# depmix: An R-package for fitting mixture models on mixed multivariate data with Markov dependencies

Ingmar Visser[1]
Department of Psychology, University of Amsterdam
i.visser@uva.nl

November 21, 2007

[1]Correspondence concerning this manual should be adressed to: Ingmar Visser, Department of Psychology, University of Amsterdam, Roetersstraat 15, 1018 WB, Amsterdam, The Netherlands

**Abstract**

`depmix` implements a general class of mixture models with Markovian dependencies between them in the R programming language (R Development Core Team, 2007). This includes standard Markov models, latent/hidden Markov models, and latent class and finite mixture distribution models. The models can be fitted on mixed multivariate data with multinomial and/or gaussian distributions. Parameters can be estimated subject to general linear constraints. Parameter estimation is done through a direct optimization approach with gradients using the `Rdonlp2` optimization routine. A number of illustrative examples are included.

# Contents

# Chapter 1

# Introduction

Markov and latent Markov models are frequently used in the social sciences, in different areas and applications. In psychology, they are used for modelling learning processes, see Wickens (1982), for an overview, and Schmittmann et al. (2006) for a recent application. In economics, latent Markov models are commonly used as regime switching models, see e.g. Kim (1994) and Ghysels (1994). Further applications include speech recognition (Rabiner, 1989), EEG analysis (Rainer and Miller, 2000), and genetics (Krogh, 1998). In those latter areas of application, latent Markov models are usually referred to as hidden Markov models.

The `depmix` package was motivated by the fact that Markov models are used commonly in the social sciences, but no comprehensive package was available for fitting such models. Common programs for Markovian models include Panmark (Van de Pol et al., 1996), and for latent class models Latent Gold (Vermunt and Magidson, 2003). Those programs are lacking a number of important features, besides not being freely available. In particular, `depmix`: 1) handles multiple case, or multiple group, analyses; 2) handles arbitrarily long time series; 3) estimates models with general linear constraints between parameters; 4) analyzes mixed distributions, i.e., combinations of categorical and continuous observed variables; 5) fits mixtures of latent Markov models to deal with population heterogeneity; 6) can fit models with covariates. Although `depmix` is specifically meant for dealing with longitudinal or time series data, for say $T > 100$, it can also handle the limit case with $T = 1$. In those cases, there are no time dependencies between observed data, and the model reduces to a finite mixture model, or a latent class model. In the next chapter, an outline is provided of the model and the likelihood equations. In the chapters after that a number of examples are presented.

## Acknowledgements

I am indebted to many people for providing help in writing this package. First and foremost Maartje Raijmakers and Verena Schmittmann tested countless earlier versions, spotted bugs and suggested many features. Moreover, Maartje Raijmakers Raijmakers et al. (2001) provided the discrimination data set. Han van der Maas provided the speed-accuracy data Maas et al. (2005) and thereby neccessitated implementing models with time-dependent covariates. Conor Dolan and Raoul Grasman both provided valuable advice on statistics in general and optimization in particular.

# Chapter 2

# Dependent mixture models

The data considered here, has the general form $O_1^1, \ldots, O_1^m$, $O_2^1, \ldots, O_2^m$, $\ldots$, $O_T^1, \ldots, O_T^m$ for an $m$-variate time series of length $T$. As an example, consider a time series of responses generated by a single subject in a reaction time experiment. The data consists of three variables, reaction time, accuracy and a covariate which is a pay-off factor which determines the reward for speed and accuracy. These variables are measured at 168, 134 and 137 occasions respectively (in Figure 2.1 the first part of this series is plotted).

