

# Package ‘depmixS4’

June 30, 2009

**Version** 0.2-2

**Date** 2009-06-24

**Title** Dependent Mixture Models

**Author** Ingmar Visser <i.visser@uva.nl>, Maarten Speekenbrink <m.speekenbrink@ucl.ac.uk>

**Maintainer** Ingmar Visser <i.visser@uva.nl>

**Depends** R (>= 2.8.1), nnet, methods, MASS

**Suggests** Rdonlp2

**Description** Fit latent (hidden) Markov models on mixed categorical and continuous (timeseries) data, otherwise known as dependent mixture models

**License** GPL (>= 2)

**URL** <http://depmix.r-forge.r-project.org/>

**Repository** CRAN

**Repository/R-Forge/Project** depmix

**Repository/R-Forge/Revision** 277

**Date/Publication** 2009-06-30 06:55:04

## R topics documented:

depmixS4-package	2
AIC	3
balance	4
depmix	5
depmix	7
depmix-class	9
depmix-methods	10
depmix.fitted-class	12
depmix.sim-class	13

fit	14
forwardbackward	17
llratio	19
mix	19
mix-class	21
mix.fitted-class	23
mix.sim-class	24
posterior	25
response	26
response-class	27
response-classes	28
responses	29
simulate	31
speed	33

## Index 34

---

depmixS4-package	<i>depmixS4 provides classes for specifying and fitting hidden Markov models</i>
------------------	--

---

## Description

depmixS4 is a framework for specifying and fitting dependent mixture models, otherwise known as hidden or latent Markov models. Optimization is done with the EM algorithm or optionally with Rdonlp2 when (general linear (in-)equality) constraints on the parameters need to be incorporated. Models can be fitted on (multiple) sets of observations. The response densities for each state may be chosen from the GLM family, or a multinomial. User defined response densities are easy to add.

Mixture or latent class (regression) models can also be fitted; these are the limit case in which the length of observed time series is 1 for all cases.

## Details

Package:	depmixS4
Type:	Package
Version:	0.2-2
Date:	2009-13-05
License:	GPL

Model fitting is done in two steps; first, models are specified through the `depmix` function (or the `mix` function for mixture and latent class models), which both use standard `glm` style arguments to specify the observed distributions; second, the model needs to be fitted by using the `fit` function; imposing constraints is done through the `fit` function. Standard output includes the optimized parameters and the posterior densities for the states and the optimal state sequence.

**Author(s)**

Ingmar Visser & Maarten Speekenbrink

Maintainer: 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**

[depmix](#), [fit](#)

**Examples**

```
# These should be added at some point ...
```

---

AIC

*Compute AIC and BIC for (dep-)mix objects*

---

**Description**

Compute AIC and BIC for `depmix` and `mix` objects.

**Usage**

```
AIC(object, ..., k = 2)
BIC(object, ...)
```

**Arguments**

<code>object</code>	A <code>depmix</code> or <code>mix</code> model.
<code>...</code>	Not used currently.
<code>k</code>	The penalty factor which defaults to 2.

**Details**

The `n` that is used in the BIC is `sum(ntimes(object))`, where `object` is the model under consideration.

**Value**

The value of the AIC or BIC respectively.

**Author(s)**

Ingmar Visser

**References**

H. Akaike. Information theory and an extension of the maximum likelihood principle. In B. N. Petrov and F. Csaki, editors, *Second International Symposium on Information Theory*, p. 267-281. Budapest: Akademiai Kiado, 1973.

G. Schwarz. Estimating the dimension of a model. *Annals of Statistics*, volume 6, p. 4761-464, 1978.

---

balance

*Balance Scale Data*

---

**Description**

Balance scale data of four distance items from 779 subjects; participants' ages are included.

**Usage**

`data(balance)`

**Format**

A data frame with 779 observations on the following variables.

**sex** Participants' sex.

**agedays** Age in days.

**age** Age in years.

**t1** Trichotomously scored distance item.

**t2** Trichotomously scored distance item.

**t3** Trichotomously scored distance item.

**t4** Trichotomously scored distance item.

**d1** Dichotomously scored distance item.

**d2** Dichotomously scored distance item.

**d3** Dichotomously scored distance item.

**d4** Dichotomously scored distance item.

**Source**

Brenda Jansen (2001), *Development of reasoning on the balance scale task: Psychometric assessment of cognitive strategies*. PhD thesis, University of Amsterdam, Department of Psychology.

**Examples**

`data(balance)`

depmix

*Dependent Mixture Model Specification***Description**

depmix creates an object of class depmix, a dependent mixture model, otherwise known as hidden Markov model. For a short description of the package see [depmixS4](#).

**Usage**

```
depmix(response, data=NULL, nstates, transition=~1, family=gaussian(),
        prior=~1, initdata=NULL, respstart=NULL, trstart=NULL, instart=NULL,
        ntimes=NULL, ...)
```

**Arguments**

response	The response to be modeled; either a formula or a list of formulae in the multivariate case; this interfaces to the glm distributions. See 'Details'.
data	An optional data.frame to interpret the variables in the response and transition arguments.
nstates	The number of states of the model.
transition	A one-sided formula specifying the model for the transitions. See 'Details'.
family	A family argument for the response. This must be a list of family's if the response is multivariate.
prior	A one-sided formula specifying the density for the prior or initial state probabilities.
initdata	An optional data.frame to interpret the variables occurring in prior. The number of rows of this data.frame must be equal to the number of cases being modeled. See 'Details'.
respstart	Starting values for the parameters of the response models.
trstart	Starting values for the parameters of the transition models.
instart	Starting values for the parameters of the prior or initial state probability model.
ntimes	A vector specifying the lengths of individual, ie independent, time series. If not specified, the responses are assumed to form a single time series. If the data argument has an attribute ntimes, then this is used.
...	Not used currently.

## Details

The function `depmix` creates an S4 object of class `depmix`, which needs to be fitted using `fit` to optimize the parameters.

The response model(s) are created by call(s) to `response` providing the response formula and the family specifying the error distribution. If response is a list of formulae, the `response`'s are assumed to be independent conditional on the latent state.

The transitions are modeled as a multinomial logistic model for each state. Hence, the transition matrix can be modeled as time-dependent, depending on predictors. The prior density is also modeled as a multinomial logistic. Both are created by calls to `transInit`.

Starting values may be provided by the respective arguments. The order in which parameters must be provided can be easily studied by using the `setpars` function.

Linear constraints on parameters can be provided as argument to the `fit` function.

The print function prints the formulae for the response, transition and prior models along with their parameter values.

