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

Version: 0.9.1

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

July 12, 2006

[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, 2006). 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 from a number of distributions including the binomial, multinomial, gaussian, lognormal and Weibull distributions. Parameters can be estimated subject to general linear constraints, and with optional inclusion of regression on (time-dependent) covariates. Parameter estimation is done through a direct optimization approach with gradients using the `nlm` optimization routine. Optional support for using the NPSOL optimization routines is provided as well. 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. (2005) 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, analysis; 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

Development of this pacakge was supported by European Commission grant 51652 (NEST) and by a VENI grant from the Dutch Organization for Scientific Research (NWO) to Ingmar Visser.

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 provided the discrimination data set. Han van der Maas provided the speed-accuracy data 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. Below, a summary is provided for these data, as well as a plot of the first timeseries, which is selected by `nind=1`.

```
> data(speed)

name=speed:        file= .../speed.rda::        found

> summary(speed)

Data set:                  speed
 nr of items:              3
 item type(s):             continuous categorical covariate
 nr of covariates:         1
 item name(s):             rt corr Pacc
 length(s) of series:      168 134 137
 nr of independent series: 3
 data:                     6.45677 5.602119 6.253829 5.451038 5.872118 6.003887  ...
```

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 $S_j$,

```
> plot(speed, nind = 1)
```

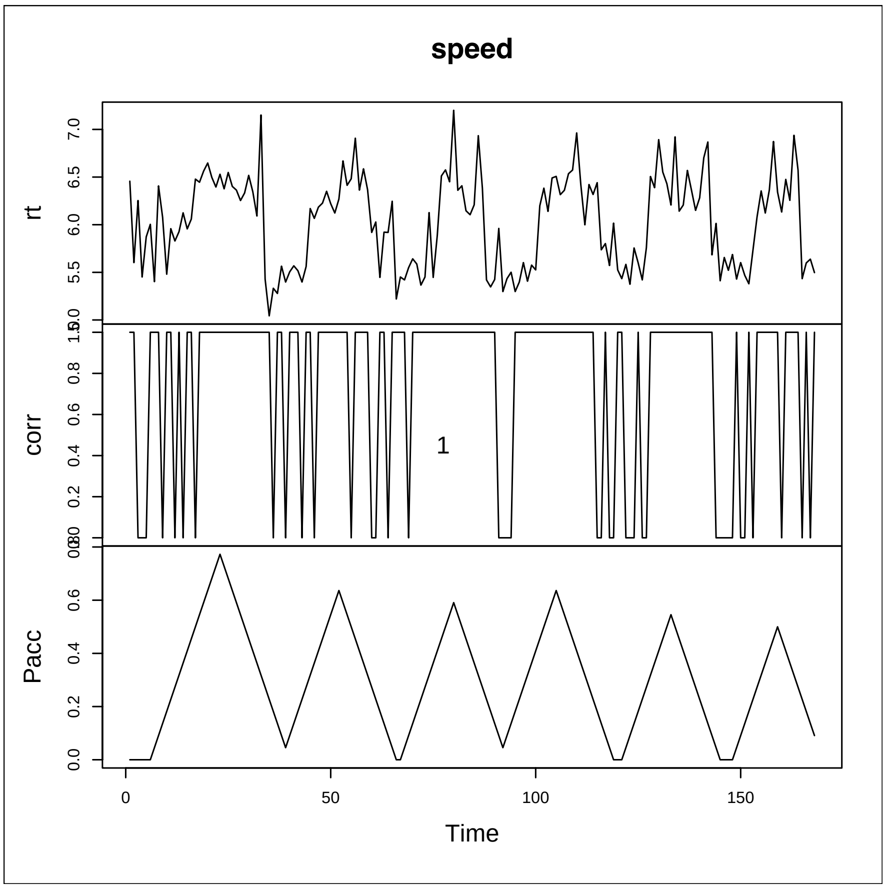

Figure 2.1: Reaction times, accuracy and pay-off values for the first series of responses in dataset `speed`.

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 parameter $\lambda_1$):

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

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

and for $t > 1$ the definition is:

$$\begin{aligned}
\psi_t(j_k;\lambda_1) &= \frac{\frac{\partial}{\partial\lambda_1} Pr(\mathbf{O}_1,\ldots,\mathbf{O}_t, S_t = j_k)}{Pr(\mathbf{O}_1,\ldots,\mathbf{O}_{t-1})} && (2.11)\\
&= \sum_{i=1}^{n_k} \Bigg\{ \psi_{t-1}(i;\lambda_1) a_{ij_k} b_{j_k}(\mathbf{O}_t) \\
&\quad + \phi_{t-1}(i) \left[\frac{\partial}{\partial\lambda_1} a_{ij_k}\right] b_{j_k}(\mathbf{O}_t) && (2.12)\\
&\quad + \phi_{t-1}(i) a_{ij_k} \left[\frac{\partial}{\partial\lambda_1} b_{j_k}(\mathbf{O}_t)\right] \Bigg\} \times (\Phi_{t-1})^{-1}.
\end{aligned}$$

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

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

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:

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

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:

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

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 $b_j()$

## 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. The EM algorithm can sometimes lead to incorrect estimates when constraints are applied to parameters in the M-step of the algorithm. The package was designed to be used with the `npsol`-library, the main reason being that it handles general linear (in-)equality constraints very well. Unfortunately, `npsol` is not freeware and hence is not distributed with **depmix**. Two other options are available for optimization using `nlm` and `optim` respectively. Linear equality constraints are fitted through reparametrization. Inequality constraints are fitted through adding a penalty to the likelihood depending on the amount by which a constraint is not satisfied. The argument `vfactor` to the fitting function can be used to control this bahavior. See details of this in the chapter on fitting models.

# 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 we make a dataset 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, ]
```

```
      continuous categorical
 [1,]  11.642114           0
 [2,]   7.996006           1
 [3,]  10.481705           0
 [4,]  10.440898           0
 [5,]   8.454049           1
 [6,]  11.578053           0
 [7,]   8.875631           1
 [8,]  10.135414           0
 [9,]  10.783023           1
[10,]  10.400851           0
```

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)

name=speed:         file= .../speed.rda::         found

> 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))
> summary(r2)

Data set:                    3-item data
 nr of items:                3
 item type(s):               continuous categorical covariate
 nr of covariates:           1
 item name(s):               rt corr Pacc
 length(s) of series:        134
 data:                       6.621406 5.332719 5.463832 5.361292 5.398163 5.384495  ...
```

Here is the full specification of the `markovdata` function.

---

`markovdata`                *Specifying Markov data objects*

---

### Description

Markovdata creates an object of class `md`, to be used by `fitdmm`.

