Journal of Statistical Software

MMMMMM YYYY, Volume VV, Issue II. *http://www.jstatsoft.org/*

depmixS4 : An R-package for hidden Markov models

Ingmar Visser
University of Amsterdam

Maarten Speekenbrink
University College London

Abstract

depmixS4 implements a general framework for defininining and estimating hidden Markov mixture model in the R programming language (R Development Core Team 2009). This includes standard Markov models, latent/hidden Markov models, and latent class and finite mixture distribution models. The models can be fitted on mixed multivariate data with multinomial and/or gaussian distributions. Parameters can be estimated subject to general linear constraints. Parameter estimation is done through an EM algorithm or by a direct optimization approach using the **Rdonlp2** optimization routine when contraints are imposed on the parameters. A number of illustrative examples are included.

Keywords: hidden Markov model, dependent mixture model, mixture model.

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, Visser, and Raijmakers (2006) for a recent application. In economics, latent Markov models are commonly used as regime switching models, see e.g. Kim (1994) and Ghysels (1994). Further applications include speech recognition (Rabiner 1989), EEG analysis (Rainer and Miller 2000), and genetics (Krogh 1998). In those latter areas of application, latent Markov models are usually referred to as hidden Markov models.

The **depmixS4** 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, Langeheine, and Jong 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. There are currently some packages in R that handle hidden Markov models but they lack a number

of features that we needed in our research. In particular, **depmixS4** was designed to meet the following goals:

1. to be able to handle parameter estimates subject to general linear (in)equality constraints
2. to be able to fit transition models with covariates, i.e., to have time-dependent transition matrices
3. to be able to include covariates in the prior or initial state probabilities of models
4. to allow for easy extensibility, in particular, to be able to add new response distributions, both univariate and multivariate, and similarly to be able to allow for the addition of other transition models, e.g., continuous time observation models

Although **depmixS4** is designed to deal 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. Although there are other specialized packages to deal with mixture data, one specific feature that we needed ourselves which is to the best of our knowledge not available in other packages is the possibility to include covariates on the prior probabilities of class membership. In the next section, an outline is provided of the model and the likelihood equations.

2. The dependent mixture model

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 on 168, 134 and 137 occasions respectively (in Figure 1 the first part of this series is plotted).

The latent Markov model is commonly associated with data of this type, although 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 sections, the likelihood and estimation procedure for the hidden Markov model is described, given data of the above form. These models are called dependent mixture models because one of the authors (Ingmar Visser) thought it was time for a new name for these models[1]

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. matrices $\mathbf{A}_t$ of transition probabilities $a_{ij,t}$ for the transition from state S_i to state S_j at time t,

[1]Only later did I find out that Leroux and Puterman (1992) already coined the term dependent mixture models in an application with hidden Markov mixtures of Poisson count data.