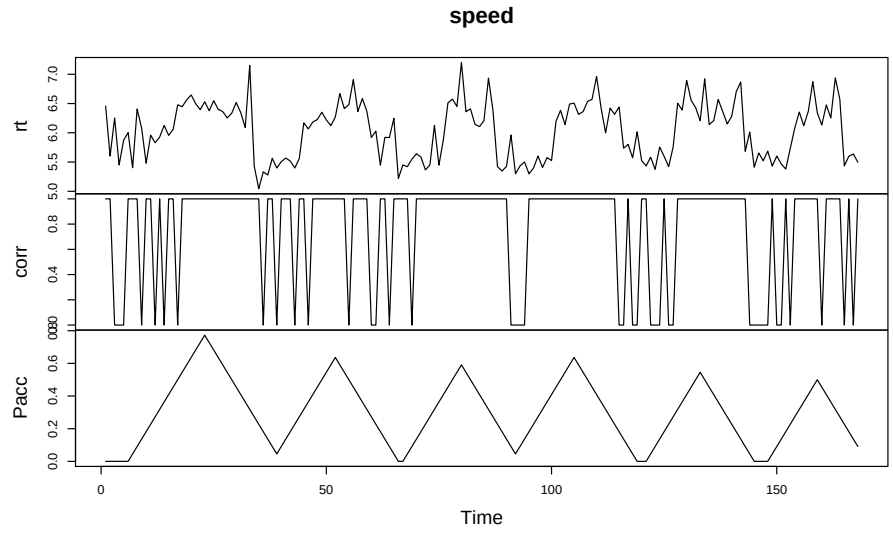


Figure 2.1: Reaction times, accuracy and pay-off values for the first series of responses in dataset `speed`.

The latent Markov model is commonly associated with data of this type, albeit usually only multinomial variables are considered. However, common estimation procedures, such as those implemented in Van de Pol et al. (1996) are not suitable for long time series due to underflow problems. In contrast, the

hidden Markov model is typically only used for 'long' univariate time series. In the next section, the likelihood and estimation procedure for the hidden Markov model is described, given data of the above form.

The dependent mixture model is defined by the following elements:

1. a set $\mathbf{S}$ of latent classes or states $S_i$, $i = 1, \ldots, n$,
2. a matrix $\mathbf{A}$ of transition probabilities $a_{ij}$ for the transition from state $S_i$ to state $S_j$,
3. a set $\mathbf{B}$ of observation functions $b_j(\cdot)$ that provide the conditional probabilities associated with latent state $S_j$,
4. a vector $\boldsymbol{\pi}$ of latent state initial probabilities $\pi_i$

When transitions are added to the latent class model, it is more appropriate to refer to the classes as states. The word class is rather more associated with a stable trait-like attribute whereas a state can change over time.

## 2.1 Likelihood

The loglikelihood of hidden Markov models is usually computed by the so-called forward-backward algorithm (Baum and Petrie, 1966; Rabiner, 1989), or rather by the forward part of this algorithm. Lystig and Hughes (2002) changed the forward algorithm in such a way as to allow computing the gradients of the loglikelihood at the same time. They start by rewriting the likelihood as follows (for ease of exposition the dependence on the model parameters is dropped here):

$$L_T = Pr(\mathbf{O}_1, \ldots, \mathbf{O}_T) = \prod_{t=1}^{T} Pr(\mathbf{O}_t | \mathbf{O}_1, \ldots, \mathbf{O}_{t-1}), \tag{2.1}$$

where $Pr(\mathbf{O}_1|\mathbf{O}_0) := Pr(\mathbf{O}_1)$. Note that for a simple, i.e. observed, Markov chain these probabilities reduce to $Pr(\mathbf{O}_t|\mathbf{O}_1, \ldots, \mathbf{O}_{t-1}) = Pr(\mathbf{O}_t|\mathbf{O}_{t-1})$. The log-likelihood can now be expressed as:

$$l_T = \sum_{t=1}^{T} \log[Pr(\mathbf{O}_t | \mathbf{O}_1, \ldots, \mathbf{O}_{t-1})]. \tag{2.2}$$

To compute the log-likelihood, Lystig and Hughes (2002) define the following (forward) recursion:

$$\phi_1(j) := Pr(\mathbf{O}_1, S_1 = j) = \pi_j b_j(\mathbf{O}_1) \tag{2.3}$$

$$\begin{aligned} \phi_t(j) &:= Pr(\mathbf{O}_t, S_t = j | \mathbf{O}_1, \ldots, \mathbf{O}_{t-1}) \\ &= \sum_{i=1}^{N} [\phi_{t-1}(i) a_{ij} b_j(\mathbf{O}_t)] \times (\Phi_{t-1})^{-1}, \end{aligned} \tag{2.4}$$