## Value

`depmix` returns an object of class `depmix` which has the following slots:

<code>response</code>	A list of a list of response models; the first index runs over states; the second index runs over the independent responses in case a multivariate response is provided.
<code>transition</code>	A list of <code>transInit</code> models, ie multinomial logistic models with length the number of states.
<code>prior</code> <code>dens, trDens, init</code>	A multinomial logistic model for the initial state probabilities. See <code>depmix-class</code> help for details. For internal use.
<code>stationary</code>	Logical indicating whether the transitions are time-dependent or not; for internal use.
<code>ntimes</code>	A vector containing the lengths of independent time series; if data is provided, <code>sum(ntimes)</code> must be equal to <code>nrow(data)</code> .
<code>nstates</code>	The number of states of the model.
<code>nresp</code>	The number of independent responses.
<code>npars</code>	The total number of parameters of the model. Note: this is <i>not</i> the degrees of freedom because there are redundancies in the parameters, in particular in the multinomial models for the transitions and prior probabilities.

## Author(s)

Ingmar Visser & Maarten Speekenbrink

## 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**

[fit](#), [transInit](#), [response](#), [depmix-methods](#) for accessor functions to depmix objects.

**Examples**

```
# create a 2 state model with one continuous and one binary response
data(speed)
mod <- depmix(list(rt~1,corr~1),data=speed,nstates=2,family=list(gaussian(),multinomial()))
# print the model, formulae and parameter values
mod
```

depmix

*Dependent Mixture Model Specification: the long way***Description**

`makeDepmix` creates an object of class `depmix`. This function is meant for full control, e.g. specifying each response model and the transition and prior models 'by hand'. For the default easier specification of models, please see [depmix](#). This function is meant for specifying one's own response models.

**Usage**

```
makeDepmix(response, transition, prior, ntimes = NULL, stationary = TRUE,
...)
```

**Arguments**

<code>response</code>	A two-dimensional list of response models. See 'Details'.
<code>transition</code>	A list of transition models, each created by a call to <a href="#">transInit</a> on for possibilities of specifying such models. The length of this list should be the number of states of the model.
<code>prior</code>	The initial state probabilities model; created through a call to <a href="#">transInit</a> .
<code>ntimes</code>	A vector specifying the lengths of individual, ie independent, time series. If not specified, the responses are assumed to form a single time series.
<code>stationary</code>	Logical indicating whether the transition models include time-varying covariates; used internally to determine the dimensions of certain arrays, notably <code>trDens</code> .
<code>...</code>	Not used currently.

## Details

The function `makeDepmix` creates an S4 object of class `depmix`, which needs to be fitted using `fit` to optimize the parameters. This function is provided to have full control, eg by specifying one's own response models with distributions that are not provided.

The response model(s) should be created by call(s) to `response` or one's own created response models that should extend the response class and have the following methods: `dens`, `predict` and optionally `fit`. The fit function should have an argument `w`, providing the weights. If the fit function is not provided, optimization should be done by using `Rdonlp` (use `method="donlp"` in calling `fit` on the `depmix` model, this is currently not done automatically). The first index of response models runs over the states of the model, and the seconde index over the responses to be modeled.

## Value

See the `depmix` help page for the return value, a `depmix` object.

## Author(s)

Ingmar Visser & Maarten Speekenbrink

## See Also

`fit`, `transInit`, `response`, `depmix-methods` for accessor functions to `depmix` objects.

## Examples

```
# below example recreates the model from the depmix help page albeit in a
# roundabout way
```

```
data(speed)
```

```
rModels <- list(
  list(
    GLMresponse(formula=rt~1,data=speed,family=gaussian(),pstart=c(5.52,.202)),
    GLMresponse(formula=corr~1,data=speed,family=multinomial(),pstart=c(0.5,0.5))
  ),
  list(
    GLMresponse(formula=rt~1,data=speed,family=gaussian(),pstart=c(6.39,.24)),
    GLMresponse(formula=corr~1,data=speed,family=multinomial(),pstart=c(.1,.9))
  )
)
```

```
trstart=c(0.9,0.1,0.1,0.9)
```

```
transition <- list()
transition[[1]] <- transInit(~1,nstates=2,data=data.frame(1),pstart=c(trstart[1:2]))
transition[[2]] <- transInit(~1,nstates=2,data=data.frame(1),pstart=c(trstart[3:4]))
```

```
instart=c(0,1)
```

```
inMod <- transInit(~1,ns=2,ps=instart,data=data.frame(rep(1,3)))
```

```

mod <- makeDepmix(response=rModels,transition=transition,prior=inMod,ntimes=attr(speed,"ntim
logLik(mod)

fm <- fit(mod)

fm

summary(fm)

```

---

depmix-class      *Class "depmix"*

---

### Description

A `depmix` model.

### Slots

**response:** List of list of response objects.

**transition** List of `transInit` objects.

**prior:** `transInit` object.

**dens:** Array of dimension  $\text{sum}(\text{ntimes}) * \text{nresp} * \text{nstates}$  providing the densities of the observed responses for each state.

**trDens:** Array of dimension  $\text{sum}(\text{ntimes}) * \text{nstates}$  providing the probability of a state transition depending on the predictors.

**init:** Array of dimension  $\text{length}(\text{ntimes}) * \text{nstates}$  with the current predictions for the initial state probabilities.

**stationary:** Logical indicating whether the transitions are time-dependent or not; for internal use.

**ntimes:** A vector containing the lengths of independent time series; if data is provided,  $\text{sum}(\text{ntimes})$  must be equal to  $\text{nrow}(\text{data})$ .

**nstates:** The number of states of the model.

**nresp:** The number of independent responses.

**npars:** The total number of parameters of the model. This is not the degrees of freedom, ie there are redundancies in the parameters, in particular in the multinomial models for the transitions and prior.

### Accessor Functions

The following functions should be used for accessing the corresponding slots:

**npars:** The number of parameters of the model.

**nresp:** The number of responses.

**nstates:** The number of states.

**ntimes:** The vector of independent time series lengths.

**Author(s)**

Ingmar Visser

---

depmix-methods      *'depmix' and 'mix' methods.*

---

**Description**

Various methods for depmix and mix objects.

