform very well as the EM algorithm is relatively fast compared to direct numerical estimation.

For parameter estimation, well known forward–backward recursions are used. As the straightforward implementation of these algorithms pose a great risk of under-

and overflow, typically forward probabilities are scaled such that there should be no underflow. Although this is often sufficient for forward algorithm, this can result an overflow problem in backward algorithm where the same scaling factors are used. Thus in addition to scaling approach, `seqHMM` uses logarithm scale for most of the computations, which further reduces the numerical unstabilities. On the other hand, as there is a need to back-transform to natural scale during the algorithms, the log-space approach seems to be typically about 3–4 times slower than scaling approach. For well-behaving models the scaling approach can still be used by setting argument `log_space` to `FALSE`.

Most of the functions relating to model estimation and state inference support parallel computation via OpenMP interface. The user can choose the number of parallel threads (e.g. the number of cores) to use for the specific task. By default, no parallelization is done.

The `fit_model` function returns the model, the log-likelihood, and estimation results from each step.

```
sc_fit$model

## Initial probabilities :
##       [,1]
## [1,] 0.986
## [2,] 0.014
## [3,] 0.000
## [4,] 0.000
##
## Transition probabilities :
##          to
## from      State 1 State 2 State 3 State 4
##   State 1   0.886  0.0546 0.03220  0.0275
##   State 2   0.000  0.8898 0.08342  0.0267
##   State 3   0.000  0.0000 0.78724  0.2128
```

```
##   State 4   0.000  0.0000 0.00136  0.9986
##
## Emission probabilities :
##            symbol_names
## state_names 0 1       2     3       4      5     6      7
##     State 1 1 0 0.00000 0.000 0.00000 0.0000 0.000 0.0000
##     State 2 0 1 0.00000 0.000 0.00000 0.0000 0.000 0.0000
##     State 3 0 0 0.00194 0.998 0.00000 0.0000 0.000 0.0000
##     State 4 0 0 0.21448 0.000 0.00279 0.0245 0.711 0.0473
```

5.2 HMMs for multichannel data

A HMM for multichannel data is fitted in a similar way using the same `build_hmm` and `fit_model` functions. Now we build a model for three-channel data with four hidden states. One emission matrix (with hidden states as rows and observed states as columns) is required for each channel, i.e. 4x3 matrix for marriages and 4x2 matrices for the other two. Also here the initial values are constructed from using the observed state frequences at different age categories (see the example of fitting HMM for single-channel data). Now we use an upper triangular matrix in the transition matrix indicating a left-to-right model where transitions back to previous states are not allowed. This seems like a valid option from a life-course prespective. Also, in the previous single-channel model of the same data the transition matrix was estimated almost upper triangular.

```
# Initial values for initial state probabilities
mc_inits <- c(0.90, 0.07, 0.02, 0.01)

# Initial values for transition matrix
mc_trans <- matrix(
  c(0.90, 0.06, 0.03, 0.01,
       0, 0.90, 0.07, 0.03,
```

```
       0,    0, 0.90, 0.10,
       0,    0,    0,    1),
  nrow = 4, ncol = 4, byrow = TRUE)

# Initial values for emission matrices
mc_B_marr <- matrix(NA, nrow=4, ncol=3)
mc_B_marr[1,] <- seqstatf(marr.seq[, 1:4])[, 2] + 0.1
mc_B_marr[2,] <- seqstatf(marr.seq[, 5:8])[, 2] + 0.1
mc_B_marr[3,] <- seqstatf(marr.seq[, 9:12])[, 2] + 0.1
mc_B_marr[4,] <- seqstatf(marr.seq[, 13:16])[, 2] + 0.1
mc_B_marr <- mc_B_marr / rowSums(mc_B_marr)

mc_B_child <- matrix(NA, nrow=4, ncol=2)
mc_B_child[1,] <- seqstatf(child.seq[, 1:4])[, 2] + 0.1
mc_B_child[2,] <- seqstatf(child.seq[, 5:8])[, 2] + 0.1
mc_B_child[3,] <- seqstatf(child.seq[, 9:12])[, 2] + 0.1
mc_B_child[4,] <- seqstatf(child.seq[, 13:16])[, 2] + 0.1
mc_B_child <- mc_B_child / rowSums(mc_B_child)

mc_B_left <- matrix(NA, nrow=4, ncol=2)
mc_B_left[1,] <- seqstatf(left.seq[, 1:4])[, 2] + 0.1
mc_B_left[2,] <- seqstatf(left.seq[, 5:8])[, 2] + 0.1
mc_B_left[3,] <- seqstatf(left.seq[, 9:12])[, 2] + 0.1
mc_B_left[4,] <- seqstatf(left.seq[, 13:16])[, 2] + 0.1
mc_B_left <- mc_B_left / rowSums(mc_B_left)

# Build HMM
mc_hmm <- build_hmm(
  observations = list(marr.seq, child.seq, left.seq),
  initial_probs = mc_inits, transition_probs = mc_trans,
```

```
  emission_probs = list(mc_B_marr, mc_B_child, mc_B_left),
  channel_names = c("Marriage", "Parenthood", "Residence"))

# Fit HMM
mc_fit <- fit_model(mc_hmm)
```

5.3 Evaluating and comparing HMMs

5.3.1 Log-likelihood

Log-likelihood is stored in the result from `fit_model`, but also the `logLik` function computes it. Models with the same number of parameters can be compared using log-likelihood: the bigger the value, the better the model.

```
mc_fit$logLik

## [1] -16854.16

logLik(mc_fit$model)

## 'log Lik.' -16854.16 (df=25)
```