### Usage

```
markovdata(dat, itemtypes, nitems = length(itemtypes), ntimes =
         length(as.matrix(dat))/nitems, replicates = rep(1,
         length(ntimes)), inames = NULL, dname = NULL, xm =
         NA)

## S3 method for class 'md':
summary(object, ...)
## S3 method for class 'md':
plot(x, nitems = 1:(min(5, dim(x)[2])),
                nind = 1:(min(5,length(attributes(x)$ntimes))),...)
## S3 method for class 'md':
print(x, ...)

dname(object)
ntimes(object)
itemtypes(object)
replicates(object)
```

```
ncov(object)
inames(object)
nitems(object)
ind(object)
```

## Arguments

| | |
|---|---|
| `dat` | An R object to be coerced to markovdata, a data frame or matrix. |
| `itemtypes` | A vector providing the types of measurement with possible values 'continuous', 'categorical', and 'covariate'. This is mainly only used to rearrange the data when there are covariates in such a way that the covariate is in the last column. Only one covariate is supported in estimation of models. |
| `ntimes` | The number of repeated measurements, ie the length of the time series (this may be a vector containing the lengths of independent realiazations). It defaults the number of rows of the data frame or data matrix. |
| `replicates` | Using this argument case weights can be provided. This is particularly usefull in eg latent class analysis with categorical variables when there usually are huge numbers of replicates, ie identical response patterns. `depmix` computes the raw data log likelihood for each case separately. Thus, when there are many replicates of a case a lot of computation time is saved by specifying case weights instead of providing the full data set. |
| `inames` | The names of items. These default to the column names of matrices or dataframes. |
| `dname` | The name of the dataset, used in summary, print and plot functions. |
| `xm` | `xm` is the missing data code. It can be any value but zero. Missing data are recoded into `NA`. |
| `object,x` | An object of class `md`. |
| `...` | Further arguments passed on to plot and summary. |
| `nitems,nind` | In the plot function, these arguments control which data are to be plotted, ie nitems indicates a range of items, and nind a range of realizations, respectively. |

## Details

The function `markovdata` coerces a given data frame or matrix to be an object of class `md` such that it can be used in `fithmm`. The `md` object has its own summary, print and plot methods.

The functions dname, itemtypes, ntimes, and replicates retrieve the respective attributes with these names; similarly `ncov, nitems, inames`, and `ind` retrieve the number of covariates, the number of items (the number of columns of the data), the column names and the number of `ind`ependent realizations respectively.

**Value**

An md-object is a matrix of dimensions sum(ntimes) by nitems, containing the measured variables and covariates rearranged such that the covariate appears in the last column. The column names are `inames` and the matrix has three further attributes:

| | |
|---|---|
| `dname` | The name of the data set. |
| `itemtypes` | See above. |
| `ntimes` | See above. This will be a vector computed as ntimes=rep(ntimes,nreal). |
| `replicates` | The number of replications of each case, used as weigths in computing the log likelihood. |

**Author(s)**

Ingmar Visser ⟨i.visser@uva.nl⟩

**See Also**

`dmm`, `depmix`

**Examples**

```
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"))
summary(md)

data(speed)
summary(speed)
plot(speed,nind=2)

# split the data into three data sets
# (to perform multi group analysis)
r1=markovdata(dat=speed[1:168,],item=itemtypes(speed))
r2=markovdata(dat=speed[169:302,],item=itemtypes(speed))
r3=markovdata(dat=speed[303:439,],item=itemtypes(speed))
summary(r2)
```

## 3.2 Data set speed

Throughout this manual we will use a data set called speed, and hence we provide some background information on how these data were gathered.

`speed` *Speed Accuracy Switching Data*

**Description**

This data set is a bivariate series of reaction times and accuracy scores of a single subject switching between slow and accurate responding and fast guessing on a lexical decision task. The slow and accurate responding, and the fast guessing can be modelled using two states, with a switching regime between them. The dataset further contains a third variable called Pacc, representing the relative pay-off for accurate responding, which is on a scale of zero to one. The value of Pacc was varied during the experiment to induce the switching. This data set is a subset of data from experiment 2 in *Van der Maas et al, 2005*.

**Usage**

```
data(speed)
```

**Format**

An object of class `markovdata`.

**Source**

Han L. J. Van der Maas, Conor V. Dolan and Peter C. M. Molenaar (2005), Phase Transitions in the Trade-Off between Speed and Accuracy in Choice Reaction Time Tasks. *Manuscript in revision.*

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?

## 3.3 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))
> summary(mod)

 Model:  2 -state model
 Number of parameters:  15
 Free parameters:       9
 Number of states:      2
 Number of items:       2
 Item types:            gaussian 2

 Parameter values, transition matrix
```

| distribution | code | parameters |
|---|---|---|
| multinomial | $2, 3, 4, \ldots$ | $p_1, p_2, p_3, \ldots$ |
| gaussian, normal | 1 | $\mu, \sigma$ |
| lognormal | -21 | $l\mu$ , $l\sigma$ |
| weibull | -22 | shape (a), scale (b) |
| gamma | -23 | shape (a), scale (s) |
| 3lognormal | -31 | $l\mu$ , $l\sigma$, shift |
| 3weibull | -32 | shape (a), scale (b), shift |
| 3gamma | -33 | shape (a), scale (s), shift |

Table 3.1: Allowable distribution names, internal codes, and number of parameters.

```
       State1 State2
State1  0.081  0.919
State2  0.438  0.562


 Parameter values, observation parameters

       Item1,mean Item1,stddev Item2,p 1 Item2,p 2
State1     11.321        1.594     0.055     0.945
State2      9.952        1.958     0.557     0.443


 Parameter values, initial state probabilies

    State1 State2
val  0.421  0.579
```

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 listed in Table 3.1, along with their internal code, and the parameter names. The R-internal code is used for estimating these parameters. Specifics of these distributions and their estimation can be found in their respective help files. Itemtypes can be specified by their name or by their internal code, except in the case of multinomial items, which have to be specified by a number.

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)
> summary(mod)
```

```
 Model:  2 -state model
 Number of parameters:  15
 Free parameters:       9
 Number of states:      2
 Number of items:       2
 Item types:            1 2

 Parameter values, transition matrix

       State1 State2
State1    0.9    0.1
State2    0.2    0.8


 Parameter values, observation parameters

       Item1,mean Item1,stddev Item2,p 1 Item2,p 2
State1          2            1       0.7       0.3
State2          5            2       0.2       0.8


 Parameter values, initial state probabilies

    State1 State2
val    0.5    0.5
```

---

## `dmm` *Dependent Mixture Model Specification*

---

### Description

dmm `dmm` creates an object of class `dmm`, a dependent mixture model.

lca `lca` creates an object of class `dmm,lca`, a latent class model or an independent mixture model.