**Usage**

```
## S4 method for signature 'depmix':
  logLik(object,method="lystig")
## S4 method for signature 'mix':
  logLik(object,method="lystig")

## S4 method for signature 'depmix':
  nobs(object, ...)
## S4 method for signature 'mix':
  nobs(object, ...)

## S4 method for signature 'depmix':
  npar(object)
## S4 method for signature 'mix':
  npar(object)

## S4 method for signature 'depmix':
  freepars(object)
## S4 method for signature 'mix':
  freepars(object)

## S4 method for signature 'depmix':
  setpars(object,values, which="pars",...)
## S4 method for signature 'mix':
  setpars(object,values, which="pars",...)

## S4 method for signature 'depmix':
  getpars(object,which="pars",...)
## S4 method for signature 'mix':
  getpars(object,which="pars",...)
```

**Arguments**

object      A depmix or mix object.

values	To be used in <code>setpars</code> to set new parameter values; see the example.
method	The log likelihood can be computed by either the forward backward algorithm from Rabiner, 1989, or by the method of Lystig and Hughes, 2002. The latter is the default as it is faster because in the forward backward routine the state and transition smoothed probabilities are also computed which are not necessary for the log likelihood. Those smoothed variables, and the forward and backward variables are accessible through the <code>forwardbackward</code> function.
which	The default "pars" returns a vector of all parameters of a depmix object; the alternative value "fixed" return a logical vector of the same length indicating which parameters are fixed. The <code>setpars</code> functions sets parameters (or the logical fixed vector) to new values; <code>setpars</code> also recomputes the dens, trans and init slots of depmix objects. Note that the <code>getpars</code> and <code>setpars</code> functions for depmix objects simply call the functions of the same name for the response and transition models.
...	Not used currently.

### Value

logLik	returns a logLik object with attributes <code>df</code> and <code>nobs</code> .
nobs	returns the number of observations (used in computing the BIC).
npar	returns the number of parameters of a model.
freepars	returns the number of non-fixed parameters.
setpars	returns a (dep-)mix object with new parameter values.
getpars	returns a vector with the current parameter values.

### Author(s)

Ingmar Visser

### Examples

```
# create a 2 state model with one continuous and one binary response
data(speed)
mod <- depmix(list(rt~1,corr~1),data=speed,nstates=2,family=list(gaussian(),multinomial()))

# get the loglikelihood of the model
logLik(mod)

# to see the ordering of parameters to use in setpars
mod <- setpars(mod, value=1:npar(mod))
mod

# to see which parameters are fixed (by default only baseline parameters in
# the multinomial logistic models for the transition models and the initial
# state probabilities model
mod <- setpars(mod, getpars(mod,which="fixed"))
mod
```

---

 depmix.fitted-class

*Class "depmix.fitted"*


---

## Description

A fitted `depmix` model.

## Slots

A `depmix.fitted` object is a `depmix` object with three additional slots, here is the complete list:

**response:** List of list of `response` objects.

**transition** List of `transInit` objects.

**prior:** `transInit` object.

**dens:** Array of dimension  $\text{sum}(\text{ntimes}) * \text{nresp} * \text{nstates}$  providing the densities of the observed responses for each state.

**trDens:** Array of dimension  $\text{sum}(\text{ntimes}) * \text{nstates}$  providing the probability of a state transition depending on the predictors.

**init:** Array of dimension  $\text{length}(\text{ntimes}) * \text{nstates}$  with the current predictions for the initial state probabilities.

**stationary:** Logical indicating whether the transitions are time-dependent or not; for internal use.

**ntimes:** A vector containing the lengths of independent time series; if data is provided,  $\text{sum}(\text{ntimes})$  must be equal to  $\text{nrow}(\text{data})$ .

**nstates:** The number of states of the model.

**nresp:** The number of independent responses.

**npars:** The total number of parameters of the model. This is not the degrees of freedom, ie there are redundancies in the parameters, in particular in the multinomial models for the transitions and prior.

**message:** This provides some information on convergence, either from the EM algorithm or from `Rdonlp2`.

**conMat:** The linear constraint matrix, which has zero rows if there were no constraints.

**lin.lower** The lower bounds on the linear constraints.

**lin.upper** The upper bounds on the linear constraints.

**posterior:** Posterior (Viterbi) state sequence (not implemented currently).

## Details

The print function shows some convergence information, and the summary method shows the parameter estimates.

**Extends**

`depmix.fitted` extends the `depmix` class.

**Author(s)**

Ingmar Visser

---

`depmix.sim-class`    *Class "depmix.sim"*

---

**Description**

A `depmix.sim` model. The `depmix.sim` class directly extends the `depmix` class, and has an additional slot for the true states. A `depmix.sim` model can be generated by `simulate(mod, ...)`, where `mod` is a `depmix` model.

**Slots**

**response:** List of list of response objects.

**transition** List of `transInit` objects.

**prior:** `transInit` object.

**dens:** Array of dimension `sum(ntimes)*nresp*nstates` providing the densities of the observed responses for each state.

**trDens:** Array of dimension `sum(ntimes)*nstates` providing the probability of a state transition depending on the predictors.

**init:** Array of dimension `length(ntimes)*nstates` with the current predictions for the initial state probabilities.

**stationary:** Logical indicating whether the transitions are time-dependent or not; for internal use.

**ntimes:** A vector containing the lengths of independent time series; if data is provided, `sum(ntimes)` must be equal to `nrow(data)`.

**nstates:** The number of states of the model.

**nresp:** The number of independent responses.

**npars:** The total number of parameters of the model. This is not the degrees of freedom, ie there are redundancies in the parameters, in particular in the multinomial models for the transitions and prior.

**states:** A matrix with the true states.

**Accessor Functions**

The following functions should be used for accessing the corresponding slots:

**npars:** The number of parameters of the model.

**nresp:** The number of responses.

**nstates:** The number of states.

**ntimes:** The vector of independent time series lengths.

**Author(s)**

Maarten Speekenbrink

fit

*Fit 'depmix' or 'mix' models***Description**

`fit` optimizes parameters of `depmix` or `mix` models, optionally subject to general linear (in)equality constraints.

**Usage**

```
## S4 method for signature 'depmix':
  fit(object, fixed=NULL, equal=NULL, conrows=NULL,
       conrows.upper=0, conrows.lower=0, method=NULL, ...)

## S4 method for signature 'depmix.fitted':
  summary(object)

## S4 method for signature 'mix':
  fit(object, fixed=NULL, equal=NULL, conrows=NULL,
       conrows.upper=0, conrows.lower=0, method=NULL, ...)

## S4 method for signature 'mix.fitted':
  summary(object)
```

**Arguments**

<code>object</code>	An object of class (dep-)mix.
<code>fixed</code>	Vector of mode logical indicating which parameters should be fixed.
<code>equal</code>	Vector indicating equality constraints; see details.
<code>conrows</code>	Rows of a general linear constraint matrix; see details.
<code>conrows.upper</code> , <code>conrows.lower</code>	Upper and lower bounds for the linear constraints; see details.
<code>method</code>	The optimization method; mostly determined by constraints.
<code>...</code>	Further arguments passed on to the optimization methods.