Next we fit a couple of models with different estimation procedures and compare the results using log-likelihood. We first note that for this model the EM algorithm converges to optimum so the local optimization step is not necessary (the difference between the log-likelihoods is neglibigle, -5.7e-5):

```
mc_fit$em_results$logLik

## [1] -16854.16

-mc_fit$local_results$objective

## Error in -mc_fit$local_results$objective:  invalid argument to unary operator
```

Using only gradient-based optimization without the initial EM step produces substantially poorer fit:

```
mc_hmm_local <- fit_model(mc_hmm, em_step = FALSE)

## Error in fit_model(mc_hmm, em_step = FALSE): No method chosen for estimation.
Choose at least one from em_step, global_step, and local_step.

mc_hmm_local$logLik

## Error in eval(expr, envir, enclos):  object 'mc_hmm_local' not found
```

Next we will test the performance of the MLSL optimization by setting `global = TRUE`. Final polishing is done with the L-BFGS. The default estimation time is set to 60 seconds in global step, but usually it is advisable to run it longer. Depending on the performance of the computer, the optimizer may also be able to perform more or less evaluations in the given time limit. Setting `maxtime = 0` omits the time limit and performs the number of evaluations given for `maxeval`. The more complex the model, the more time the optimizer needs to throughly search the parameter space.

```
mc_hmm_global <- fit_model(mc_hmm, em_step = FALSE, global = TRUE,
  control_global = list(maxtime = 0, maxeval = 1000))

## Error in fit_model(mc_hmm, em_step = FALSE, global = TRUE, control_global
= list(maxtime = 0, :  object 'maxTM' not found

mc_hmm_global$logLik

## Error in eval(expr, envir, enclos):  object 'mc_hmm_global' not found
```

In this case the starting values are good enough so that the EM algorithm finds the best solution. The MLSL algorithm is much slower and doesn't quite find the optimum on its own during 1000 evaluations.

Automatic restarts of the EM algorithm are called by setting the value `restarts` in the `control_em` argument. By default, both transition and emission probabilities are

modified by adding a random numbers from a normal distribution with a mean of 0 and a standard error defined by `sd_restart` (0.25 by default). Rows of matrices are then scaled to sum to one. Here randomizing is only useful for a check-up since the original EM already finds the optimum.

```
set.seed(123)
mc_hmm_restarts <- fit_model(mc_hmm,
  control_em = list(restarts = 100, restart_transition = FALSE))

## Warning in fit_model(mc_hmm, control_em = list(restarts = 100, restart_transition
= FALSE)): Unknown names in control_em:  restarts, restart_transition

mc_hmm_restarts$logLik

## [1] -16854.16
```

5.3.2 Bayesian information criterion (BIC)

The `BIC` function gives the value of the Bayesian information criterion (BIC).

```
BIC(mc_fit$model)

## [1] 33967.66
```

BIC can be used to compare models with a different number of parameters, e.g. for choosing the best number of hidden states. We fit a model with three hidden states and compare the results to the four-state model.

```
mc_inits_3s <- c(0.90, 0.07, 0.03)

mc_A_3s <- matrix(
  c(0.90, 0.07, 0.03,
       0, 0.90, 0.10,
       0,    0,    1),
```

```
  nrow = 3, ncol = 3, byrow = TRUE)

mc_B_marr_3s <- matrix(NA, nrow=3, ncol=3)
mc_B_marr_3s[1,] <- seqstatf(marr.seq[, 1:5])[, 2] + 0.1
mc_B_marr_3s[2,] <- seqstatf(marr.seq[, 6:10])[, 2] + 0.1
mc_B_marr_3s[3,] <- seqstatf(marr.seq[, 11:16])[, 2] + 0.1
mc_B_marr_3s <- mc_B_marr_3s / rowSums(mc_B_marr_3s)

mc_B_child_3s <- matrix(NA, nrow=3, ncol=2)
mc_B_child_3s[1,] <- seqstatf(child.seq[, 1:5])[, 2] + 0.1
mc_B_child_3s[2,] <- seqstatf(child.seq[, 6:10])[, 2] + 0.1
mc_B_child_3s[3,] <- seqstatf(child.seq[, 11:16])[, 2] + 0.1
mc_B_child_3s <- mc_B_child_3s / rowSums(mc_B_child_3s)

mc_B_left_3s <- matrix(NA, nrow=3, ncol=2)
mc_B_left_3s[1,] <- seqstatf(left.seq[, 1:5])[, 2] + 0.1
mc_B_left_3s[2,] <- seqstatf(left.seq[, 6:10])[, 2] + 0.1
mc_B_left_3s[3,] <- seqstatf(left.seq[, 11:16])[, 2] + 0.1
mc_B_left_3s <- mc_B_left_3s / rowSums(mc_B_left_3s)

mc_hmm_3s <- build_hmm(
  observations = list(marr.seq, child.seq, left.seq),
  initial_probs = mc_inits_3s, transition_probs = mc_A_3s,
  emission_probs = list(mc_B_marr_3s, mc_B_child_3s, mc_B_left_3s),
  channel_names = c("Marriage", "Parenthood", "Residence"))

mc_fit_3s <- fit_model(mc_hmm_3s)

mc_fit_3s$logLik

## [1] -22617.08
```

```
BIC(mc_fit_3s$model)

## [1] 45410.52
```

BIC for the four-state model is 33967.66, which is less than 45410.52, BIC for the three-state model. We can conclude, that the model with four hidden states is better.