### Usage

```
dmm(nstates, itemtypes, modname = NULL, fixed = NULL,
         stval = NULL, conrows = NULL, conpat = NULL, tdfix =
         NULL, tdst = NULL, linmat = NULL, snames = NULL,
         inames = NULL)
## S3 method for class 'dmm':
summary(object, specs=FALSE, precision=3, se=NULL, ...)

lca(nclasses, itemtypes, modname = NULL, fixed = NULL,
          stval = NULL, conrows = NULL, conpat = NULL,
                      linmat = NULL, snames = NULL, inames = NULL)
```

## Arguments

| | |
|---|---|
| `nstates` | The number of latent states/classes of the model. |
| `nclasses` | The number of classes of an lca model, ie the number of states in a `dmm` model. They are now called classes because they do not change over time. |
| `itemtypes` | A vector of length `nitems` providing the type of measurement, 1 for continuous (=gaussian) data, 2 for a binary item, n>3 for categorical items with n answer possibilities. Answer categories are assumed to be unordered categorical. Ordinal responses can be implemented using inequality and/or linear constraints. |
| `modname` | A character string with the name of the model, good when fitting many models. Components of mixture models keep their own names. Names are printed in the summary. Boring default names are provided. |
| `fixed` | A vector of length the number of parameters of the model idicating whether parameters are fixed (0) or not (>0). This may be identical to conpat (see below). |
| `stval` | Start values of the parameters. These will be random if not specified. Start values must be specified (for all parameters) if there are fixed parameters. |
| `conrows` | Argument `conrows` can be used to specify general constraints between parameters. See details below. |
| `conpat` | Argument `conpat` can be used to specify fixed parameters and equality constraints. It can not be used in conjuction with fixed. See details below. |
| `tdfix,tdst` | The first is a logical vector indicating (with 1's) which parameters are dependent on covariates (it should have length npars). Tdst provides the starting values for the regression parameters. Using tdcov=TRUE in fitdmm will actually fit the regression parameters. The covariate itself has to be specified in the data as "covariate" (see help on markovdata) and should be scaled to 0-1. |
| `linmat` | A complete matrix of linear constraints. This argument is intended for internal use only, it is used by the fit routine to re-create the model with the fitted parameter values. Warning: use of this argument results in complete replacement of the otherwise created matrix A, which contains e.g. sum contraints for transition matrix parameters. If `linmat` is provided, make sure it is correct, otherwise strange results may occur in fitting models. |
| `snames` | Names for the states may be provided in statenames. Defaults are State1, State2 etc. They are printed in the summary. |
| `inames` | Names for items may be provided in itemnames. Defaults are Item1, Item2 etc. They are printed in the summary. |
| `dmm` | Object of class `dmm`. |

| | |
|---|---|
| `precision` | Precision sets the number of digits to be printed in the summary functions. |
| `se` | Vector with standard errors, these are passed on from the summary.fit function if and when ses are available. |
| `specs,...` | Internal use. |
| `object` | An object of class `dmm`. |

## Details

The function `dmm` creates an object of class `dmm` and sets random initial parameter values if these are not provided. Even though `dmm` is not a mixture of Markov models, the mixture parameter is is included in the parameter vector. This is important when specifying constraints. Parameters are ordered as follows: the first parameter(s) are the mixing proportions of the mixture of Markov and/or latent class models. I.e., when a single latent class model or a single Markov chain is fitted, this mixture proportion has value 1.0 and is it is fixed in estimation. After the mixing proportions, the next parameters in the parameter vector are the transition matrix parameters, the square of nstates in row-major order. That is, first the transition probabilities from state 1 to all the other states are given, then the probabilities from state 2 to all the other states etc. Next are the observation matrix parameters. These are provided consecutively for each state/class. Ie a trichtomous item model with two states has 6 observation parameters; the first three are the probabilities of observing category 1, 2 and 3 respectively in state 1 (which sum to one), and then similarly for state 2. As another example: suppose we have model for one binary item and one gaussian item, in that order, we would have 4 observation parameters for each state, first the probabilities of observing a symbol from category 1 or 2 in state 1, the two parameters, the mean and standard deviation for state 1, and then the same state 2 (see the example in fitdmm with data from rudy). Finally the initial state probabilities are provided, in the order of the states. In the case of a latent class model or a finite mixture model, these parameters are usually denote as the mixture proportions.

Linear constraints can be set using arguments `conrows` and `conpat`. `conrows` must be contain nc by npars values, in row major order, with nc the number of contraints to be specified. `conrows` is used to define general linear constraints. A row of `conrows` must contain the partial derivatives of a general linear constraint with respect to each of the parameters. Suppose we want the constraint x1 -2*x2=0, one row of conrows should contain a 1 in position one and -2 in position and zeroes in the remaining positions. In the function `mixdmm` `conrows` is understood to specify linear constraints on the mixing proportions only. As a consequence, it is not possible to easily constrain parameters between components of a mixture model.

`conpat` can be used as a shortcut for both fixed and conrows. It must be a single vector of length npars contaning 0's (zeroes) for fixed parameters, 1's (ones) for free parameters and higher numbers for possibly equality constrained parameters. E.g. `conpat=c(1,1,0,2,2,3,3,3)` would indicate that pars 1 and 2 are freely estimated, par 3 is fixed at its startvalue (which must be provided in this case), par 4 and 5 are to estimated equal and pars 6, 7 and 8 are also to be estimated equal.

## Value

`dmm` returns an object of class `dmm` which has its own summary method. This will print the parameter values, itemtypes, number of (free) parameters, and the number of states.

There is no print method. Using print will print all fields of the model which is a list of the following:

| | |
|---|---|
| `modname` | See above. |
| `nstates` | See above |
| `snames` | See above. |
| `nitems` | The number of items(=length(itemtypes)). |
| `itemtypes` | See above. |
| `inames` | See above. |
| `npars` | The total parameter count of the model. |
| `nparstotal` | The total number of parameters of when the covariate parameters are included. |
| `freepars` | The number of freely estimated parameters (it is computed as sum(as.logical(fixed))-rank(qr(A)). |
| `freeparsnotd` | The number of freely estimated parameters (it is computed as sum(as.logical(fixed))-rank(qr(A)); this version without the covariate parameters. |
| `pars` | A vector of length npars containing parameter values. |
| `fixed` | `fixed` is a (logical) vector of length npars specifying which parameters are fixed and which are not. |
| `A` | The matrix A contains the general linear constraints of the model. nrow(A) is the number of linear constraints. A starts with a number of rows for the sum constraints for the transition, observation and initial state parameters, after which the user provided constraints are added. |
| `bu,bl` | bu and bl represent the upper and lower bounds of the parameters and the constraints. These vectors are each of length npars + nrow(A). |
| `bllin,bulin` | The lower and upper bounds of the linear constraints. |
| `td,tdin,tdtr,tdob,tdfit` | Logicals indicating whehter there covariates, in which parameters they are, and whether they are estimated or not (the latter is used to decide whether to print those values or not). |
| `st` | Logical indicating whether the model has user specified starting values. |

`lca` returns an object of class `dmm, lca`, and is otherwise identical to a `dmm` object. The only difference is that the transition matrix parameters are irrelevant, and consequently they are not printed in the summary function.