## Details

Models are fitted by the EM algorithm if there are no constraints on the parameters. Otherwise the general optimizer `donlp2` from the package `Rdonlp2` is used which handles general linear (in-)equality constraints.

Three types of constraints can be specified on the parameters: fixed, equality, and general linear (in-)equality constraints. Constraint vectors should be of length `npar(object)`. See help on `getpars` and `setpars` about the ordering of parameters.

The `equal` argument is used to specify equality constraints: parameters that get the same integer number in this vector are estimated to be equal. Any integers can be used in this way except 0 and 1, which indicate fixed and free parameters, respectively.

Using the `donlp2` optimizer a Newton-Raphson scheme is employed to estimate parameters subject to linear constraints by imposing:

$$bl \leq A * x \leq bu,$$

where  $x$  is the parameter vector,  $bl$  is a vector of lower bounds,  $bu$  is a vector of upper bounds, and  $A$  is the constraint matrix.

The `conrows` argument is used to specify rows of  $A$  directly, and the `conrows.lower` and `conrows.upper` arguments to specify the bounds on the constraints. `conrows` is a matrix of `npar(object)` columns and one row for each constraint (a vector in the case of a single constraint). Examples of these three ways of constraining parameters are provided below.

`llratio` performs a log-likelihood ratio test on two `fit`'ted models; the first object should have the largest degrees of freedom (find out by using `freepars`).

## Value

`fit` returns an object of class `depmix.fitted` which contains the original `depmix` object, and further has slots:

```
message      : Convergence information.
conMat       : The constraint matrix A, see details.
posterior    : The posterior state sequence (computed with the viterbi algorithm), and the
               posterior probabilities (delta probabilities in Rabiner, 1989, notation).
```

The `print` method shows the `message` along with the likelihood and AIC and BIC; the `summary` method prints the parameter estimates.

Posterior densities and the viterbi state sequence can be accessed through `posterior`.

As fitted models are `depmixS4` models, they can be used as starting values for new fits, for example with constraints added.

## Author(s)

Ingmar Visser & Maarten Speekenbrink

## 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.

**Examples**

```

data(speed)

# 2-state model on rt and corr from speed data set with Pacc as covariate on the transition
# starting values for the transition pars (without those EM does not get off the ground)
set.seed(1)
tr=runif(6)
trst=c(tr[1:2],0,tr[3:5],0,tr[6])
mod1 <- depmix(list(rt~1,corr~1),data=speed,transition=~Pacc,nstates=2,family=list(gaussian(
  trstart=trst)
# fit the model
fmod1 <- fit(mod1)
fmod1 # to see the logLik and optimization information
# to see the parameters
summary(fmod1)

## Not run:
# NOTE: this requires Rdonlp2 package to be installed

# FIX SOME PARAMETERS

# get the starting values of this model to the optimized
# values of the previously fitted model to speed optimization

pars <- c(unlist(getpars(fmod1)))

# constrain the initial state probs to be 0 and 1
# also constrain the guessing probs to be 0.5 and 0.5
# (ie the probabilities of corr in state 1)
# change the ones that we want to constrain
pars[2]=+Inf # this means the process will always start in state 2
pars[14]=0 # the corr parameters in state 1 are now both 0, corresponding the 0.5 prob
mod2 <- setpars(mod1,pars)

# fix the parameters by setting:
free <- c(0,0,rep(c(0,1),4),1,1,0,0,1,1,0,1)
# fit the model
fmod2 <- fit(mod2,fixed=!free)

# likelihood ratio insignificant, hence fmod2 better than fmod1
llratio(fmod1,fmod2)

# NOW ADD SOME GENERAL LINEAR CONSTRAINTS

# set the starting values of this model to the optimized
# values of the previously fitted model to speed optimization

pars <- c(unlist(getpars(fmod2)))
mod3 <- setpars(mod2,pars)

# start with fixed and free parameters

```

```

compat <- c(0,0,rep(c(0,1),4),1,1,0,0,1,1,0,1)
# constrain the beta's on the transition parameters to be equal
compat[4] <- compat[8] <- 2
compat[6] <- compat[10] <- 3

fmod3 <- fit(mod3,equal=compat)

llratio(fmod2,fmod3)

# above constraints can also be specified using the conrows argument as follows
conr <- matrix(0,2,18)
# pars 4 and 8 have to be equal, otherwise stated, their diffence should be zero
conr[1,4] <- 1
conr[1,8] <- -1
conr[2,6] <- 1
conr[2,10] <- -1

# note here that we use the fitted model fmod2 as that has appropriate
# starting values
fmod3b <- fit(fmod2,conrows=conr,fixed=!free) # using free defined above

## End(Not run)

data(balance)
# four binary items on the balance scale task

instart=c(0.5,0.5)
set.seed(1)
respstart=runif(16)
# note that ntimes argument is used to make this a mixture model
mod4 <- mix(list(d1~1,d2~1,d3~1,d4~1), data=balance, nstates=2,
               family=list(multinomial(),multinomial(),multinomial(),multinomial()),
               respstart=respstart,instart=instart)

fmod4 <- fit(mod4)

# add age as covariate on class membership by using the prior argument
instart=c(0.5,0.5,0,0) # we need the initial probs and the coefficients of age
set.seed(2)
respstart=runif(16)
mod5 <- mix(list(d1~1,d2~1,d3~1,d4~1), data=balance, nstates=2,
               family=list(multinomial(),multinomial(),multinomial(),multinomial()),
               instart=instart, respstart=respstart, prior=~age, initdata=balance)

fmod5 <- fit(mod5)

# check the likelihood ratio; adding age significantly improves the goodness-of-fit
llratio(fmod5,fmod4)

```

**Description**

Compute the forward and backward variables of a depmix object.

**Usage**

```
## S4 method for signature 'depmix':
  forwardbackward(object, return.all=TRUE, ...)
```

**Arguments**

<code>object</code>	A depmix object.
<code>return.all</code>	If FALSE, only gamma and xi and the log likelihood are returned (which are the only variables needed in using EM).
<code>...</code>	Not currently used.

**Value**

`forwardbackward` returns a list of the following (the variables are named after the notation from Rabiner, 1989):

<code>alpha</code>	The forward variables.
<code>beta</code>	The backward variables.
<code>gamma</code>	The smoothed transition probabilities.
<code>xi</code>	The smoothed state probabilities.
<code>sca</code>	The scale factors (called lambda in Rabiner).
<code>logLike</code>	The log likelihood (which is $\sum(\log(sca))$ ).