5.4 Converting multichannel models to single-channel models

The `mc_to_sc` function converts multichannel models to single-channel representations. It combines observed states and multiplies emission parameters across channels. Instead of three emission matrices of size 4×3, 4×2, and 4×2 we now have one emission matrix of size 4×10, since there are 10 state combinations in the new data.

```
sc_hmm <- mc_to_sc(mc_fit$model)
```

6 Plotting hidden Markov models

When the function is fitted, the `hmm` object can be easily plotted as a directed graph with the `plot` method. By default, each hidden state is represented as a pie chart with the (combinations of) emitted states as slices. States with emission probability less than 0.05 are combined into one category, unless the user specifies otherwise. By default, the initial state probabilities are given below the vertices, and the widths of the edges giving the transition probabilities vary according to the parameter values. If the `hmm` object has multiple channels, the `mc_to_sc` function is automatically used to convert the multichannel model into a single channel model during plotting. Figure 6 illustrates a default plot for the four-state model fitted earlier.

```
plot(mc_fit$model)
```

The `hmm` plots can be modified in various ways such as changing sizes and positions of vertices, thickness and curvature of edges, colors and fonts of labels, and position

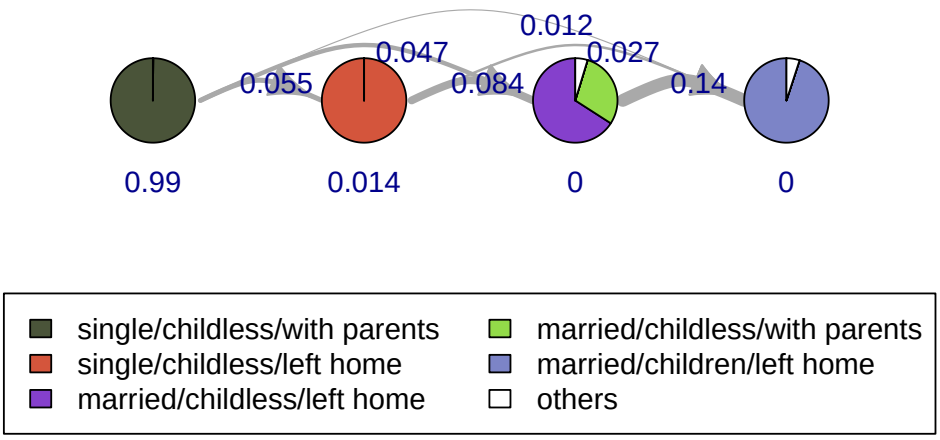


Figure 6: Plotting HMM as a directed graph with the `plot` method.

and contents of the legend. The `igraph` package (**?**) is used for plotting directed graphs, so most arguments of the `plot.igraph` function work in `plot.hmm` as well. In Figure 7 we draw bigger vertices, modify the legend, change curvatures of edges, and give a new label for the combined slice.

```
plot(
  mc_fit$model, vertex.size = 50, ncol.legend = 2, legend.prop = 0.4,
  edge.curved = c(0, 0.7, -0.6, 0, 0.7, 0),
  combined.slice.label = "States with probability < 0.05")
```

By default, the vertices are plotted vertically aligned but that can be changed with the `layout` argument. Ready-made options include horizontal and vertical positioning, but also coordinates and layout functions of the `igraph` package can be given to position the vertices. Coordinates are given in a two-column matrix, with x coordinates in the first column and y coordinates in the second. Arguments `xlim` and `ylim` set the lengths of the axes and `rescale = FALSE` prevents rescaling the coordinates to the $[-1,1] \times [-1,1]$ interval (the default).

In Figure 8 we change the positions of the vertex labels (initial probabilities), call for straight and thinner edges with smaller arrows, modify the legend, ask to plot all emitted states, and change colors.

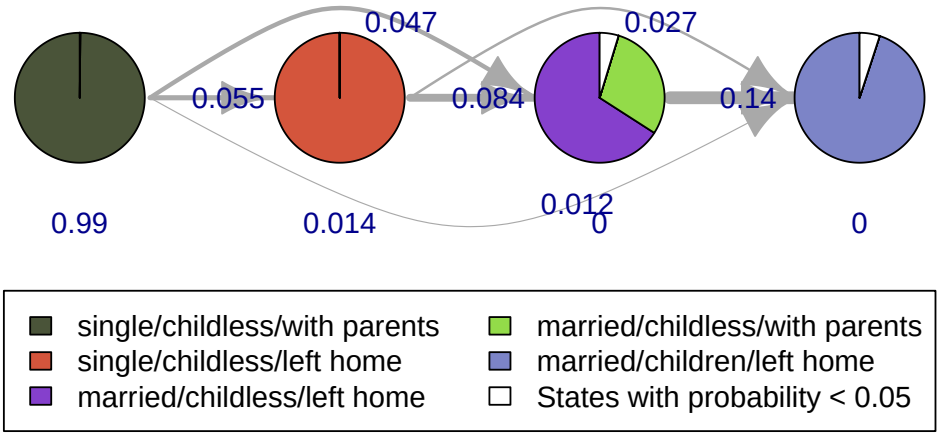


Figure 7: Another version of the same model as in Figure 6.

```
plot(
  mc_fit$model,
  layout = matrix(c(1,3,3,5,
                    0,-1,1,0), ncol=2),
  xlim = c(0.5, 5.5), ylim = c(-1.5, 1.5), rescale = FALSE,
  vertex.size = 50, vertex.label.pos = c(pi, pi/2, -pi/2, 0),
  edge.curved = FALSE, cex.edge.width = 0.8, edge.arrow.size = 1.2,
  withlegend = "top", legend.prop = 0.3, cex.legend = 1.1, ncol.legend = 2,
  combine.slices = 0, cpal = colorpalette[[15]][1:10])
```

A color must be provided for all (combinations of) observed states present in the data, whether or not shown in the graph. The `seqHMM` package uses the `colorpalette` data for defining the colors in the plots. It is a list of 60 ready-made color palettes with 1–60 distinct colors and is automatically loaded with the package and available for use. See also the `RColorBrewer` package for more color palettes with distinct colors. The `plot_colors` function is provided fot easy visualization of color palettes.