where $\Phi_t = \sum_{i=1}^{N} \phi_t(i)$. Combining $\Phi_t = Pr(\mathbf{O}_t|\mathbf{O}_1, \ldots, \mathbf{O}_{t-1})$, and equation (2.2) gives the following expression for the log-likelihood:

$$l_T = \sum_{t=1}^{T} \log \Phi_t. \tag{2.5}$$

The above forward recursion can readily be generalized to mixture models, in which it is assumed that the data are realizations of a number of different LMMs and the goal is to assign posterior probabilities to sequences of observations. This situation occurs, for example, in learning data where different learning strategies may lead to different answer patterns. From an observed sequence of responses, it may not be immediately clear from which learning process they stem. Hence, it is interesting to consider a mixture of latent Markov models which incorporate restrictions that are consistent with each of the learning strategies.

To compute the likelihood of a mixture of $K$ models, define the forward recursion variables as follows (these variables now have an extra index $k$ indicating that observation and transition probabilities are from latent model $k$):

$$\phi_1(j_k) = Pr(\mathbf{O}_1, S_1 = j_k) = p_k \pi_{j_k} b_{j_k}(\mathbf{O}_1). \tag{2.6}$$

$$\begin{aligned} \phi_t(j_k) &= Pr(\mathbf{O}_t, S_t = j_k | \mathbf{O}_1, \ldots, \mathbf{O}_{t-1}) \\ &= \left[ \sum_{k=1}^{K} \sum_{i=1}^{n_k} \phi_{t-1}(i_k) a_{ij_k} b_{j_k}(\mathbf{O}_t) \right] \times (\Phi_{t-1})^{-1}, \end{aligned} \tag{2.7}$$

where $\Phi_t = \sum_{k=1}^{K} \sum_{i=1}^{n_k} \phi_t(j_k)$. Note that the double sum over $k$ and $n_k$ is simply an enumeration of all the states of the model. Now, because $a_{ij_k} = 0$ whenever $S_i$ is not part of component $k$, the sum over $k$ can be dropped and hence equation 2.7 reduces to:

$$\phi_t(j_k) = \left[ \sum_{i=1}^{n_k} \phi_{t-1}(i_k) a_{ij_k} b_{j_k}(\mathbf{O}_t) \right] \times (\Phi_{t-1})^{-1} \tag{2.8}$$

The loglikelihood is computed by applying equation 2.5 on these terms. For multiple cases, the log-likelihood is simply the sum over the individual log-likelihoods.

**Computational considerations** From equations (2.3–2.4), it can be seen that computing the forward variables, and hence the log-likelihood, takes $O(Tn^2)$ computations, for an $n$-state model and a time series of length $T$. Consider a mixture of two components, one with two states and the other with three states. Using equations (2.3–2.4) to compute the log-likelihood of this model one needs $O(Tn^2) = O(T \times 25)$ computations whereas with the mixture equations (2.6–2.7), $\sum_{n_i} O(n_i^2 T)$ computations are needed, in this case $O(T \times 13)$. So, it can be seen that in this easy example the computational cost is almost halved.

## 2.2 Gradients

See equations 10–12 in Lystig and Hughes (2002) for the score recursion functions of the hidden Markov model for a univariate time series. Here the corresponding score recursion for the multivariate mixture case are provided. The $t = 1$ components of this score recursion are defined as (for an arbitrary param-

eter $\lambda_1$):

$$\psi_1(j_k;\lambda_1) := \frac{\partial}{\partial\lambda_1} Pr(\mathbf{O}_1|S_1 = j_k) \tag{2.9}$$

$$\begin{aligned} &= \left[\frac{\partial}{\partial\lambda_1} p_k\right] \pi_{j_k} b_{j_k}(\mathbf{O}_1) + p_k \left[\frac{\partial}{\partial\lambda_1}\pi_{j_k}\right] b_{j_k}(\mathbf{O}_1) \\ &\quad + p_k \pi_{j_k} \left[\frac{\partial}{\partial\lambda_1} b_{j_k}(\mathbf{O}_1)\right], \end{aligned} \tag{2.10}$$

and for $t > 1$ the definition is:

$$\psi_t(j_k;\lambda_1) = \frac{\frac{\partial}{\partial\lambda_1} Pr(\mathbf{O}_1,\ldots,\mathbf{O}_t, S_t = j_k)}{Pr(\mathbf{O}_1,\ldots,\mathbf{O}_{t-1})} \tag{2.11}$$

$$\begin{aligned} &= \sum_{i=1}^{n_k} \Bigg\{ \psi_{t-1}(i;\lambda_1) a_{ij_k} b_{j_k}(\mathbf{O}_t) \\ &\quad + \phi_{t-1}(i) \left[\frac{\partial}{\partial\lambda_1} a_{ij_k}\right] b_{j_k}(\mathbf{O}_t) \\ &\quad + \phi_{t-1}(i) a_{ij_k} \left[\frac{\partial}{\partial\lambda_1} b_{j_k}(\mathbf{O}_t)\right] \Bigg\} \times (\Phi_{t-1})^{-1}. \end{aligned} \tag{2.12}$$

Using above equations, Lystig and Hughes (2002) derive the following equation for the partial derivative of the likelihood:

$$\frac{\partial}{\partial\lambda_1} l_T = \frac{\boldsymbol{\Psi}_T(\lambda_1)}{\boldsymbol{\Phi}_T}, \tag{2.13}$$

where $\Psi_t = \sum_{k=1}^{K}\sum_{i=1}^{n_k} \psi_t(j_k;\lambda_1)$. Starting from the equation from the logarithm of the likelihood, this is easily seen to be correct:

$$\begin{aligned} \frac{\partial}{\partial\lambda_1} \log Pr(\mathbf{O}_1,\ldots,\mathbf{O}_T) &= Pr(\mathbf{O}_1,\ldots,\mathbf{O}_T)^{-1} \frac{\partial}{\partial\lambda_1} Pr(\mathbf{O}_1,\ldots,\mathbf{O}_T) \\ &= \frac{Pr(\mathbf{O}_1,\ldots,\mathbf{O}_{T-1})}{Pr(\mathbf{O}_1,\ldots,\mathbf{O}_T)} \Psi_T(\lambda_1) \\ &= \frac{\boldsymbol{\Psi}_T(\lambda_1)}{\boldsymbol{\Phi}_T}. \end{aligned}$$

Further, to actually compute the gradients, the partial derivatives of the parameters and observation distribution functions are neccessary, i.e., $\frac{\partial}{\partial\lambda_1} p_k$, $\frac{\partial}{\partial\lambda_1}\pi_i$, $\frac{\partial}{\partial\lambda_1} a_{ij}$, and $\frac{\partial}{\partial\lambda_1}\mathbf{b}_i(\mathbf{O}_t)$. Only the latter case requires some attention. We need the following derivatives $\frac{\partial}{\partial\lambda_1}\mathbf{b}_j(\mathbf{O}_t) = \frac{\partial}{\partial\lambda_1}\mathbf{b}_j(O_t^1,\ldots,O_t^m)$, for arbitrary parameters $\lambda_1$. To stress that $\mathbf{b}_j$ is a vector of functions, we here used boldface. First note that because of local independence we can write:

$$\frac{\partial}{\partial\lambda_1}\left[b_j(O_t^1,\ldots,O_t^m)\right] = \frac{\partial}{\partial\lambda_1}\left[b_j(O_t^1)\right] \times \left[b_j(O_t^2)\right],\ldots,\left[b_j(O_t^m)\right].$$

Applying the chain rule for products we get:

$$\frac{\partial}{\partial\lambda_1}[b_j(O_t^1,\ldots,O_t^m)] = \sum_{l=1}^{m} \left[\prod_{i=1,\ldots,\hat{l},\ldots,m} b_j(O_t^i)\right] \times \frac{\partial}{\partial\lambda_1}[b_j(O_t^l)], \tag{2.14}$$

where $\hat{l}$ means that that term is left out of the product. These latter terms, $\frac{\partial}{\partial \lambda_1}[b_j(O_t^k)]$, are easy to compute given either multinomial or gaussian observation densities $b_j(\dot{)}$

## 2.3 Parameter estimation

Parameters are estimated in `depmix` using a direct optimization approach instead of the EM algorithm which is frequently used for this type of model. The EM algorithm however has some drawbacks. First, it can be slow to converge. Second, applying constraints to parameters can be problmatic. Third, the EM algorithm can sometimes lead to incorrect estimates when constraints are applied to parameters in the M-step of the algorithm.