## Author(s)

Ingmar Visser ⟨i.visser@uva.nl⟩

## References

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

On latent class models: A. L. McCutcheon (1987). *Latent class analysis.* Sage Publications.

**See Also**

`mixdmm` on defining mixtures of `dmm`'s, `mgdmm` for defining multi group models, and `generate` for generating data from models.

**Examples**

```
# create a 2 state model with one continuous and one binary response
# with start values provided in st
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)
summary(mod)

# 2 class latent class model with equal conditional probabilities in each class
stv=c(1,rep(c(0.9,0.1),5),rep(c(0.1,0.9),5),0.5,0.5)
# here the conditional probs of the first item are set equal to those in
# the subsequent items
conpat=c(1,rep(c(2,3),5),rep(c(4,5),5),1,1)
lc=lca(ncl=2,itemtypes=rep(2,5),conpat=conpat,stv=stv)
summary(lc)
```

### 3.3.1 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)
> summary(gen)

Data set:                   2-item data
 nr of items:               2
 item type(s):              1 2
 item name(s):              1 2
 length(s) of series:       100 50
 nr of independent series:  2
 data:                      1.662868 2.212274 2.745824 1.660324 1.394527 7.677464  ...
```

---

`generate` *Generate data from a dependent mixture model*

---

**Description**

generate `generate` generates a dataset according to a given `dmm`.

```
> plot(gen)
```

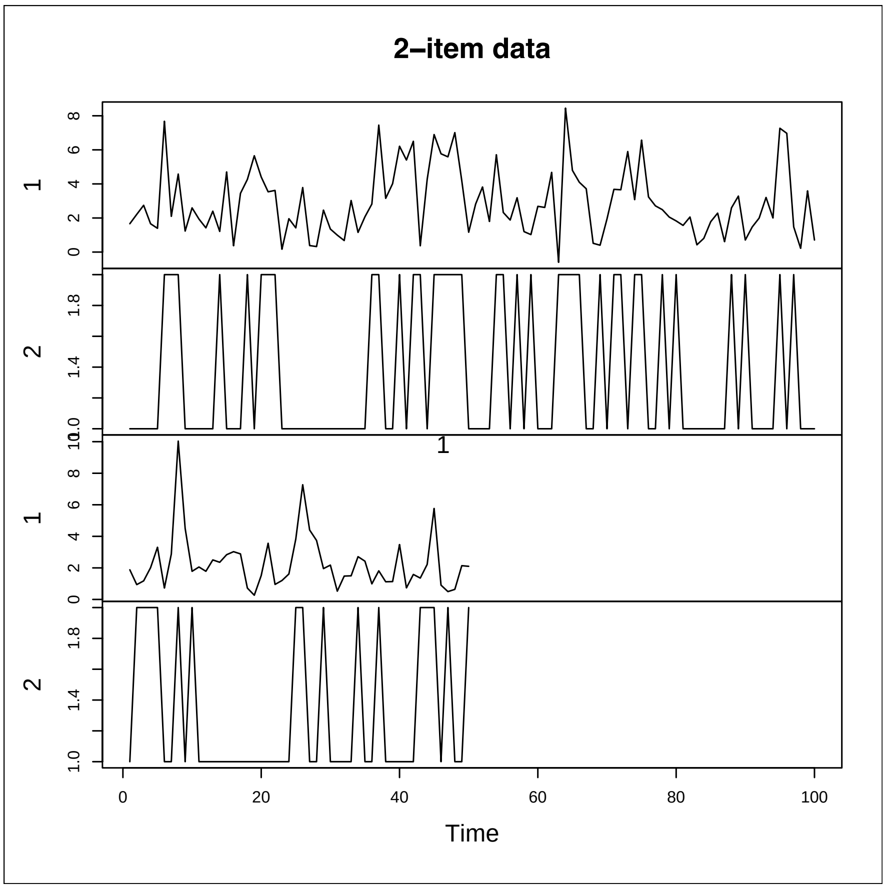

Figure 3.1: Two timeseries generated by 2-state model with one gaussian item and one binary item.

## Usage

```
generate(ntimes,dmm,nreal=1)
```

## Arguments

| | |
|---|---|
| `ntimes` | The number of repeated measurements, ie the length of the time series (this may be a vector containing the lengths of independent realiazations). |
| `dmm` | Object of class `dmm` or `mixdmm`. |
| `nreal` | The number of independent realizations that is to generated. Each of them will have the dimension of `ntimes`; all this does is replace ntimes by rep(ntimes,nreal). |

## Details

`generate` generates a date set of the specified dimensions `ntimes` and `nreal` using the parameter values in `dmm`, which should be an object of class `dmm` or `mixdmm`. `generate` does not handle multi group models, which can be run separately.

## Value

Generate returns an object of class `markovdata`. The return object has an attribute called instates, a vector with the starting states of each realization. When the model is a mixture the return has another attribute `incomp` containing the components of each realization.

## Author(s)

Ingmar Visser ⟨i.visser@uva.nl⟩

## See Also

`dmm`, `markovdata`

## Examples

```
# create a 2 state model with one continuous and one binary response
# with start values provided in st
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)

# generate two series of lengths 100 and 50 respectively using above model
gen<-generate(c(100,50),mod)

summary(gen)
plot(gen)
```

## 3.4 Fitting models

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

```
> data(speed)

name=speed:         file= .../speed.rda::         found

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

Initial loglikelihood:  -297.4306
iteration = 0
Step:
[1] 0 0 0 0 0 0 0 0 0
Parameter:
[1]  0.4074968 -0.6030893 -0.5559689  0.7830820 -0.1846599  6.4032844  0.2318094
[8]  5.5372838  0.2189520
Function Value
[1] 297.4306
Gradient:
[1]  17.838774   4.395048   1.009810   6.808036   9.848922  30.788829 -43.417653
[8]  32.399634  54.905265

iteration = 16
Parameter:
[1]  0.3877964 -0.6050630 -0.5429171  0.7942191 -0.2057752  6.3914953  0.2396807
[8]  5.5200651  0.2019121
Function Value
[1] 296.1200
Gradient:
[1] -0.55687642  0.99484837  3.34099578  0.08263532  3.08080738 -0.57541695
[7] -0.09220585 -0.81976170 -0.25060547

Successive iterates within tolerance.
Current iterate is probably solution.