If `return.all=FALSE`, only gamma, xi and the log likelihood are returned.

**Author(s)**

Maarten Speekenbrink & Ingmar Visser

**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.

**Examples**

```
# add some later
```

---

llratio	<i>Log likelihood ratio test on two fitted models</i>
---------	---

---

**Description**

Performs a log likelihood ratio test on two fitted `depmix` models.

**Usage**

```
llratio(basemodel, constrainedmodel, ...)
```

**Arguments**

<code>basemodel</code>	Fitted model with a <code>logLik</code> method.
<code>constrainedmodel</code>	Fitted model with a <code>logLik</code> method.
<code>...</code>	Not currently used.

**Value**

`llratio` returns an object of class `llratio` which has slots:

<code>value</code>	: Minus twice the loglikelihood difference.
<code>df</code>	: The degrees of freedom, ie the difference in number of freely estimated parameters between the models.

The print method shows the value, the degrees of freedom and the corresponding p-value under the chisquared distribution.

**Author(s)**

Ingmar Visser

---

<code>mix</code>	<i>Mixture Model Specification</i>
------------------	------------------------------------

---

**Description**

`mix` creates an object of class `mix`, an (independent) mixture model (as a limit case of dependent mixture models in which all observed time series are of length 1), otherwise known latent class or mixture model. For a short description of the package see [depmixS4](#).

**Usage**

```
mix(response, data=NULL, nstates, family=gaussian(),
      prior=~1, initdata=NULL, respstart=NULL, instart=NULL,...)
```

**Arguments**

<code>response</code>	The response to be modeled; either a formula or a list of formulae in the multivariate case; this interfaces to the glm distributions. See 'Details'.
<code>data</code>	An optional data.frame to interpret the variables in the response and transition arguments.
<code>nstates</code>	The number of states of the model.
<code>family</code>	A family argument for the response. This must be a list of family's if the response is multivariate.
<code>prior</code>	A one-sided formula specifying the density for the prior or initial state probabilities.
<code>initdata</code>	An optional data.frame to interpret the variables occurring in prior. The number of rows of this data.frame must be equal to the number of cases being modeled. See 'Details'.
<code>respstart</code>	Starting values for the parameters of the response models.
<code>instart</code>	Starting values for the parameters of the prior or initial state probability model.
<code>...</code>	Not used currently.

**Details**

The function `mix` creates an S4 object of class `mix`, which needs to be fitted using `fit` to optimize the parameters.

The response model(s) are created by call(s) to `response` providing the response formula and the family specifying the error distribution. If response is a list of formulae, the response's are assumed to be independent conditional on the latent class.

The prior density is modeled as a multinomial logistic. This model is created by a call to `transInit`.

Starting values may be provided by the respective arguments. The order in which parameters must be provided can be easily studied by using the `setpars` function.

Linear constraints on parameters can be provided as argument to the `fit` function.

The print function prints the formulae for the response and prior models along with their parameter values.

**Value**

`mix` returns an object of class `mix` which has the following slots:

<code>response</code>	A list of a list of response models; the first index runs over states; the second index runs over the independent responses in case a multivariate response is provided.
-----------------------	--

prior	A multinomial logistic model for the initial state probabilities.
dens,init	See <a href="#">mix-class</a> help for details. For internal use.
ntimes	A vector made by <code>rep(1, nrow(data))</code> ; for internal use only.
nstates	The number of states of the model.
nresp	The number of independent responses.
npars	The total number of parameters of the model. Note: this is <i>not</i> the degrees of freedom because there are redundancies in the parameters, in particular in the multinomial models for the transitions and prior probabilities.

**Author(s)**

Ingmar Visser

**References**

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

**See Also**

[fit](#), [transInit](#), [response](#), [depmix-methods](#) for accessor functions to depmix objects.

**Examples**

```
# four binary items on the balance scale task
data(balance)

# define a latent class model
instart=c(0.5,0.5)
set.seed(1)
respstart=runif(16)
# note that ntimes argument is used to make this a mixture model
mod <- mix(list(d1~1,d2~1,d3~1,d4~1), data=balance, nstates=2,
              family=list(multinomial(),multinomial(),multinomial(),multinomial()),
              respstart=respstart,instart=instart)
# to see the model
mod
```

---

mix-class

*Class "mix"*

---

**Description**

A [mix](#) model.

## Objects from the Class

Objects can be created by calls to `mix`.

## Slots

**response:** List of list of `response` objects.

**prior:** `transInit` object; model for the prior probabilities, also unconditional probabilities

**dens:** Array of dimension `sum(ntimes)*nresp*nstates` providing the densities of the observed responses for each state.

**init:** Array of dimension `length(ntimes)*nstates` with the current predictions for the initial state probabilities.

**nstates:** The number of states (classes) of the model.

**nresp:** The number of independent responses.

**ntimes:** A vector of 1's for each case; for internal use.

**npars:** The total number of parameters of the model. This is not the degrees of freedom, ie there are redundancies in the parameters, in particular in the multinomial models for the transitions and prior.

## Accessor Functions

The following functions should be used for accessing the corresponding slots:

**npars:** The number of parameters of the model.

**nresp:** The number of responses.

**nstates:** The number of states.

**ntimes:** The vector of independent time series lengths.

## Author(s)

Ingmar Visser

## Examples

```
showClass("mix")
```

---

mix.fitted-class    *Class "mix.fitted"*

---

## Description

A fitted `mix` model.

## Slots

A `mix.fitted` object is a `mix` object with three additional slots, here is the complete list:

**response:** List of list of response objects.

**prior:** `transInit` object.

**dens:** Array of dimension `sum(ntimes)*nresp*nstates` providing the densities of the observed responses for each state.

**init:** Array of dimension `length(ntimes)*nstates` with the current predictions for the initial state probabilities.

**ntimes:** A vector containing the lengths of independent time series; if data is provided, `sum(ntimes)` must be equal to `nrow(data)`.

**nstates:** The number of states of the model.

**nresp:** The number of independent responses.

**npars:** The total number of parameters of the model. This is not the degrees of freedom, ie there are redundancies in the parameters, in particular in the multinomial models for the transitions and prior.

**message:** This provides some information on convergence, either from the EM algorithm or from `Rdonlp2`.

**conMat:** The linear constraint matrix, which has zero rows if there were no constraints.

**lin.lower** The lower bounds on the linear constraints.

**lin.upper** The upper bounds on the linear constraints.

**posterior:** Posterior (Viterbi) state sequence (not implemented currently).

## Details

The print function shows some convergence information, and the summary method shows the parameter estimates.