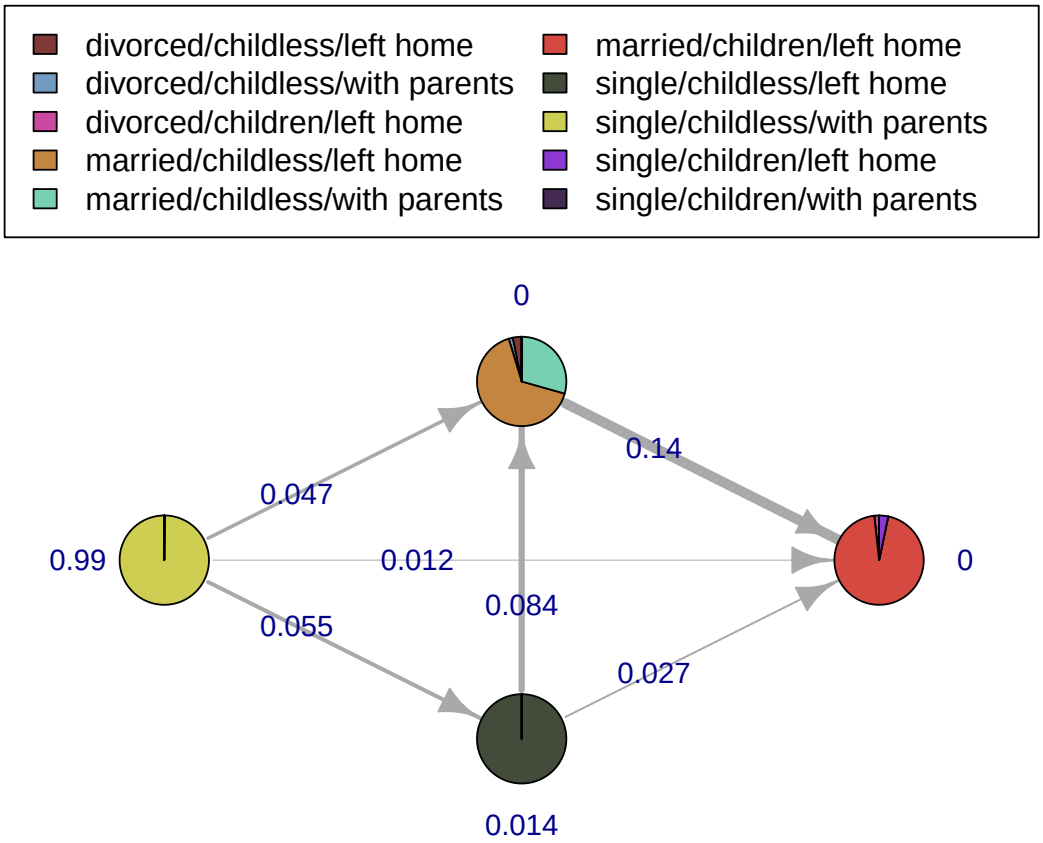


Figure 8: Another version of the same model as in Figure 6.

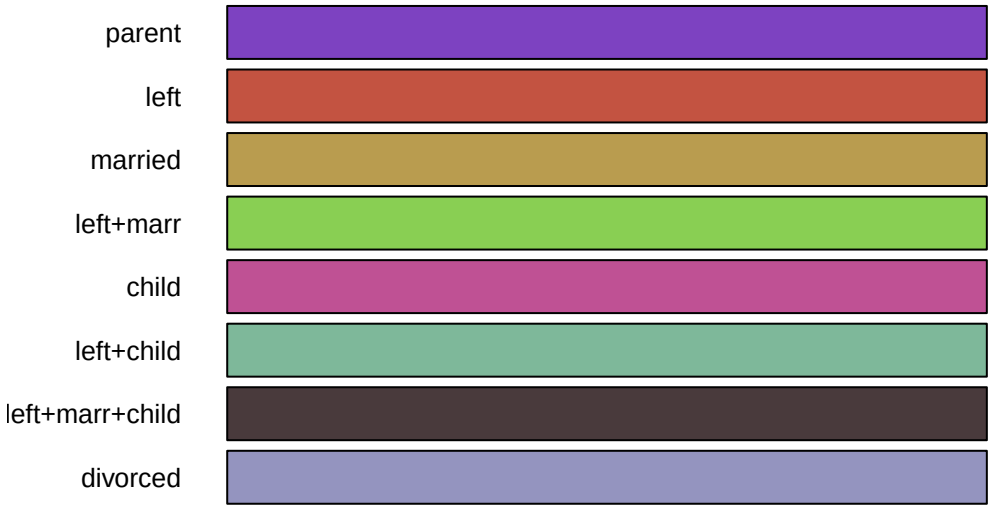


Figure 9: Plotting colorpalettes can be handy when choosing colors for states.

```
plot_colors(
  colorpalette[[8]],
  labels = c("parent", "left", "married", "left+marr", "child",
             "left+child", "left+marr+child", "divorced"))
```

The `ssplot` function also accepts an object of class `hmm`. It can be used for plotting observations and/or most probable paths of hidden states or their state distributions. The function automatically computes the most probable paths of hidden states with the `hidden_paths` function, if the user does not provide them. Here the sequences are sorted according to multidimensional scaling scores that are computed from optimal matching dissimilarities for hidden state sequences. Any dissimilarity method available in `TraMineR` can be used instead of the default (see the documentation of the `seqdef` function for more information).

```
ssplot(
  mc_fit$model, plots = "both", type = "I", sortv = "mds.hidden",
  xtlab = 15:30, xlab = "Age", title = "Observed and hidden state sequences")
```