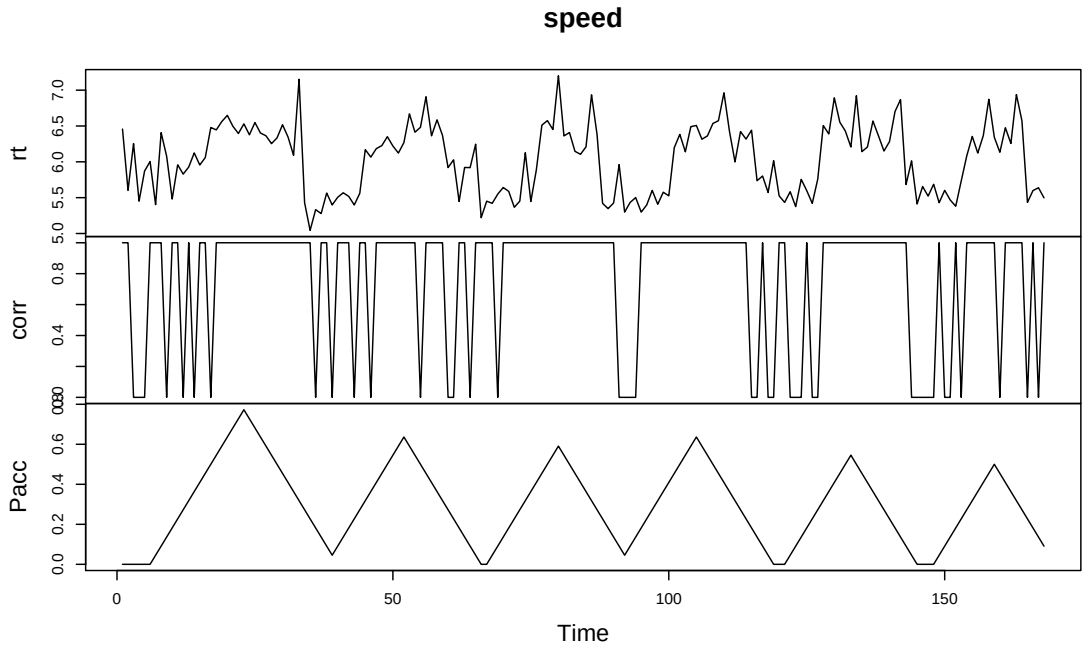

Figure 1: Response times, accuracy and pay-off values for the first series of responses in dataset `speed`.

3. a set $\mathbf{B}_t$ of observation functions $b_j^k(\cdot)$ that provide the conditional probabilities of observations O_t^k associated with latent state S_j,
4. a vector $\boldsymbol{\pi}$ of latent state initial probabilities π_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.

The dependent mixture model is then given by the following equations:

$$S_t = AS_{t-1}, t = 2, \ldots, T \tag{1}$$

$$O_t = f(O_t|S_t), \tag{2}$$

where S_t is a sequence of hidden states, A is a transition matrix, O_t is an (possibly multivariate) observation and f is a density function for O_t conditional on the hidden state S_t. In the example data above, f could be a gaussian distribution function for the response time variable, and a Bernouilli for the accuracy data. In the models we are considering here, both the transition probabilities A and the initial state probabilities π may depend on covariates as well as the response distributions f. See for example Frühwirth-Schnatter (2006) for an overview of hidden Markov models with extensions.

2.1. Likelihood

The log-likelihood 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 log-likelihood 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):

$$L_T = Pr(\mathbf{O}_1, \ldots, \mathbf{O}_T) = \prod_{t=1}^{T} Pr(\mathbf{O}_t | \mathbf{O}_1, \ldots, \mathbf{O}_{t-1}), \tag{3}$$

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:

$$l_T = \sum_{t=1}^{T} \log[Pr(\mathbf{O}_t | \mathbf{O}_1, \ldots, \mathbf{O}_{t-1})]. \tag{4}$$

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

$$\phi_1(j) := Pr(\mathbf{O}_1, S_1 = j) = \pi_j b_j(\mathbf{O}_1) \tag{5}$$

$$\begin{aligned} \phi_t(j) &:= Pr(\mathbf{O}_t, S_t = j | \mathbf{O}_1, \ldots, \mathbf{O}_{t-1}) \\ &= \sum_{i=1}^{N} [\phi_{t-1}(i) a_{ij} b_j(\mathbf{O}_t)] \times (\Phi_{t-1})^{-1}, \end{aligned} \tag{6}$$

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 (4) gives the following expression for the log-likelihood:

$$l_T = \sum_{t=1}^{T} \log \Phi_t. \tag{7}$$

2.2. Parameter estimation

Parameters are estimated in **depmixS4** using the EM algorithm or through the use of a general Newton-Raphson optimizer.

In the EM algorithm, parameters are estimated by iteratively maximising the expected joint likelihood of the parameters given the observations and states. Let $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, \boldsymbol{\theta}_3)$ be the general parameter vector consisting of three subvectors with parameters for the prior model, transition model, and response model respectively. The joint log likelihood can be written as

$$\log Pr(O_{1:T}, S_{1:T}|\boldsymbol{\theta}) = \log Pr(S_1|\boldsymbol{\theta}_1) + \sum_{t=2}^{T} \log Pr(S_t|S_{t-1}, \boldsymbol{\theta}_2) + \sum_{t=1}^{T} \log Pr(O_t|S_t, \boldsymbol{\theta}_3) \tag{8}$$

This likelihood depends on the unobserved states S_t. In the Expectation step, we replace these with their expected values given a set of (initial) parameters $\boldsymbol{\theta}' = (\boldsymbol{\theta}'_1, \boldsymbol{\theta}'_2, \boldsymbol{\theta}'_3)$ and observations $O_{1:T}$. The expected log likelihood