## Extends

Class "`mix`", directly.

## Author(s)

Ingmar Visser

**Examples**

```
showClass("mix.fitted")
```

---

```
mix.sim-class      Class "mix.sim"
```

---

**Description**

A `mix.sim` model. The `mix.sim` class directly extends the `mix` class, and has an additional slot for the true states. A `mix.sim` model can be generated by `simulate(mod, ...)`, where `mod` is a `mix` model.

**Slots**

- response:** List of list of response objects.
- prior:** `transInit` object.
- dens:** Array of dimension `sum(ntimes)*nresp*nstates` providing the densities of the observed responses for each state.
- init:** Array of dimension `length(ntimes)*nstates` with the current predictions for the initial state probabilities.
- ntimes:** A vector containing the lengths of independent time series; not applicable for `mix` objects, i.e. this is a vector of 1's.
- nstates:** The number of states/classes of the model.
- nresp:** The number of independent responses.
- npars:** The total number of parameters of the model. This is not the degrees of freedom, ie there are redundancies in the parameters, in particular in the multinomial models for the transitions and prior.
- states:** A matrix with the true states/classes.

**Accessor Functions**

The following functions should be used for accessing the corresponding slots:

- npar:** The number of parameters of the model.
- nresp:** The number of responses.
- nstates:** The number of states.
- ntimes:** The vector of independent time series lengths.

**Author(s)**

Ingmar Visser

---

posterior	<i>Posterior states/classes</i>
-----------	---------------------------------

---

### Description

Return the posterior states for a fitted (dep-)mix object. In the case of a latent class or mixture model these are the class probabilities.

### Usage

```
## S4 method for signature 'depmix.fitted':
  posterior(object)
## S4 method for signature 'mix.fitted':
  posterior(object)
```

### Arguments

object      A (dep-)mix object.

### Value

posterior      : Returns a data.frame with `nstates(object) + 1` columns; the first column has the viterbi states, the other columns have the delta probabilities, see Rabiner (1989).

### Author(s)

Ingmar Visser

### 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.

### Examples

```
# add some later
```

---

 response
 

---



---

*Methods for creating depmix response models*


---

**Description**

Create response objects for `depmix` models using formulae and family objects.

**Usage**

```
GLMresponse(formula, data=NULL, family=gaussian(), pstart=NULL,
             fixed=NULL, prob=TRUE, ...)

transInit(formula, nstates, data=NULL, family=multinomial(),
           pstart=NULL, fixed=NULL, prob=TRUE, ...)

## S4 method for signature 'response':
getdf(object)
```

**Arguments**

<code>formula</code>	A model <a href="#">formula</a> .
<code>data</code>	An optional <code>data.frame</code> to interpret the variables from the formula argument in.
<code>family</code>	A family object; see the <a href="#">responses</a> help page a list of currently available options.
<code>pstart</code>	Starting values for the coefficients and other parameters, ie the standard deviation for the <code>gaussian()</code> family.
<code>fixed</code>	Logical vector indicating which paramters are to be fixed.
<code>prob</code>	Logical indicating whether the starting values for <code>multinomial()</code> family models are probabilities or logistic parameters (see details).
<code>nstates</code>	The number of states of the model.
<code>object</code>	Object of class response.
<code>...</code>	Not used currently.

**Details**

Both methods use the familiar formula interface from `glm` to specify how responses (or transition or prior parameters) depend on covariates/predictors.

The `GLMresponse` model is an interface to the `glm` functions of which the functionality is leant: `predict`, `fit` and `density` functions.

The `transInit` response model provides functionality for multinomial responses that are currently fit using `nnet` (this may change in the future but this should not affect the interface of this function). Note that the `transInit` model actually lacks a response, ie the `y`-slot is empty, at the time of construction, as the transitions are not observed.

**Value**

GLMresponse and transInit return objects of class GLMresponse and transInit respectively; both classes extend the [response-class](#).

getdf returns the number of free parameters of a response model.

**Author(s)**

Ingmar Visser & Maarten Speekenbrink

---

response-class      *Class "response"*

---

**Description**

A generic [response](#) model for [depmix](#) models.

**Arguments**

`object`      An object of class "response".

**Details**

This class offers a framework from which to build specific response models such as glm based responses or multinomial responses. For extensibility, objects with class `response` should have at least methods: `dens` to return the dens'ity of responses, and `getpars` and `setpars` methods to get and set parameters.

**Slots**

**parameters:** A (named) list of parameters.

**fixed:** A logical vector indicating which parameters are fixed.

**y:** A matrix with the actual response; possibly multivariate.

**x:** A design matrix; possibly only an intercept term.

**npars:** The number of parameters.

**Accessor Functions**

The following functions should be used for accessing the corresponding slots:

**npars:** The number of parameters of the model.

**getdf:** The number of non-fixed parameters.

**Author(s)**

Ingmar Visser & Maarten Speekenbrink

---

response-classes    *Class "GLMresponse" and class "transInit"*

---

## Description

Specific instances of response models for `depmix` models.

## Details

The `GLMresponse`-class offers an interface to the `glm` functions that are subsequently used in fitting the `depmix` model of which the response is a part.

The `transInit` is an extension of `response` that is used to model the transition matrix and the initial state probabilities by the use of a multinomial logistic model, the difference being that in fact the response is missing as the transitions between states are not observed. This class has its own fit function which is an interface to the `multinom` function in `nnet`.

## Slots

Both `GLMresponse` and `transInit` contain the `response`-class. In addition to the slots of that class, these classes have the following slots:

**formula:** A formula that specifies the model.

**family:** A family object specifying the link function. Currently, the only options are `gaussian()` from the `stats`-package and `multinomial`.

## Accessor Functions

The following functions should be used for accessing the corresponding slots:

**npar:** The number of parameters of the model.

**getdf:** The number of non-fixed parameters.

## Author(s)

Ingmar Visser & Maarten Speekenbrink

---

 responses
 

---

*Response models currently implemented in depmix.*


---

## Description

Depmix contains a number of default response models. We provide a brief description of these here.

### BINOMresponse

`BINOMresponse` is a binomial response model. It derives from the basic `GLMresponse` class.

**y:** The dependent variable can be either a binary vector, a factor, or a 2-column matrix, with successes and misses.

**x:** The design matrix.

**Parameters:** A named list with a single element “coefficients”, which contains the GLM coefficients.

### GAMMAresponse

`GAMMAresponse` is a model for a Gamma distributed response. It extends the basic `GLMresponse` class directly.