Final loglikelihood:  -296.1200
Computing posteriors
Computing standard errors
This took  16 iterations,  6.044  seconds
```

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 gives the first and the last iteration of the optimization; 0 gives no output, and setting it to 5 or higher values will produce output at each iteration. These values correspond with the 0,1, and 2 printlevel of `nlm`. When using `optim`, the `printlevel` argument is used to set the `REPORT` argument of `optim` (see its help page for details). Printlevel 0 gives report 0, printlevel 1 gives report 10, printlevels 2–4 give report 5 and printlevel>4 gives report 1, producing output at every iteration. Printlevels starting from 15 and higher produce increasingly annoying output from the C-routines that compute the loglikelihood.

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 sum(ntimes(dat)). The first column contains the a posteriori component model, 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.

```
> summary(fitex)

Model:  2 -state model  fitted at  Wed Jul 12 16:37:59 2006
Optimization information, method is  nlm
 Iterations:  16
 Inform:  2  (look up the respective manuals for more information.)

 Loglikelihood of fitted model:  -296.12
 AIC:  610.24
 BIC:  647.001
 Number of observations (used in BIC):  439
 Fitted model
 Model:  2 -state model
 Number of parameters:  15
 Free parameters:       9
 Number of states:      2
 Number of items:       2
 Item types:            1 2

 Parameter values, transition matrix

       State1 State2
State1  0.916  0.084
se      0.018  0.018
t      50.854  4.671
State2  0.104  0.896
se      0.024  0.024
t       4.263 36.678


 Parameter values, observation parameters

       Item1,mean Item1,stddev Item2,p 1 Item2,p 2
State1      6.391        0.240     0.098     0.902
se          0.016        0.012     0.019     0.019
t         399.429       20.735     5.060    46.543
State2      5.520        0.202     0.469     0.531
```

```
se          0.017        0.014     0.037     0.037
t         324.383       14.475    12.638    14.313


 Parameter values, initial state probabilies

    State1 State2
val  1.000  0.000
se   0.578  0.578
t    1.731  0.000
```

---

`fitdmm` *Fitting Dependent Mixture Models*

---

## Description

fitdmm `fitdmm` fits mixtures of hidden/latent Markov models on arbitrary length time series of mixed categorical and continuous data. This includes latent class models and finite mixture models (for time series of length 1), which are in effect independent mixture models.

`posterior` `posterior` computes the most likely latent state sequence for a given dataset and model.

## Usage

```
fitdmm(dat,dmm,printlevel=1,poster=TRUE,tdcov=0,ses=TRUE,
                method="nlm",der=1,vfactor=15,iterlim=100,
                accuracy="standard",kmst=!dmm$st,kmrep=5,postst=TRUE)
loglike(dat, dmm, tdcov = 0, grad = FALSE, hess = FALSE, set
        = TRUE, grInd = 0, sca = 1, printlevel = 1)
posterior(dat,dmm,tdcov=0,printlevel=1)
computeSes(dat,dmm)
bootstrap(object,dat,samples=100, pvalonly=0,...)
## S3 method for class 'fit':
summary(object, precision=3, fd=1, ...)
oneliner(object,precision=3)
```

## Arguments

| | |
|---|---|
| `dat` | An object (or list of objects) of class `md`, see markovdata. If dat is a list of objects of class `md` a multigroup model is fitted on these data sets. |
| `dmm` | An object (or a list of objects) of class `dmm`, see dmm. If dmm is a list of objects of class `dmm`, these are taken to components of a mixture of dmm's model and will be coerced to class `mixdmm`. In any case, the model that is fitted a multigroup mixture of dmm's with default ngroups=1 and number of components=1. |

| | |
|---|---|
| `printlevel` | `printlevel` controls the output provided by the C-routines that are called to optimize the parameters. The default of 1 provides minmal output: just the initial and final loglikelihood of the model. Setting higher values will provide more output on the progress the iterations. |
| `poster` | By default posteriors are computed, the result of which can be found in fit$post. |
| `method` | This is the optimization algorithm that is used. NLM is the default method. There is further support for optim and NPSOL. |
| `der` | Specifies whether derivatives are to be used in optimization. |
| `vfactor` | vfactor controls optimization in optim and nlm. Since in those routines there is no possibility for enforcing constraints, constraints are enforced by adding a penalty term to the loglikelihood. The penalty term is printed at the end of optimization if it is not close enough to zero. This may have several reasons. When parameters are estimated at bounds for example. This can be solved by fixing those parameters on their boundary values. When this is not acceptable vfactor may be increased such that the penalty is larger and the probability that they actually hold in the fitted model is correspondingly higher. |
| `tdcov` | Logical, when set to TRUE, given that the model and data have covariates, the corresponding parameters will be estimated. |
| `ses` | Logical, determines whether standard errors are computed after optimization. |
| `iterlim` | The iteration limit for npsol, defaults to 100, which may be too low for large models. |
| `accuracy` | This argument can be used to set accuracy of optimization when using `nlm` as optimizer. It can take values ”standard” (the default), ”high” and ”best” for increasing levels of accuracy. |
| `grad` | logical; if TRUE the gradients are returned. |
| `hess` | logical; if TRUE the hessian is returned; it is not implemented currently and hence setting it to true will produce a warning. |
| `set` | Whith the default value TRUE, the data and models parameters are sent to the C/C++ routines before computing the loglikelihood. When set is FALSE, this is not done. If an incorrect model was set earlier in the C-routines this may cause serious errors and/or crashes. |
| `sca` | If set to -1.0 the negative loglikelihood, gradients and hessian are returned. |
| `object` | An object of class `fit`, ie the return value of fitdmm. |
| `kmst,postst` | These arguments control the generation of starting values by kmeans and posterior estimates respectively. |
| `kmrep` | If no starting values are provided, `kmrep` sets of starting values are generated using kmeans in appropriate cases. The best resulting set of starting values is optimized further. |
| `grInd` | Logical argument; if TRUE, individual contributions of each independent realization to the gradient vector will be returned. |

| | |
|---|---|
| `fd` | Print the finite difference based standard errors in the summary if both those and bootstrapped standard errors are available. |
| `samples` | The number of samples to be used in bootstrapping. |
| `pvalonly` | Logical, if 1 only a bootstrapped pvalue is returned and not fitted paramaters to compute standard errors, optimization is truncated when the loglikelihood is better than the original loglikelihood. |
| `precision` | Precision sets the number of digits to be printed in the summary functions. |
| `...` | Used in summary. |

### Details

The function `fitdmm` optimizes the parameters of a mixture of `dmm`s using a general purpose optimization routine subject to linear constraints on the parameters.

### Value

`fitdmm` returns an object of class `fit` which has a summary method that prints the summary of the fitted model, and the following fields:

| | |
|---|---|
| `date,timeUsed,totMem` | The date that the model was fitted, the time it took to so and the memory usage. |
| `loglike` | The loglikelihood of the fitted model. |
| `aic` | The AIC of the fitted model. |
| `bic` | The BIC of the fitted model. |
| `mod` | The fitted model. |
| `post` | See function posterior for details. |
| `logl` | The loglikelihood. |
| `gr,grset` | `gr` contains the gradients. `grset` is a logical vector giving information as to which gradients are set, currently all gradients are set except the gradients for the mixing proportions. |
| `hs,hsset` | `hs` contains the hessian. `hsset` is a logical giving information as to which elements are computed. |
| `states` | A matrix of dimension 2+sum(nstates) by sum(length(ntimes)) containing in the first column the a posteriori component, in the second column the a posteriori state and in the remaining column the posterior probabilities of all states. |
| `comp` | Contains the posterior component number for each independent realization; all ones for a single component model. |

`computeSes` returns a vector of length `npars` with the standard errors and a matrix `hs` with the hessian used to compute them. The routine is not fail safe and can produce errors, ie when the (corrected) hessian is singular.

`bootstrap` returns an object of class `fit` with three extra fields, the bootstrapped standard errors, bse, a matrix with goodness-of-fit measures of the bootstrap samples, ie logl, AIC and BIC and pbetter, which is the proportion of bootstrap samples that resulted in better fits than the original model.

`summary.fit` pretty-prints the outputs.

`oneliner` returns a vector of loglike, aic, bic, mod$npars$, $mod$freepars, date.

## Note

The `repeated` library by Jim Lindsey fits hidden markov models. `fitdmm` fits time series of arbitrary length and mixtures of `dmm`s, where, to the best of my knowledge, other packages are limited due to the different optimization routines that are commonly used for these types of models (this is certainly so for categorical data models).

## Author(s)

Ingmar Visser ⟨i.visser@uva.nl⟩, Development of this pacakge was supported by European Commission grant 51652 (NEST) and by a VENI grant from the Dutch Organization for Scientific Research (NWO).

## References

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

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

## See Also

`dmm`,`markovdata`,`repeated`

## Examples