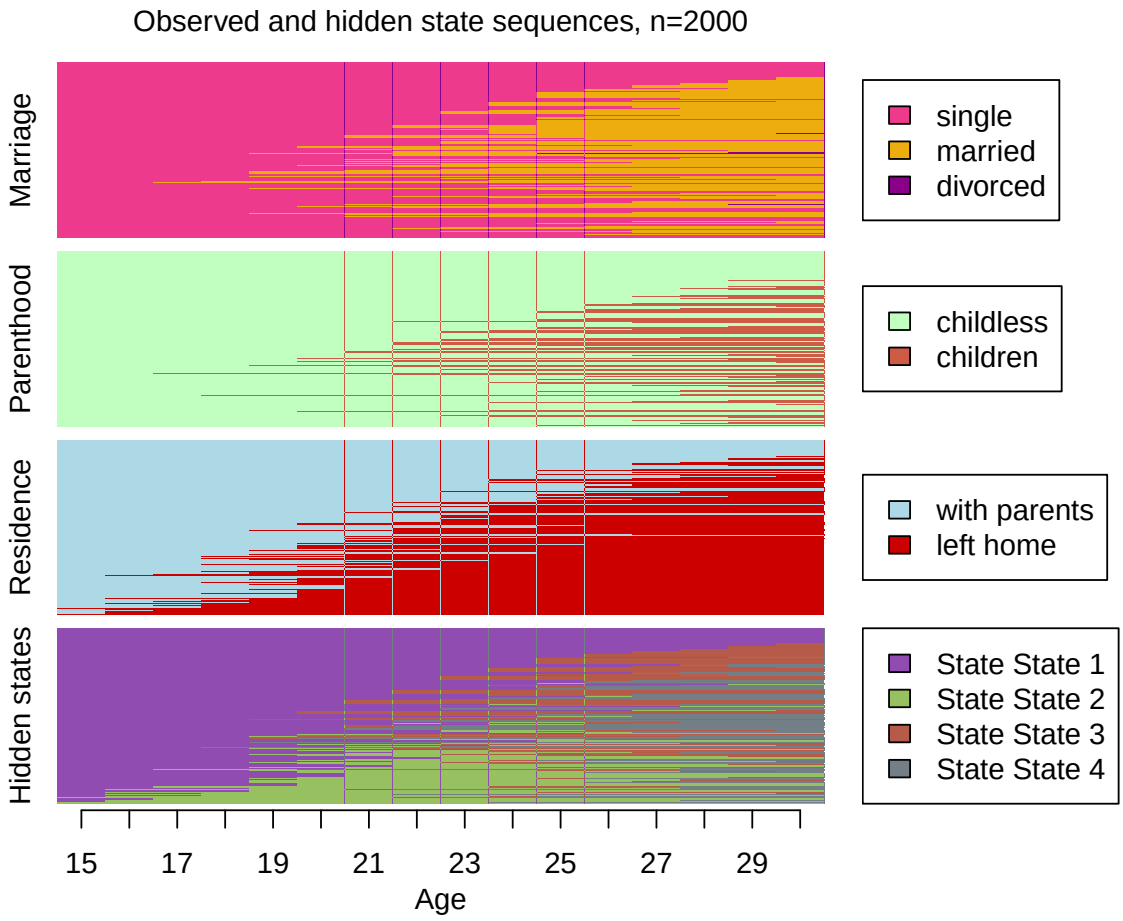


Figure 10: Using the `ssplot` function to draw observations and the most probable hidden state paths given the model.

7 Mixture hidden Markov models

Assume that we have a set of models (clusters) $\mathcal{M} = \{\mathcal{M}^1, \ldots, \mathcal{M}^K\}$, where $\mathcal{M}^k = \{\pi^k, A^k, B_1^k, \ldots, B_C^k\}$ for $k = 1, \ldots, K$. For each observation sequence $\mathbf{y}_i$, the prior cluster probability, i.e. the probability that sequence $\mathbf{y}_i$ is generated by submodel $\mathcal{M}^k$, is defined as multinomial distribution:

$$P(\mathcal{M}^k|\mathbf{x}_i) = w_{ik} = \frac{e^{\beta_k \mathbf{x}_i}}{1 + \sum_{j=2}^{K} e^{\beta_j \mathbf{x}_i}}, \tag{3}$$

where $\mathbf{x}_i$ consists of the covariates relating to sequence $\mathbf{y}_i$. The first cluster is set as the reference by fixing $\beta_1 = 0$. Note that by convention we use β when referring to regression coefficients.

The mixture hidden Markov model (MHMM) can be regarded as a special type HMM by combining the K submodels into one large hidden Markov model consisting of $\sum_{k=1}^{K} S_k$ states, where the initial state vector contains elements of form $w_k \pi^k$. Now the transition matrix is block diagonal

$$A = \begin{pmatrix} A^1 & 0 & \cdots & 0 \\ 0 & A^2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & A^K \end{pmatrix}, \tag{4}$$

where the diagonal blocks $A^k, k = 1, \ldots, K$, are square matrices containing the transition probabilities of one cluster. The off-diagonal blocks are zero matrices, so transitions between clusters are not allowed. Similarly, the emission matrices for each channel contains stacked emission matrices B^k.