**y:** The dependent variable.

**x:** The design matrix.

**Parameters:** A named list with a single element “coefficients”, which contains the GLM coefficients.

### MULTINOMresponse

`MULTINOMresponse` is a model for a multinomial distributed response. It extends the basic `GLMresponse` class, although the functionality is somewhat different from other models that do so.

**y:** The dependent variable. This is a binary matrix with  $N$  rows and  $Y$  columns, where  $Y$  is the total number of categories.

**x:** The design matrix.

**Parameters:** A named list with a single element “coefficients”, which is an  $\text{ncol}(x)$  by  $\text{ncol}(y)$  matrix which contains the GLM coefficients.

### MVNresponse

`MVNresponse` is a model for a multivariate normal distributed response.

**y:** The dependent variable. This is a matrix.

**x:** The design matrix.

**Parameters:** A named list with a elements “coefficients”, which contains the GLM coefficients, and “Sigma”, which contains the covariance matrix.

**NORMresponse**

NORMresponse is a model for a normal (Gaussian) distributed response. It extends the basic [GLMresponse](#) class directly.

**y:** The dependent variable.

**x:** The design matrix.

**Parameters:** A named list with elements “coefficients”, which contains the GLM coefficients, and “sd”, which contains the standard deviation.

**POISSONresponse**

POISSONresponse is a model for a Poisson distributed response. It extends the basic [GLMresponse](#) class directly.

**y:** The dependent variable.

**x:** The design matrix.

**Parameters:** A named list with a single element “coefficients”, which contains the GLM coefficients.

**Author(s)**

Maarten Speekenbrink & Ingmar Visser

**Examples**

```
# binomial response model
x <- rnorm(1000)
library(boot)
p <- inv.logit(x)
ss <- rbinom(1000,1,p)
mod <- GLMresponse(cbind(ss,1-ss)~x, family=binomial())
fit(mod)
glm(cbind(ss,1-ss)~x, family=binomial)

# gamma response model
x=runif(1000,1,5)
res <- rgamma(1000,x)
## note that gamma needs proper starting values which are not
## provided by depmixS4 (even with them, this may produce warnings)
mod <- GLMresponse(res~x, family=Gamma(), pstart=c(0.8,1/0.8))
fit(mod)
glm(res~x, family=Gamma)

# multinomial response model
x <- sample(0:1,1000,rep=TRUE)
mod <- GLMresponse(sample(1:3,1000,rep=TRUE)~x, family=multinomial(), pstart=c(0.33,0.
mod@y <- simulate(mod)
fit(mod)
colSums(mod@y[which(x==0),])/length(which(x==0))
```

```

colSums(mod@y[which(x==1),])/length(which(x==1))

# multivariate normal response model
mn <- c(1,2,3)
sig <- matrix(c(1,.5,0,.5,1,0,0,0,2),3,3)
y <- mvrnorm(1000,mn,sig)
mod <- MVNresponse(y~1)
fit(mod)
colMeans(y)
var(y)

# normal (gaussian) response model
y <- rnorm(1000)
mod <- GLMresponse(y~1)
fm <- fit(mod)
cat("Test gaussian fit: ", all.equal(getpars(fm),c(mean(y),sd(y)),check=FALSE))

# poisson response model
x <- abs(rnorm(1000,2))
res <- rpois(1000,x)
mod <- GLMresponse(res~x,family=poisson())
fit(mod)
glm(res~x, family=poisson)

```

---

simulate

*Methods to simulate from (dep-)mix models*


---

## Description

Random draws from (dep-)mix objects.

## Usage

```

## S4 method for signature 'depmix':
simulate(object, nsim=1, seed=NULL, ...)

## S4 method for signature 'mix':
simulate(object, nsim=1, seed=NULL, ...)

## S4 method for signature 'response':
simulate(object, nsim=1, seed=NULL, times, ...)

## S4 method for signature 'GLMresponse':
simulate(object, nsim=1, seed=NULL, times, ...)

## S4 method for signature 'transInit':
simulate(object, nsim=1, seed=NULL, times, is.prior=FALSE, ...)

```

**Arguments**

<code>object</code>	Object to generate random draws. An object of class <code>mix</code> , <code>depmix</code> , <code>response</code> or <code>transInit</code>
<code>nsim</code>	The number of draws (one draw simulates a data set of the size that is defined by <code>ntimes</code> ); defaults to 1.
<code>seed</code>	Set the seed.
<code>times</code>	(optional) An indicator vector indicating for which times in the complete series to generate the data. For internal use.
<code>is.prior</code>	For <code>transInit</code> objects, indicates whether it is a prior ( <code>init</code> ) model, or not (i.e., it is a transition model)
<code>...</code>	Not used currently.

**Details**

For a `depmix` model, `simulate` generates `nsim` random state sequences, each of the same length as the observation sequence in the `depmix` model (i.e., `sum(ntimes(object))`). The state sequences are then used to generate `nsim` observation sequences of the same length.

For a `mix` model, `simulate` generates `nsim` random class assignments for each case. Those assignments are then used to generate observation/response values from the appropriate distributions.

Setting the `times` option selects the time points in the total state/observation sequence (i.e., counting is continued over `ntimes`). Direct calls of `simulate` with the `times` option are not recommended.

**Value**

For a `depmix` object, a new object of class `depmix.sim`.

For a `transInit` object, a state sequence.

For a `response` object, an observation sequence.

**Author(s)**

Maarten Speekenbrink

**Examples**

```

y <- rnorm(1000)
respst <- c(0,1,2,1)
trst <- c(0.9,0.1,0.1,0.9)

df <- data.frame(y=y)

mod <- depmix(y~1,data=df,respst=respst,trst=trst,inst=c(0.5,0.5),nti=1000,nst=2)

mod <- simulate(mod)

```

---

`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 from a single subject from experiment 2 in *Van der Maas et al, 2005*.

**Usage**

```
data(speed)
```

**Format**

A data frame with 439 observations on the following 3 variables.

**rt** a numeric vector

**corr** a numeric vector

**Pacc** a numeric vector

**Source**

Han L. J. Van der Maas, Conor V. Dolan and Peter C. M. Molenaar (2007), Phase Transitions in the Trade-Off between Speed and Accuracy in Choice Reaction Time Tasks. *Manuscript in preparation*.