```
# COMBINED RT AND CORRECT/INCORRECT SCORES from a 'switching' experiment

data(speed)
mod <- dmm(nsta=2,itemt=c(1,2)) # gaussian and binary items
fit1 <- fitdmm(dat=speed,dmm=mod)
summary(fit1)

# add some constraints using conpat
conpat=rep(1,15)
conpat[1]=0
conpat[14:15]=0
conpat[8:9]=0
# use starting values from the previous model fit, except for the guessing
# parameters which should really be 0.5
stv=c(1,.896,.104,.084,.916,5.52,.20,.5,.5,6.39,.24,.098,.90,0,1)
mod=dmm(nstates=2,itemt=c("n",2),stval=stv,conpat=conpat)

fit2 <- fitdmm(dat=speed,dmm=mod)
```

```
summary(fit2)

# add covariates to the model to incorporate the fact the accuracy pay off changes per trial
# 2-state model with covariates + other constraints
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
stcov=rep(0,15)
stcov[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=stcov,modname="twoboth+cov"

fit3 <- fitdmm(dat=speed,dmm=mod,tdcov=1,der=0,ses=0,vfa=80,accu="best")
summary(fit3)

# split the data into three time series
data(speed)
r1=markovdata(dat=speed[1:168,],item=itemtypes(speed))
r2=markovdata(dat=speed[169:302,],item=itemtypes(speed))
r3=markovdata(dat=speed[303:439,],item=itemtypes(speed))

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

# define 3-group model with equal transition parameters, and no
# equalities between the obser parameters
mgr <-mgdmm(dmm=mod,ng=3,trans=TRUE,obser=FALSE)

fitmg <- fitdmm(dat=list(r1,r2,r3),dmm=mgr)
summary(fitmg)

# LEARNING DATA AND MODELS (with absorbing states)

data(discrimination)

# all or none model with error prob in the learned state
fixed = c(0,0,0,1,1,1,1,0,0,0,0)
stv = c(1,1,0,0.03,0.97,0.1,0.9,0.5,0.5,0,1)
allor <- dmm(nstates=2,itemtypes=2,fixed=fixed,stval=stv,modname="All-or-none")

# Concept identification model: learning only after an error
st=c(1,1,0,0,0,0.5,0.5,0.5,0.25,0.25,0.05,0.95,0,1,1,0,0.25,0.375,0.375)
```

```
# fix some parameters
fx=rep(0,19)
fx[8:12]=1
fx[17:19]=1
# add a couple of constraints
conr1 <- rep(0,19)
conr1[9]=1
conr1[10]=-1
conr2 <- rep(0,19)
conr2[18]=1
conr2[19]=-1
conr3 <- rep(0,19)
conr3[8]=1
conr3[17]=-2
conr=c(conr1,conr2,conr3)
cim <- dmm(nstates=3,itemtypes=2,fixed=fx,conrows=conr,stval=st,modname="CIM")

# define a mixture of the above models ...
mix <- mixdmm(dmm=list(allor,cim),modname="MixAllCim")

# ... and fit it on the combined data discrimination
fitmix <- fitdmm(discrimination,mix)
summary(fitmix)
```

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 mostly done through reparametrization. Inequality constraints are enforced by adding a penalty to the loglikelihood when the constraint is not satisfied. The penalty is linear in the amount by which the constraint is not satisfied, and not logarithmic or something similar which is often used (see e.g. the documentation for `constrOptim` which uses a logarithmic boundary for inequality constraints). This has advantages and disadvantages. There are two marked disadvantages. First, the loglikelihood is not smooth at the boundary of the paramter space. Second, it can happen that the constraint is not satisfied. Whenever cosntraints are not satisfied `fitdmm` exits with a warning stating the amount by which it is not satisfied. This can be remedied by upping the `vfactor` argument which simply increases the penalty by this factor (its default value is 5). An advantage is that using a linear penalty, it is possible that the parameter is estimated at the boundary, which is prohibited with logarithmic boundaries.

**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, in the order listed in Table 3.1
   (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, 0.896, 0.104, 0.084, 0.916, 5.52, 0.2, 0.5, 0.5, 6.39,
+     0.24, 0.098, 0.902, 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.

The constrained model has the following estimated parameters[1]:

```
> summary(fitfix)

Model:  2 -state model  fitted at  Fri Apr 15 14:52:22 2005
Optimization information, method is  nlm
 Iterations:  11
 Inform:  1  (look up the respective manuals for more information.)