Optimization in `depmix` is done using `Rdonlp2` Tamura (ry.tamura@gmail.com); Spellucci (2002) (and there is support for using `NPSOL`). These packages are preferred because they can deal with both general linear and non-linear constraints between parameters.

# Chapter 3

# Using depmix

Three steps are involved in using `depmix` which are illustrated below with examples:

1. data specification with function `markovdata`
2. model specification with function `dmm`
3. model fitting with function `fitdmm`

To be able to fit models, data need to in a specific format created for this package. Basically, data should be in the form of a matrix with each row corresponding to measures taken at a single measurement occasion for a single subject. The function `markovdata` further only requires one argument providing the itemtypes, being one of categorical, continuous or covariate. A `markovdata` object is a matrix with a number of attributes.

## 3.1 Creating data sets

As an example, a dataset is created in below code with two variables measured at two times 50 occasions.

```
x=rnorm(100,10,2)
y=ifelse(runif(100)<0.5,0,1)
z=matrix(c(x,y),100,2)
md=markovdata(z,itemtypes=c("cont","cat"),ntimes=c(50,50))
md[1:10,]
```

In the example below, we split the dataset `speed` into three separate datasets, which we later use as an example to do multi-group analysis.

```
data(speed)
r1=markovdata(dat=speed[1:168,],itemt=itemtypes(speed))
r2=markovdata(dat=speed[169:302,],itemt=itemtypes(speed))
r3=markovdata(dat=speed[303:439,],itemt=itemtypes(speed))
```

### 3.1.1 Example data: speed

Throughout this manual a data set called `speed` is used. It consists of three time series with three variables: reaction time, accuracy, and a covariate Pacc which defines the relative pay-off for speeded and accurate responding. The participant in this experiment switches between fast responding at chance level and relatively slower responding at a high level of accuracy.

Interesting hypotheses to test are: is the switching regime symmetric? Is there evidence for two states or does one state suffice? Is the guessing state actually a guessing state, i.e., is the probability correct at chance level of 0.5?

## 3.2 Defining models

A dependent mixture model is defined by the number of states, and by the item distribution functions, and can be created with the `dmm`-function as follows:

```
mod <-dmm(nstates=2,itemtypes=c("gaus",2))
```

Here `itemtypes` is a vector of length the number of items measured at each occasion specifying the desired distributions, in this case the first item is to follow a normal distribution, and the second item follows a bernouilli distribution. Allowable distributions are multinomial and gaussian. Multinomial items are specified by the number of categories, i.e. 2 or higher, whereas gaussian items are specified by either using "gaussian" or 1.

The function `dmm` returns an object of class `dmm` which has its own summary function providing the parameter values of the model. See the help files for further details. Except in simple cases, starting values can be a problem in latent Markov models, and so in general it's best to provide them if you have a fairly good idea of what to expect. Providing starting values is done through the stval argument:

```
st <- c(1,0.9,0.1,0.2,0.8,2,1,0.7,0.3,5,2,0.2,0.8,0.5,0.5)
mod <- dmm(nsta=2,itemt=c(1,2), stval=st)
```

### 3.2.1 Generating data

The `dmm`-class has a `generate` method that can be used to generate data according to a specified model.

```
gen<-generate(c(100,50),mod)
```

## 3.3 Fitting models

Fitting models is done using the function `fitdmm`. The standard call only requires a dataset and a model as in:

```
data(speed)
mod <- dmm(nstates=2,itemtypes=c(1,2))
fitex <- fitdmm(speed,mod)
```

Calling `fitdmm` produces some online ouput about the progress of the optimization which can be controlled with the `printlevel` argument. Its default value of 1 just prints the log-likelihood at each iteration of the optimization. Printlevels starting from 15 and higher produce increasingly annoying output from the C-routines that compute the log-likelihood.