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}') = E_{\boldsymbol{\theta}'}(\log Pr(O_{1:T}, S_{1:T}|O_{1:T}, \boldsymbol{\theta})) \tag{9}$$

can be written as

$$Q(\boldsymbol{\theta}, \boldsymbol{\theta}') = \sum_{j=1}^{n} \gamma_1(j) \log Pr(S_1 = j|\boldsymbol{\theta}_1) + \sum_{t=2}^{T} \sum_{j=1}^{M} \sum_{k=1}^{n} \xi_t^i(j,k) \log Pr(S_t = k|S_{t-1} = j, \boldsymbol{\theta}_2) + \sum_{t=1}^{T} \sum_{j=1}^{n} \gamma_t^i(j) \ln Pr(O_t|S_t = j, \boldsymbol{\theta}_3), \quad (10)$$

where the expected values $\xi_t(j,k) = P(S_t = k, S_{t-1} = j|O_{1:T}, \boldsymbol{\theta}')$ and $\gamma_t(j) = P(S_t = j|O_{1:T}, \boldsymbol{\theta}')$ can be computed effectively by the Forward-Backward algorithm (see e.g., Rabiner 1989). The Maximisation step consists of the maximisation of (10) for $\boldsymbol{\theta}$. As the r.h.s. of (10) consists of three separate parts, we can maximise separately for $\boldsymbol{\theta}_1$, $\boldsymbol{\theta}_2$ and $\boldsymbol{\theta}_3$. In common models, maximisation for $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$ is performed by the `nnet.default` routine in **nnet**, and maximisation for $\boldsymbol{\theta}_3$ by the `glm` routine.

The EM algorithm however has some drawbacks. First, it can be slow to converge towards the end of optimization (although it is usually faster than direct optimization at the start, so possibly a combination of EM and direct optimization is fastest). Second, applying constraints to parameters can be problematic; in particular, EM can lead to wrong parameter estimates when applying constraints. Hence, in **depmixS4**, EM is used by default in unconstrained models, but otherwise, direct optimization is done using **Rdonlp2** Tamura (2007); Spellucci (2002), because it handles general linear (in)equality constraints, and optionally also non-linear constraints.

3. Using depmixS4

Two steps are involved in using **depmixS4** which are illustrated below with examples:

1. model specification with function `depmix`
2. model fitting with function `fit`

3.1. Example data: speed

Throughout this manual a data set called `speed` is used. It consists of three time series with three variables: response time, accuracy, and a covariate Pacc which defines the relative pay-off for speeded and accurate responding. The participant in this experiment switches between fast responding at chance level and relatively slower responding at a high level of accuracy. Interesting hypotheses to test are: is the switching regime symmetric? Is there evidence for two states or does one state suffice? Is the guessing state actually a guessing state, i.e., is the probability of a correct response at chance level (0.5)?

3.2. A simple example

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

```
> set.seed(1)
> mod <- depmix(rt~1, data=speed, nstates=2, trstart=runif(4))
```

The `depmix` function returns an object of class `depmix` which contains the model specification (and not a fitted model!). Note also that start values for the transition parameters are provided in this call using the `trstart` argument. The package does not provide automatic starting values.

The so-defined models needs to be `fit`ted with the following:

```
> fm <- fit(mod)
```

The `fit`-function returns an object of class `depmix.fitted` which extends the `depmix` class, adding convergence information (and information about constraints if these were applied, see below). The `print` method provides summary information on convergence, the log likelihood and the AIC and BIC values. These statistics may be extracted using `logLik`, `AIC` and `BIC`, respectively.

```
> fm
```

```
Convergence info: Log likelihood converged to within tol.
'log Lik.' -84.34175 (df=7)
AIC:  182.6835
BIC:  211.275
```

The `summary` method of `fit`ted models provides the parameter estimates, first for the prior probabilities model, second for the transition model, and third for the response models.

```
> summary(fm)
```

```
Initial state probabilties model
Model of type multinomial, formula: ~1
Coefficients:
     [,1]      [,2]
[1,]    0 -11.25688
Probalities at zero values of the covariates.
0.999987 1.291798e-05

Transition model for state (component) 1
Model of type multinomial, formula: ~1
Coefficients:
[1]  0.000000 -2.392455
Probalities at zero values of the covariates.
0.9162501 0.08374986

Transition model for state (component) 2
Model of type multinomial, formula: ~1
```

```
Coefficients:
[1] 0.000000 2.139255
Probalities at zero values of the covariates.
0.1053396 0.8946604

Response model(s) for state 1

Response model for response 1
Model of type gaussian, formula: rt ~ 1
Coefficients:
[1] 6.385492
sd  0.2439376

Response model(s) for state 2

Response model for response 1
Model of type gaussian, formula: rt ~ 1
Coefficients:
[1] 5.511151
sd  0.1926063
```

3.3. Transition parameters

The transition matrix is parametrized as a list of multinomial logistic models. The initial state probabilities are similarly parametrized as a multinomial logistic model. Both models use a baseline category parametrization, meaning that the parameter for the base category is fixed at zero (see Agresti (see 2002, p. 267 ff.) for multinomial logistic models and various parameterizations). The default base category is the first state. Hence, for example, for a 3-state model, the initial state probability model would have three parameters of which the first is fixed at zero and the other two are freely estimated.