 Loglikelihood of fitted model:  -296.585
 AIC:  605.169
 BIC:  629.676
 Number of observations (used in BIC):  439
 Fitted model
 Model:  2 -state model
 Number of parameters:  15
 Free parameters:       6
```

[1]Note that in running this example with the starting values from the unconstrained model, the initial loglikelihood is worse than the final loglikelihood because the initial likelihood is based on parameters that do not satisfy the constraints.

```
 Number of states:      2
 Number of items:       2
 Item types:            normal 2

 Parameter values, transition matrix

       State1 State2
State1  0.909  0.091
se      0.015  0.015
t      61.030  6.088
State2  0.091  0.909
se      0.015  0.015
t       6.088 61.030


 Parameter values, observation parameters

       Item1,mean Item1,stddev Item2,p 1 Item2,p 2
State1      5.521        0.203     0.500     0.500
se          0.017        0.014     0.000     0.000
t         325.190       14.623        NA        NA
State2      6.392        0.239     0.098     0.902
se          0.016        0.012     0.019     0.019
t         400.978       20.788     5.052    46.525


 Parameter values, initial state probabilies

    State1 State2
val      0      1
se       0      0
t       NA     NA
```

## 4.2 Multi group/case analysis

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 of length(ng). If it is a single model, each group has an identical structural model (same fixed and constrained parameters), and else each group has its model. 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[2].

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 resulting parameters are:

```
> summary(fitmg)

Model:  3 group model  fitted at  Thu Apr 28 11:17:27 2005
Optimization information, method is  nlm
 Iterations:  31
 Inform:  2  (look up the respective manuals for more information.)

 Loglikelihood of fitted model:  -280.026
 AIC:  594.053
 BIC:  663.489
 Number of observations (used in BIC):  439
 Fitted model
 Model:  3 group model
 Nr of groups:      3
 Nr of parameters:  45
 Free parameters:   17
 Model for group:  1
 Model:  2 -state model
 Number of parameters:  15
 Free parameters:       7
 Number of states:      2
 Number of items:       2
 Item types:            normal 2

 Parameter values, transition matrix

       State1 State2
State1  0.903  0.097
State2  0.084  0.916


 Parameter values, observation parameters

       Item1,mean Item1,stddev Item2,p 1 Item2,p 2
State1      5.616        0.259     0.500     0.500
State2      6.425        0.254     0.058     0.942


 Parameter values, initial state probabilies
```

[2]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.

```
    State1 State2
val    0.5    0.5

 Model for group:  2
 Model:  2 -state model
 Number of parameters:  15
 Free parameters:       7
 Number of states:      2
 Number of items:       2
 Item types:            normal 2

 Parameter values, transition matrix

       State1 State2
State1  0.903  0.097
State2  0.084  0.916


 Parameter values, observation parameters

       Item1,mean Item1,stddev Item2,p 1 Item2,p 2
State1      5.526        0.149       0.5       0.5
State2      6.408        0.238       0.1       0.9


 Parameter values, initial state probabilies

    State1 State2
val    0.5    0.5

 Model for group:  3
 Model:  2 -state model
 Number of parameters:  15
 Free parameters:       7
 Number of states:      2
 Number of items:       2
 Item types:            normal 2

 Parameter values, transition matrix

       State1 State2
State1  0.903  0.097
State2  0.084  0.916


 Parameter values, observation parameters

       Item1,mean Item1,stddev Item2,p 1 Item2,p 2
```

```
State1      5.422        0.167     0.500     0.500
State2      6.355        0.214     0.107     0.893


 Parameter values, initial state probabilies

    State1 State2
val    0.5    0.5
```

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 value $LR = 1.815$, and it 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 were already had to be constrained to ensure that the rows of the transition matrices sum to 1). The associated $p$-value for the statistic is $p = 0.77$, indicating that constraining the transition matrix parameters does not significantly worsen the goodness-of-fit of the model.

## `mgdmm` *Multi group model specification*

### Description

mgdmm `mgdmm` creates an object of class `mgd`, a multi-group model, from a given model of either class `dmm` or class `mixdmm` or lists of these.

### Usage

```
mgdmm(dmm,ng=1,modname=NULL,trans=FALSE,obser=FALSE,init=FALSE,conpat=NULL)
## S3 method for class 'mgd':
summary(object, specs=FALSE, precision=3, se=NULL, ...)
```

### Arguments

| | |
|---|---|
| `modname` | A character string with the name of the model, good when fitting many models. Components of mixture models keep their own names. Names are printed in the summary. Boring default names are provided. |
| `dmm` | Object (or list of objects) of class `dmm`; see details below. |
| `ng` | Number of groups for a multigroup model. |
| `trans,obser,init` | Logical arguments specify whether `trans`ition parameters, `obser`vation parameters and `init`ial state parameters should be estimated equal across groups. |
| `conpat` | Can be used to specify general linear constraints. See `dmm` for details. |

| | |
|---|---|
| `precision` | Precision sets the number of digits to be printed in the summary functions. |
| `se` | Vector with standard errors, these are passed on from the summary.fit function if and when ses are available. |
| `specs,...` | Internal use. |
| `object` | An object of class `mgd`. |

## Details

The function `mgdmm` can be used to define an `mgd`-model or multi group `dmm`. Its default behavior is to create `ng` copies of the `dmm` argument, thereby providing identical starting values for each group's model. If the `dmm` argument is a list of models of length `ng`, the starting values of those models will be used instead. This may save quite some cpu time when fitting large models by providing the parameter values of separately fitted models as starting values. Currently, `depmix` does not automatically generate starting values for multi group models.

## Value

`mgdmm` returns an object of class `mgd` which contains all the fields of an object of class `dmm` and the following extra:

| | |
|---|---|
| `ng` | `ng` is the number of groups in the multigroup model. |
| `mixmod` | `mixmod` is a list of length `ng` of `mixdmm` models for each group. |
| `itemtypes` | See above. |
| `npars,freepars,pars,fixed,A,bl,bu` | The same as above but now for the combined model, here npars equals the sum of npars of the component models plus the mixing proportions. |

## Author(s)

Ingmar Visser ⟨i.visser@uva.nl⟩

## See Also

`dmm` on defining single component models, and `mixdmm` for defining mixtures of `dmm`'s.

## Examples