`Fitdmm` returns an object of class `fit` which has a summary method showing the estimated parameter values, along with standard errors, and t-ratios whenever those are available. Along with the log-likelihood, the AIC and BIC values are provided. Apart from the printed values (see summary below), a `fit`-object has a number of other fields. Most importantly, it contains a copy of the fitted model in `mod` and it has a field `post` containing posterior state estimates. That is, for each group $g$, `post$states[[g]]` is a matrix with dimensions the number of states of the model + 2, and the sum of the lengths of the time series as rows. The first column contains the a posteriori component model (see the section on mixtures of latent Markov models), the second column has the state number within the component, and the other columns are used for the a posteriori probabilities of each of the states.

# Chapter 4

# Extending and constraining models

## 4.1 Fixing and constraining parameters

Continuing the example from above, it can be seen that in one of the states, the probability of a correct answer is about .5, as is the probability of an incorrect answer, i.e., these are parameters Item2,p1 and Item2,p2. This latent state, is supposed to be a guessing state, and hence it makes sense to constrain these parameters to their theoretical values of .5. Similarly, the initial state probability for the slow state is one, and zero for the other state, and hence it makes sense to fix these parameters. The third constraint that we consider here is an equality constraint between the transition parameters. Using this constraint, we can test the hypothesis whether the switching between states is a symmetric process or not. Hence, we constrain the transition parameters $a_{11}$ and $a_{22}$.

Constraining and fixing parameters is done in a similar fashion as the `pa` command that is used in LISREL (Jöreskog and Sörbom, 1999). The `conpat` argument to the `fitdmm`-function specifies for each parameter in the model whether it's fixed (0) or free (1 or higher). Equality constraints can be imposed by having two parameters have the same number in the `conpat` vector. When only fixed values are required the `fixed` argument can be used instead of `conpat`, with zeroes for fixed parameters and other values (ones e.g.) for non-fixed parameters. Fitting the models subject to these constraints is handled by the optimization routine `donlp2`.

**Parameter numbering** When using the `conpat` and `fixed` arguments, complete vectors should be supplied, i.e., these vectors should have length of the number of parameters of the model. Parameters are numbered in the following order:

1. the mixing proportions of a mixture of latent Markov models, i.e., just one parameter for a single component model which has value 1 and is fixed
2. the component parameters for each component consisting of the following:
   (a) transition parameters in row major order, $a_{11}, a_{12}, a_{13}, \ldots, a_{21}, a_{22}, a_{23}, \ldots$

(b) the observation parameters per state and per item; the per item parameters for multinomials are just the probabilities for each category; the per item parameters for the gaussian are $\mu$ first and then $\sigma$

(c) the initial state probabilities per state

```
conpat=rep(1,15)
conpat[1]=0
conpat[14:15]=0
conpat[8:9]=0
conpat[2]=conpat[5]=2
stv=c(1,.896,.104,.084,.916,5.52,.20,.5,.5,6.39,.24,.098,.902,0,1)
mod=dmm(nstates=2,itemt=c("n",2),stval=stv,conpat=conpat)
```

In the example above `conpat` is used to specify a number of constraints. First, `conpat[1]=0` specifies that the mixing proportion of the model should be fixed (at its starting value of 1), which is always the case for single component models. Second, `conpat[14:15]=0` fixes the initial state probabilities to zero and one respectively. Similarly, for `conpat[8:9]=0`, which are the guessing state parameters for the accuracy scores. They are both fixed at 0.5 so as to make the guessing state an actual guessing state. Finally, by invoking `conpat[2]=conpat[5]=2`, transition parameters $a_{11}$ and $a_{22}$ are set to be equal. Whenever equality constraints are not sufficient, general linear constraints can be specified using the `conrows` argument.

## 4.2 Multi group/case analysis

```
conpat=rep(1,15)
conpat[1]=0
conpat[8:9]=0
conpat[14:15]=0
stv=c(1,0.9,0.1,0.1,0.9,5.5,0.2,0.5,0.5,6.4,0.25,0.9,0.1,0.5,0.5)
mod<-dmm(nstates=2,itemt=c("n",2),stval=stv,conpat=conpat)
```

`depmix` can handle multiple cases or multiple groups. A multigroup model is specified using the function `mgdmm` as follows:

```
mgr <- mgdmm(dmm=mod,ng=3,trans=TRUE,obser=FALSE)
mgrfree <- mgdmm(dmm=mod,ng=3,trans=FALSE)
```

The `ng` argument specifies the number of groups, and the `dmm` argument specifies the model for each group. `dmm` can be either a single model or list of models oflength `ng`. If it is a single model, each group has an identical structural model (same fixed and constrained parameters). Three further arguments can be used to constrain parameters between groups, `trans`, `obser`, and `init` respectively. By setting either of these to `TRUE`, the corresponding transition, observation, and initial state parameters are estimated equal between groups[1].