**Examples**

```
data(speed)
## maybe str(speed) ; plot(speed) ...
```

# Index

## \*Topic **classes**

- depmix-class, 9
- depmix.fitted-class, 12
- depmix.sim-class, 13
- mix-class, 21
- mix.fitted-class, 23
- mix.sim-class, 24
- response-class, 27

## \*Topic **datasets**

- balance, 4
- speed, 33

## \*Topic **htest**

- AIC, 3

## \*Topic **methods**

- depmix, 5, 7
- depmix-methods, 10
- fit, 14
- forwardbackward, 18
- llratio, 19
- mix, 19
- posterior, 25
- response, 26
- response-classes, 28
- simulate, 31

## \*Topic **models**

- responses, 29

## \*Topic **package**

- depmixS4-package, 2

AIC, 3

AIC, depmix-method (AIC), 3

AIC, mix-method (AIC), 3

balance, 4

BIC (AIC), 3

BIC, depmix-method (AIC), 3

BIC, mix-method (AIC), 3

BINOMresponse (responses), 29

depmix, 2, 5, 7, 7–9, 12–14, 26–28

depmix, ANY-method (depmix), 5

depmix-class, 6

depmix-methods, 7, 8, 21

depmix-class, 9

depmix-methods, 10

depmix.fit (fit), 14

depmix.fitted, 15

depmix.fitted

(depmix.fitted-class), 12

depmix.fitted-class, 12

depmix.sim (depmix.sim-class), 13

depmix.sim-class, 13

depmixS4, 5, 19

depmixS4 (depmixS4-package), 2

depmixS4-package, 2

fit, 2, 6–8, 14, 20, 21

fit, depmix-method (fit), 14

fit, mix-method (fit), 14

formula, 26

forwardbackward, 11, 17

forwardbackward, depmix-method  
(forwardbackward), 18

freepars, 15

freepars (depmix-methods), 10

freepars, depmix-method

(depmix-methods), 10

freepars, depmix.fitted-method

(depmix-methods), 10

freepars, mix-method

(depmix-methods), 10

freepars, mix.fitted-method

(depmix-methods), 10

GAMMAresponse (responses), 29

getdf (response), 26

getdf, response-method (response),  
26

getpars, 15

getpars (depmix-methods), 10

- getpars, depmix-method  
(*depmix-methods*), 10
- getpars, mix-method  
(*depmix-methods*), 10
- glm, 2, 26, 28
- GLMresponse, 29, 30
- GLMresponse (*response*), 26
- GLMresponse, formula-method  
(*response*), 26
- GLMresponse-class  
(*response-classes*), 28
  
- llratio, 15, 19
- logLik (*depmix-methods*), 10
- logLik, depmix-method  
(*depmix-methods*), 10
- logLik, mix-method  
(*depmix-methods*), 10
- loglikelihoodratio (*llratio*), 19
  
- makeDepmix (*depmix*), 7
- mix, 2, 14, 19, 21–24
- mix, ANY-method (*mix*), 19
- mix-class, 21
- mix-class, 21
- mix.fit (*fit*), 14
- mix.fitted-class, 23
- mix.sim (*mix.sim-class*), 24
- mix.sim-class, 24
- MULTINOMresponse (*responses*), 29
- MVNresponse (*responses*), 29
- MVNresponse-class  
(*response-classes*), 28
  
- nnet, 28
- nobs (*depmix-methods*), 10
- nobs, depmix-method  
(*depmix-methods*), 10
- nobs, mix-method (*depmix-methods*),  
10
- NORMresponse (*responses*), 29
- npar (*depmix-methods*), 10
- npar, depmix-method  
(*depmix-methods*), 10
- npar, mix-method (*depmix-methods*),  
10
- nresp (*depmix-class*), 9
- nresp, depmix-method  
(*depmix-class*), 9
  
- nresp, depmix.sim-method  
(*depmix.sim-class*), 13
- nresp, mix-method (*mix-class*), 21
- nresp, mix.sim-method  
(*mix.sim-class*), 24
- nstates (*depmix-class*), 9
- nstates, depmix-method  
(*depmix-class*), 9
- nstates, depmix.sim-method  
(*depmix.sim-class*), 13
- nstates, mix-method (*mix-class*), 21
- nstates, mix.sim-method  
(*mix.sim-class*), 24
- ntimes (*depmix-class*), 9
- ntimes, depmix-method  
(*depmix-class*), 9
- ntimes, depmix.sim-method  
(*depmix.sim-class*), 13
- ntimes, mix-method (*mix-class*), 21
- ntimes, mix.sim-method  
(*mix.sim-class*), 24
  
- POISSONresponse (*responses*), 29
- posterior, 15, 25
- posterior, depmix.fitted-method  
(*posterior*), 25
- posterior, mix.fitted-method  
(*posterior*), 25
  
- response, 6–8, 20, 21, 26, 27
- response-class, 27
- response-class, 27
- response-classes, 28
- responses, 26, 29
  
- setpars, 6, 15, 20
- setpars (*depmix-methods*), 10
- setpars, depmix-method  
(*depmix-methods*), 10
- setpars, mix-method  
(*depmix-methods*), 10
- show (*depmix*), 5
- show, depmix-method (*depmix*), 5
- show, depmix.fitted-method (*fit*),  
14
- show, GLMresponse-method  
(*response*), 26
- show, llratio-method (*llratio*), 19
- show, mix-method (*mix*), 19

show, `mix.fitted-method` (*fit*), 14  
show, `MVNresponse-method`  
    (*responses*), 29  
simulate, 31  
simulate, `BINOMresponse-method`  
    (*simulate*), 31  
simulate, `depmix-method`  
    (*simulate*), 31  
simulate, `GAMMAresponse-method`  
    (*simulate*), 31  
simulate, `GLMresponse-method`  
    (*simulate*), 31  
simulate, `mix-method` (*simulate*), 31  
simulate, `MULTINOMresponse-method`  
    (*simulate*), 31  
simulate, `MVNresponse-method`  
    (*simulate*), 31  
simulate, `NORMresponse-method`  
    (*simulate*), 31  
simulate, `POISSONresponse-method`  
    (*simulate*), 31  
simulate, `response-method`  
    (*simulate*), 31  
simulate, `transInit-method`  
    (*simulate*), 31  
speed, 33  
stats, 28  
summary (*depmix*), 5  
summary, `depmix-method` (*depmix*), 5  
summary, `depmix.fitted-method`  
    (*fit*), 14  
summary, `mix-method` (*mix*), 19  
summary, `mix.fitted-method` (*fit*),  
    14  
  
transInit, 6–9, 13, 20–22, 24  
transInit (*response*), 26  
transInit, `formula-method`  
    (*response*), 26  
transInit-class  
    (*response-classes*), 28