```
# create a 2 state model with one continuous and one binary response
# with start values provided in st
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)

# define 3-group model with equal transition parameters, and no
# equalities between the obser parameters
mgr <- mgdmm(dmm=mod,ng=3,trans=TRUE,obser=FALSE)
summary(mgr)
```

## 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")

> summary(fittd)

Model:  twoboth+cov  fitted at  Fri Apr 15 14:52:43 2005
Optimization information, method is  nlm
 Iterations:  28
 Inform:  3  (look up the respective manuals for more information.)

 Loglikelihood of fitted model:  -284.856
 AIC:  585.713
 BIC:  618.389
 Number of observations (used in BIC):  439
 Fitted model
 Model:  twoboth+cov
 Number of parameters:  30
 Free parameters:      8
 Number of states:      2
 Number of items:       2
 Item types:            normal 2

 Parameter values, transition matrix

       State1 State2
State1  0.890  0.110
be     -0.224  0.224
State2  0.110  0.890
be     -0.110  0.110


 Parameter values, observation parameters

       Item1,mean Item1,stddev Item2,p 1 Item2,p 2
State1      5.516        0.198       0.5       0.5
State2      6.390        0.241       0.1       0.9


 Parameter values, initial state probabilies

    State1 State2
val      0      1
```

# 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 paramaters, either user provided or generated, are further boosted by using posterior estimates. 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 `postst` 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.

---

| `mixdmm` | *Mixture of dmm's specification* |
|---|---|

---

## Description

`mixdmm` creates an object of class `mixdmm`, ie a mixture of `dmm`'s, given a list of component models of class `dmm`.

## Usage

```
mixdmm(dmm, modname=NULL, mixprop=NULL, conrows=NULL)
## S3 method for class 'mixdmm':
summary(object, specs=FALSE, precision=3, se=NULL, ...)
```

## Arguments

| | |
|---|---|
| `dmm` | A list of `dmm` objects to form the mixture. |
| `modname` | A character string with the name of the model, good when fitting many models. Components of mixture models keep their own names. Names are printed in the summary. Boring default names are provided. |
| `conrows` | Argument `conrows` can be used to specify general constraints between parameters. |
| `mixprop` | Arugement `mixprop` can be used to set the initial values of the mixing proportions of a mixture of `dmm`'s. |
| `precision` | Precision sets the number of digits to be printed in the summary functions. |
| `object` | An object of class `mixdmm`. |
| `specs,...` | Internal use. Not functioning currently. |
| `se` | Vector with standard errors, these are passed on from the summary.fit function if and when ses are available. |

## Details

The function `mixdmm` can be used to define a mixture of `dmm`'s by providing a list of such objects as argument to this function. See the `dmm` helpfile on how to use the `conrows` argument. Note that it has to be of length `npars`, ie including all parameters of the model and not just the mixing proportions.

## Value

`mixdmm` returns an object of class `mixdmm` which has the same fields as a `dmm` object. In addition it has the following fields:

| | |
|---|---|
| `nrcomp` | The number of components of the mixture model. |
| `mod` | A list of the component models, that is a list of objects of class `dmm`. |

### Author(s)

Ingmar Visser ⟨i.visser@uva.nl⟩

### See Also

`dmm` on defining single component models, and `mgdmm` on defining multi group models. See `generate` for generating data.

### Examples

```
# define component 1
# all or none model with error prob in the learned state
fixed = c(0,0,0,1,1,1,1,0,0,0,0)
stv = c(1,1,0,0.07,0.93,0.9,0.1,0.5,0.5,0,1)
allor <- dmm(nstates=2,itemtypes=2,fixed=fixed,stval=stv,modname="All-or-none")

# define component 2
# Concept identification model: learning only after an error
st=c(1,1,0,0,0,0.5,0.5,0.5,0.25,0.25,0.8,0.2,1,0,0,1,0.25,0.375,0.375)
# fix some parameters
fx=rep(0,19)
fx[8:12]=1
fx[17:19]=1
# add a couple of constraints
conr1 <- rep(0,19)
conr1[9]=1
conr1[10]=-1
conr2 <- rep(0,19)
conr2[18]=1
conr2[19]=-1
conr3 <- rep(0,19)
conr3[8]=1
conr3[17]=-2
conr=c(conr1,conr2,conr3)
cim <- dmm(nstates=3,itemtypes=2,fixed=fx,conrows=conr,stval=st,modname="CIM")

# define a mixture of the above component models
mix <- mixdmm(dmm=list(allor,cim),modname="MixAllCim")
summary(mix)
```

An example of fitting a mixture of `dmm`'s is in the `fitdmm` helpfile. It fits the model in the example to data from a discrimination learning experiment which is provided as data set `discrimination`.

---

## `discrimination` *Discrimination Learning Data*

---

**Description**

This data set is from a simple discrimation learning experiment. It consists of 192 binary series of responses of different lengths. This is a subset of the data described by *Raijmakers et al. (2001)*, and it is analyzed much more extensively using latent Markov models and depmix in *Schmittmann et al. (2006)* and *Visser et al. (2006)*..

**Usage**

```
data(discrimination)
```

**Format**

An object of class `markovdata`.

**Source**

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

Ingmar Visser, Verena D. Schmittmann, and Maartje E. J. Raijmakers (2006). Markov process models for discrimination learning. In: Kees van Montfort, Han Oud, and Albert Satorra (Eds.), *Longitudinal models in the behavioral and related sciences*, Mahwah (NJ): Lawrence Erlbaum Associates (in press).

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

## 5.4 Known issues and future plans

**Constraint violations** Parameter optimization is done by adding a penalty to the log likelihood whenever constraints are not satisfied, ie linear inequality and box constraints. Equality constraints are fitted by reparametrization and do not suffer from this problem. Refer to the section on parameter estimation and the `fitdmm` help page on how to deal with constraint violations. Future plans include using Lagrange multipliers to overcome this problem.

# 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, 2006. 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, Conor V. Dolan, Han L. J. van der Maas, and Michael C. Neale. Discrete latent Markov models for normally distributed response data. *Multivariate Behavioral Research*, 40(4):461–488, 2005.

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.

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.

Ingmar Visser, Verena D. Schmittmann, and Maartje E. J. Raijmakers. Markov process models for discrimination learning. In Kees van Montfort, Han Oud, and Albert Satorra, editors, *Longitudinal models in the behavioral and related sciences*, chapter xxx. Lawrence Erlbaum Associates, Mahwah (NJ), In Press.

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