[1]There is at this moment no way of fine-tuning this to restrict equalities to individual parameters. However, this can be accomplished by manually changing the linear constraint matrix, and the corresponding upper and lower boundaries.

In this example, the model from above was used and fitted on the three observed series, and the `trans=TRUE` ensures that the transition matrix parameters are constrained to be equal between the models for these series, whereas the observation parameters are freely estimated, i.e. to capture learning effects.

The loglikelihood ratio statistic can be used to test whether constraining these transition parameters significantly reduces the goodness-of-fit of the model. The statistic has an approximate $\chi^2$ distribution with $df = 4$ because in each but the first model, two transition matrix parameters were estimated equal to the parameters in the first model (note that the other two transition parameters already had to be constrained to ensure that the rows of the transition matrices sum to 1).

## 4.3 Models with time-dependent covariates

Specifying a model with covariates is done by including two arguments in a call to `dmm` called `tdfix` and `tdst`, where `td` means time dependent. `tdfix` is a logical vector of length the number of parameters of the model, specifying which parameters are to be estimated time-dependent. For an arbitrary parameter $\lambda$, the model that is estimated has the form:

$$\lambda_t = \lambda_0 + \beta x_t, \tag{4.1}$$

where $\lambda_0$ is the intercept of the parameter, $\beta$ is the regression coefficient, and $x_t$ is the time-dependent covariate. The covariate has to be scaled to lie between 0 and 1; this is neccessary to be able to impose the right constraints on $\beta$ in order to ensure that $\lambda_t$ is always appropriate, ie within its lower and upper bounds (mostly 0 and 1 for multinomial item parameters and transition parameters etc). The current version of `depmix` does not have non-time-dependent covariates, which can simply be faked by having $x_t$ be constant, and there is only support for a single covariate.

In the example below, the transition parameters (numbers 2–5) are defined to depend on the covariate which is the pay-off for accuracy. Providing starting values for the covariates is optional. If not provided they are chosen at random around 0 which usually works just fine.

```
conpat=rep(1,15)
conpat[1]=0
conpat[8:9]=0
conpat[14:15]=0
conpat[2]=2
conpat[5]=2
stv=c(1,0.9,0.1,0.1,0.9,5.5,0.2,0.5,0.5,6.4,0.25,0.9,0.1,0,1)
tdfix=rep(0,15)
tdfix[2:5]=1
tdst=rep(0,15)
tdst[2:5]=c(-0.4,0.4,0.15,-0.15)

mod<-dmm(nstates=2,itemt=c("n",2),stval=stv,conpat=conpat,
tdfix=tdfix,tdst=tdst,modname="twoboth+cov")
```

# Chapter 5

# Special topics

## 5.1 Starting values

Although providing your own starting values is preferable, `depmix` has a routine for generating starting values using the `kmeans`-function from the `stats`-package. This will usually provide reasonable starting values, but can be way off in a number of cases. First, for univariate categorical time series, `kmeans` does not work at all, and `depmix` will provide a warning. Second, for multivariate series with unordered categorical items with more than 2 categories, `kmeans` may provide good starting values, but they may similarly be completely off, due to the implicit assumption in `kmeans` that the categories are indicating an underlying continuum. Starting values using `kmeans` are automatically provided when a model is specified without starting values. The argument `kmst` to the `fitdmm`-function can be used to control this behavior.

Starting values of the parameters, either user provided or generated, can be further boosted by using posterior estimates using the Viterbi algorithm Rabiner (1989). That is, first the a posteriori latent states are generated from the current parameter values for the data at hand. Next, from the a posteriori latent states, new parameter estimates are derived. This is done by default and can be controlled by the `poststst` argument. Provided that the starting values were close to their true values, using this procedure further pushes those parameters in the right direction. If however the original values were bad, this procedure may result in bad estimates, i.e., optimization will lead to some non-optimal local maximum of the loglikelihood.