The posterior cluster probabilities $P(M^k|\mathbf{y}_i, \mathbf{x}_i)$ can be obtained as

$$\begin{aligned} P(M^k|\mathbf{y}_i, \mathbf{x}_i) &= \frac{P(\mathbf{y}_i|M^k, \mathbf{x}_i) P(M^k|\mathbf{x}_i)}{P(\mathbf{y}_i|\mathbf{x}_i)} \\ &= \frac{P(\mathbf{y}_i|M^k, \mathbf{x}_i) P(M^k|\mathbf{x}_i)}{\sum_{j=1}^{K} P(\mathbf{y}_i|M^j, \mathbf{x}_i) P(M^j|\mathbf{x}_i)} = \frac{L_k^i}{L^i}, \end{aligned} \tag{5}$$

where L^i is the likelihood of the complete MHMM for individual i, and L_k^i is the likelihood of cluster k for individual i. These are straightforwardly computed from

forward probabilities. Posterior cluster probabilities can be used e.g. for computing classification tables.

MHMMs are fitted in a similar way to HMMs using the `build_mhmm` and `fit_model` functions. Functions such as `logLik`, `BIC`, `trim_model`, and `mc_to_sc` work for `mhmm` objects as well.

Even though missing observations can be handled in sequences, the covariates must be completely observed for each subject. Subjects with missing covariates are easily dropped from the data set with the `complete.cases` function.

```
# Starting values for initial state probabilities
initial_probs1 <- c(0.9, 0.07, 0.02, 0.01)
initial_probs2 <- c(0.9, 0.04, 0.03, 0.01, 0.01, 0.01)

# Starting values for transition probabilities
A1 <- matrix(
  c(0.80, 0.16, 0.03, 0.01,
       0, 0.90, 0.07, 0.03,
       0,    0, 0.90,  0.1,
       0,    0,    0,    1),
  nrow = 4, ncol = 4, byrow = TRUE)

A2 <- matrix(
  c(0.80, 0.10, 0.05, 0.03, 0.01, 0.01,
       0, 0.70, 0.10, 0.10, 0.05, 0.05,
       0,    0, 0.85, 0.01, 0.10, 0.04,
       0,    0,    0, 0.90, 0.05, 0.05,
       0,    0,    0,    0, 0.90,  0.1,
       0,    0,    0,    0,    0,    1),
  nrow = 6, ncol = 6, byrow = TRUE)

# Starting values for emission probabilities
```

```r
# Cluster 1
alphabet(child.seq) # Checking for the order of observed states

## [1] "childless" "children"

B1_child <- matrix(
  c(0.99, 0.01, # High probability for childless
    0.99, 0.01,
    0.99, 0.01,
    0.99, 0.01),
  nrow = 4, ncol = 2, byrow = TRUE)


alphabet(marr.seq)

## [1] "single"   "married"  "divorced"

B1_marr <- matrix(
  c(0.98, 0.01, 0.01, # High probability for single
    0.98, 0.01, 0.01,
    0.01, 0.98, 0.01, # High probability for married
    0.01, 0.01, 0.98), # High probability for divorced
  nrow = 4, ncol = 3, byrow = TRUE)


alphabet(left.seq)

## [1] "with parents" "left home"

B1_left <- matrix(
  c(0.99, 0.01, # High probability for living with parents
    0.01, 0.99, # High probability for having left home
    0.01, 0.99,
    0.01, 0.99),
  nrow = 4, ncol = 2, byrow = TRUE)
```

```
# Cluster 2
B2_child <- matrix(
  c(0.99, 0.01, # High probability for childless
    0.99, 0.01,
    0.99, 0.01,
    0.01, 0.99),
  nrow = 4, ncol = 2, byrow = TRUE)

B2_marr <- matrix(
  c(0.98, 0.01, 0.01, # High probability for single
    0.98, 0.01, 0.01,
    0.01, 0.98, 0.01, # High probability for married
    0.01, 0.70, 0.29),
  nrow = 4, ncol = 3, byrow = TRUE)

B2_left <- matrix(
  c(0.99, 0.01, # High probability for living with parents
    0.01, 0.99, # High probability for having left home
    0.01, 0.99,
    0.01, 0.99),
  nrow = 4, ncol = 2, byrow = TRUE)

# Cluster 3
B3_child <- matrix(
  c(0.99, 0.01, # High probability for childless
    0.99, 0.01,
    0.01, 0.99,
    0.99, 0.01,
    0.01, 0.99,
```

```
    0.01, 0.99),
  nrow = 6, ncol = 2, byrow = TRUE)

B3_marr <- matrix(
  c(0.98, 0.01, 0.01, # High probability for single
    0.98, 0.01, 0.01,
    0.98, 0.01, 0.01,
    0.01, 0.98, 0.01, # High probability for married
    0.01, 0.98, 0.01,
    0.01, 0.01, 0.98), # High probability for divorced
  nrow = 6, ncol = 3, byrow = TRUE)

B3_left <- matrix(
  c(0.99, 0.01, # High probability for living with parents
    0.01, 0.99,
    0.50, 0.50,
    0.01, 0.99,
    0.01, 0.99,
    0.01, 0.99),
  nrow = 6, ncol = 2, byrow = TRUE)

# Birth cohort
biofam3c$covariates$cohort <- cut(
  biofam3c$covariates$birthyr, c(1908, 1935, 1945, 1957))
biofam3c$covariates$cohort <- factor(
  biofam3c$covariates$cohort,
  labels = c("1909-1935", "1936-1945", "1946-1957"))

# Build MHMM
init_mhmm <- build_mhmm(
```

```
  observations = list(marr.seq, child.seq, left.seq),
  transition_probs = list(A1, A1, A2),
  emission_probs = list(list(B1_marr, B1_child, B1_left),
                        list(B2_marr, B2_child, B2_left),
                        list(B3_marr, B3_child, B3_left)),
  initial_probs = list(initial_probs1, initial_probs1, initial_probs2),
  formula = ~sex + cohort, data = biofam3c$covariates,
  cluster_names = c("Cluster 1", "Cluster 2", "Cluster 3"),
  channel_names = c("Marriage", "Parenthood", "Residence"))

set.seed(456)
fit_model <- fit_model(
  init_mhmm, control_em = list(restarts = 50, restart_transition = FALSE))

## Warning in fit_model(init_mhmm, control_em = list(restarts = 50, restart_transition
= FALSE)): Unknown names in control_em:  restarts, restart_transition
```

The `summary` method computes summaries of the MHMM, e.g. standard errors for covariates and prior and posterior probabilities for subjects. A `print` method shows some summaries of these: estimates and standard errors for covariates, log-likelihood and BIC, and information on most probable clusters and prior probabilities. Parameter estimates for transitions, emissions, and initial probabilities are omitted by default.

There are two types of standard errors which can be computed for regression coefficients. The conditional standard errors can be computed using analytical formulas for Hessian, but these do not take account the estimation uncertainty regarding the other model parameters. For unconditional standard errors which take account of possible correlation between the estimates of β and other model parameters, the Hessian is computed using finite difference approximation of the jacobian of the analytical gradients. Naturally, the latter option is computationally more demanding, but it should still be a preferred option in final analysis as it gives more realistic results.

```r
summary(fit_model$model, conditional_se = FALSE)
```

```
## Covariate effects :
## Cluster 1 is the reference.
##
## Cluster 2 :
##                   Estimate  Std. error
## (Intercept)          3.070       0.344
## sexwoman            -0.507       0.289
## cohort1936-1945      0.557       0.401
## cohort1946-1957      0.238       0.333
##
## Cluster 3 :
##                   Estimate  Std. error
## (Intercept)          2.170       0.353
## sexwoman            -0.244       0.305
## cohort1936-1945     -0.181       0.413
## cohort1946-1957     -0.879       0.346
##
## Log-likelihood: -12851.59   BIC: 26761.27
##
## Means of prior cluster probabilities :
## Cluster 1 Cluster 2 Cluster 3
##    0.0374    0.7739    0.1887
##
## Most probable clusters :
##              Cluster 1  Cluster 2  Cluster 3
## count               56       1655        289
## proportion       0.028      0.828      0.144
##
## Classification table :
```

```
## Mean cluster probabilities (in columns) by the most probable cluster (rows)
##
##           Cluster 1 Cluster 2 Cluster 3
## Cluster 1   0.99460   0.00511   0.00029
## Cluster 2   0.01128   0.93430   0.05442
## Cluster 3   0.00132   0.00449   0.99419
```

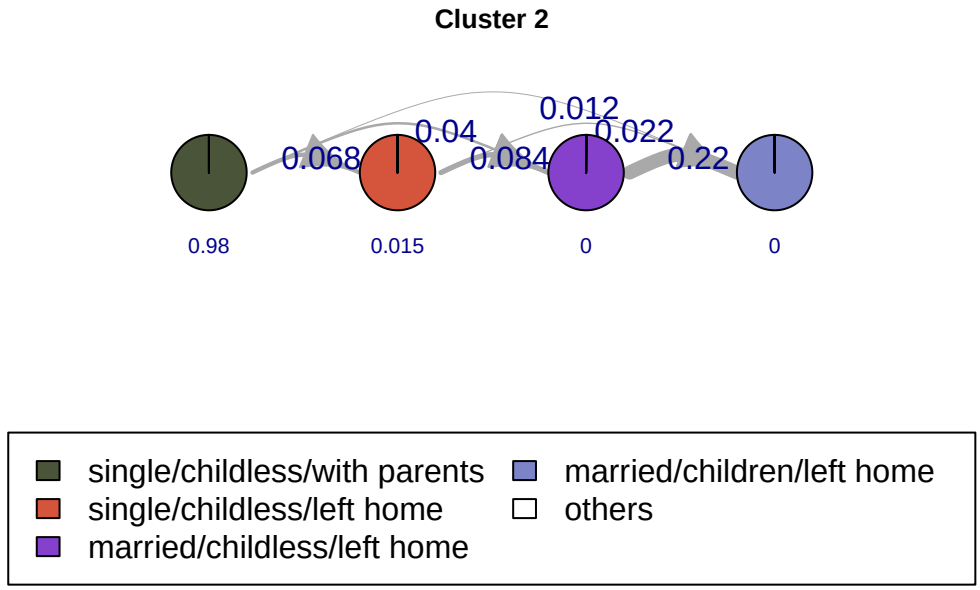
The classification table shows the mean probabilities of belonging to each cluster by the most probable cluster. The most probable cluster is determined by the posterior cluster probabilities. A good model shoud have high proportions in the diagonal. Here, for individuals assigned to cluster 1, the average probability for cluster 1 is 0.85, 0.15 for cluster 2, and close to 0 for cluster 3. The highest probability for the assigned cluster is 0.93 for cluster 3.