## 5.2 Finite mixtures and latent class models

The function `lca` can be used to specify latent class models and/or finite mixture models. It is simply a wrapper for the `dmm` function, and all it does is adding appropriate numbers of zeroes and ones to the parameter specification vectors for starting values, fixed values and linear constraints. When a model has class `lca` the summary function does not print the transition matrix (because it is fixed and/or not estimated).

## 5.3 Mixtures of latent Markov models

`depmix` provides support for fitting mixtures of latent Markov models using the `mixdmm` function; it takes a list of `dmm`'s as argument, possibly together with the starting values for the mixing proportions for each component model. There's an example in the helpfiles. It fits the model to data from a discrimination learning experiment which is provided as data set `discrimination` Raijmakers et al. (2001).

# Bibliography


L. E. Baum and T. Petrie. Statistical inference for probabilistic functions of finite state Markov chains. *Annals of Mathematical Statistics*, 67:1554–40, 1966.

Eric Ghysels. On the periodic structure of the business cycle. *Journal of Business and Economic Statistics*, 12(3):289–298, 1994.

K.G. Jöreskog and D. Sörbom. *LISREL 8 [Computer program].* Scientific Software International., Chicago, 1999.

Chang-Jin Kim. Dynamic linear models with Markov-switching. *Journal of Econometrics*, 60:1–22, 1994.

Anders Krogh. An introduction to hidden Markov models for biological sequences. In S. L. Salzberg, D. B. Searls, and S. Kasif, editors, *Computational methods in molecular biology*, chapter 4, pages 45–63. Elsevier, Amsterdam, 1998.

Theodore C. Lystig and James P. Hughes. Exact computation of the observed information matrix for hidden markov models. *Journal of Computational and Graphical Statistics*, 2002.

Han L. J. van der Maas, Conor V. Dolan, and Peter C. M. Molenaar. Phase transitions in the trade-off between speed and accuracy in choice reaction time tasks. *Manuscript in revision*, 2005.

A. L. McCutcheon. *Latent class analysis.* Number 07-064 in Sage University Paper series on Quantitative Applications in the Social Sciences. Beverly Hills: Sage Publications, 1987.

R Development Core Team. *R: A Language and Environment for Statistical Computing.* R Foundation for Statistical Computing, Vienna, Austria, 2007. URL `http://www.R-project.org`. ISBN 3-900051-07-0.

Lawrence R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of IEEE*, 77(2):267–295, 1989.

Maartje E. J. Raijmakers, Conor V. Dolan, and Peter C. M. Molenaar. Finite mixture distribution models of simple discrimination learning. *Memory & Cognition*, 29(5):659–677, 2001.

Gregor Rainer and Earl K. Miller. Neural ensemble states in prefrontal cortex identified using a hidden Markov model with a modified em algorithm. *Neurocomputing*, 32–33:961–966, 2000.

Verena D. Schmittmann, Ingmar Visser, and Maartje E. J. Raijmakers. Multiple learning modes in the development of rule-based category-learning task performance. *Neuropsychologia*, 44(11):2079–2091, 2006.

Peter Spellucci. Donlp2. 2002. URL `http://www.netlib.org/ampl/solvers/donlp2/`.

Ryuichi Tamura(ry.tamura@gmail.com). *Rdonlp2: an R extension library to use Peter Spelluci's DONLP2 from R.*, 2007. URL `http://arumat.net/Rdonlp2/`. R package version 0.3-1.

Frank Van de Pol, Rolf Langeheine, and W. De Jong. *PANMARK 3. Panel analysis using Markov chains. A latent class analysis program [User manual].* Voorburg: The Netherlands, 1996.

Jeroen K. Vermunt and Jay Magidson. *Latent Gold 3.0 [Computer program and User's Guide].* Belmont (MA), USA, 2003.

Thomas D. Wickens. *Models for Behavior: Stochastic processes in psychology.* W. H. Freeman and Company, San Francisco, 1982.