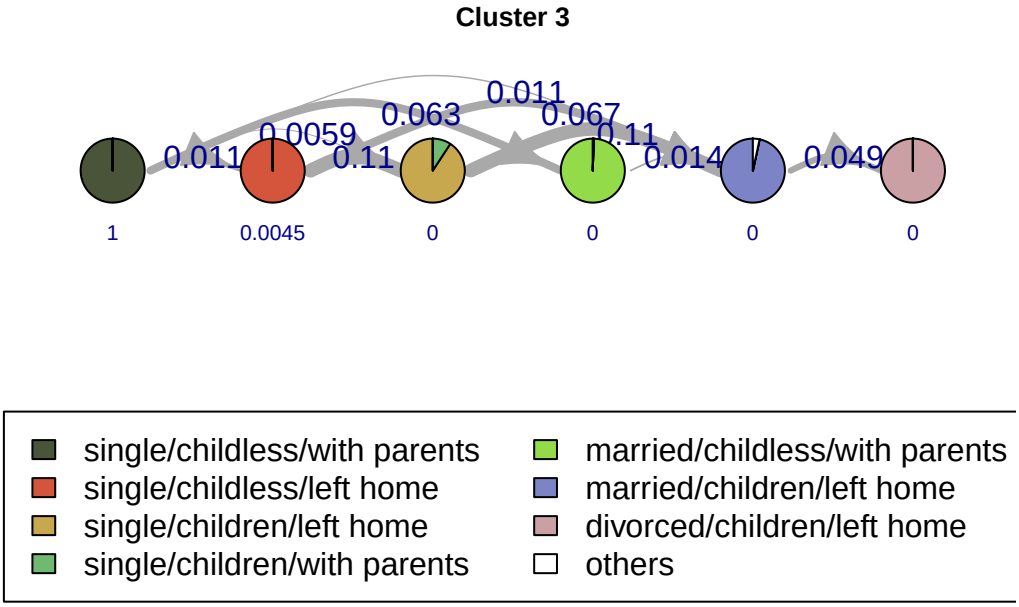
It is possible to convert a `mhmm` object into a list of separate `hmm` objects with the `separate_mhmm` function. These can then be plotted with functions designed for `hmm` class objects, e.g. `ssp` and `gridplot`. However, usually an easier option is to use plotting functions and methods designed for `mhmm` objects.

8 Plotting mixture hidden Markov models

Also MHMMs are plotted with the `plot` method. The user can choose between an interactive mode (`interactive=TRUE`), where the model for each cluster is plotted separately, and a combined plot with all models in one plot. The clusters are determined according to the `posterior_probs` function, which is called automatically. Clusters with no subjects get removed from the plot, and the function gives a warning. With the `which.plots` argument the user can choose which clusters to plot.

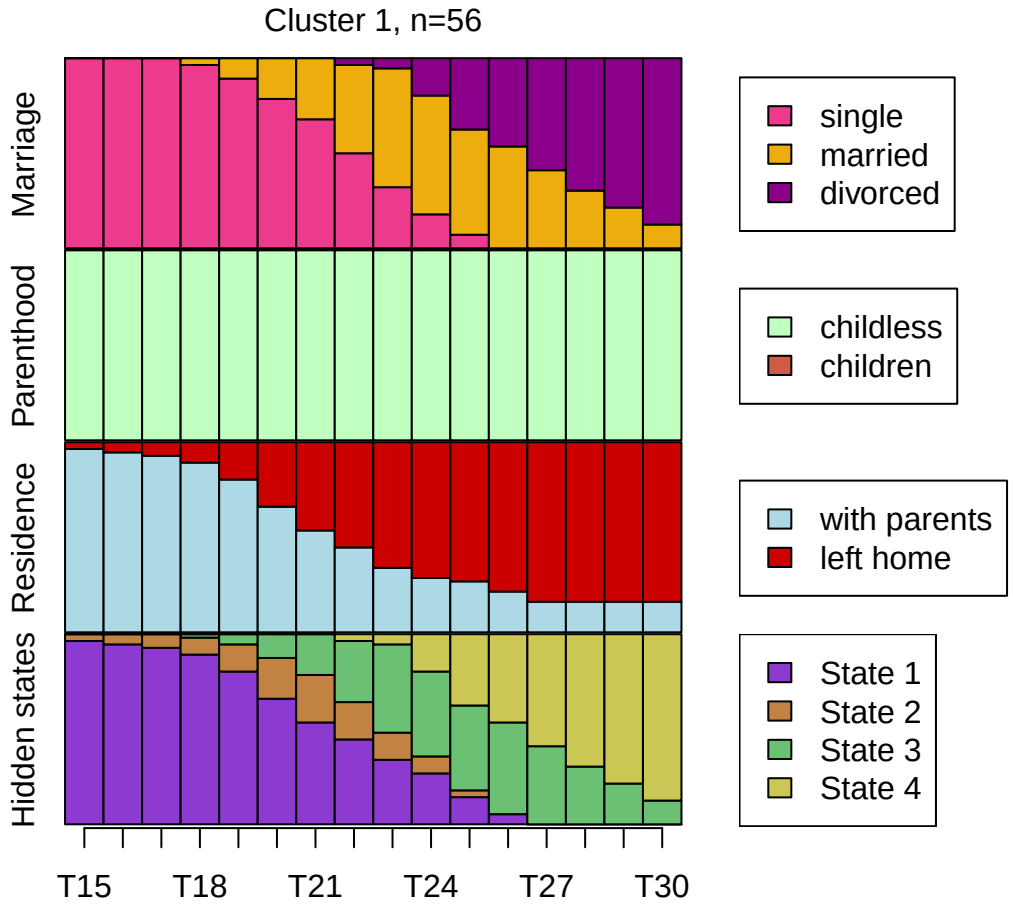
```
plot(
  fit_model$model, interactive = FALSE, which.plots = 2:3, nrow = 2,
  legend.prop = 0.35, cex.legend = 1.5, edge.label.cex = 1.5,
  ncol.legend = 2)
```





The equivalent to the `ssplot` function for MHMMs is called `mssplot`. It plots stacked sequence plots separately for each (chosen) cluster. If the user asks to plot more than one cluster, the function is interactive by default. If `ask = TRUE`, the cluster plots are chosen one by one by selecting their numbers from the menu; otherwise they are plotted in the order they appear in the `mhmm` object.

```
mssplot(fit_model$model, plots = "both", ask = FALSE, which.plots = 1)
```



Index