Covariates on the transition probabilities can be specified using a one-sided formula as in the following example:

```
> set.seed(1)
> mod <- depmix(rt~1, data=speed, nstates=2, family=gaussian(),
  transition=~scale(Pacc), instart=runif(2))
> fm <- fit(mod)

Initial state probabilties model
Model of type multinomial, formula: ~1
Coefficients:
         [,1]     [,2]
[1,]    0 10.71779
Probalities at zero values of the covariates.
2.214681e-05 0.9999779
```

```
Transition model for state (component) 1
Model of type multinomial, formula: ~scale(Pacc)
Coefficients:
         [,1]       [,2]
[1,]    0 -0.9215182
[2,]    0  1.8649734
Probalities at zero values of the covariates.
0.7153513 0.2846487

Transition model for state (component) 2
Model of type multinomial, formula: ~scale(Pacc)
Coefficients:
         [,1]     [,2]
[1,]    0 2.471442
[2,]    0 3.570856
Probalities at zero values of the covariates.
0.07788458 0.9221154

Response model(s) for state 1

Response model for response 1
Model of type gaussian, formula: rt ~ 1
Coefficients:
[1] 5.512179
sd  0.1921098

Response model(s) for state 2

Response model for response 1
Model of type gaussian, formula: rt ~ 1
Coefficients:
[1] 6.3885
sd  0.2402693
```

The summary provides all parameters of the model, also the (redundant) zeroes for the baseline category in the multinomial model. The summary also prints the transition probabilities at the zero value of the covariate. Note that scaling of the covariate is useful in this regard as it makes interpretation of these intercept probabilities easier.

3.4. Multivariate data

Multivariate data can be modelled by providing a list of formulae as well as a list of family objects for the distributions of the various responses. In above examples we have only used the response times which were modelled with the gaussian distribution. The accuracy data are in the `speed` data are modelled with a multinomial by specifying the following:

```
set.seed(1)
```

```
mod <- depmix(list(rt~1,corr~1), data=speed, nstates=2,
family=list(gaussian(),multinomial()), transition=~scale(Pacc),
instart=runif(2))
fm <- fit(mod)
```

which provides the following fitted model parameters (only the response parameters are given here):

```
> summary(fm)
\ldots
Response model(s) for state 1

Response model for response 1
Model of type gaussian, formula: rt ~ 1
Coefficients:
[1] 5.52169
sd  0.2028857

Response model for response 2
Model of type multinomial, formula: corr ~ 1
Coefficients:
 [,1]      [,2]
[1,]    0 0.1030554
Probalities at zero values of the covariates.
0.4742589 0.5257411

Response model(s) for state 2

Response model for response 1
Model of type gaussian, formula: rt ~ 1
Coefficients:
[1] 6.39369
sd  0.2373650

Response model for response 2
Model of type multinomial, formula: corr ~ 1
Coefficients:
 [,1]     [,2]
[1,]    0 2.245514
Probalities at zero values of the covariates.
0.09573715 0.9042629
```

As can be seen, state 1 has fast response times around chance level, whereas state 2 corresponds with slower responding at higher accuracy levels.

3.5. Adding covariates on the prior probabilities

The **depmixS4** contains another data set which is used to illustrate the use of covariates on

the prior probabilities of models, in this case a latent class model. The `balance` data consists of 4 binary items (correct-incorrect) on a balance scale task Siegler (1981). The data form a subset of the data published in Jansen and van der Maas (2002).

Similarly to the transition matrix, covariates on the prior probabilities of the latent states (or classes in this case), are defined by using a one-sided formula:

```
balance$age <- balance$age-5
set.seed(1)
mod <- mix(list(d1~1,d2~1,d3~1,d4~1), data=balance, nstates=2,
        family=list(multinomial(), multinomial(), multinomial(),
multinomial()), respstart=c(rep(c(0.9,0.1),4),rep(c(0.1,0.9),4)),
prior=~age, initdata=balance)
fm <- fit(mod)
```

Note here that we define a `mix` model instead of a `depmix` models as these data form independent observations.

The summary of the `fit`ted model gives (only the prior model is shown here):

```
> summary(fm)
Mixture probabilities model
Model of type multinomial, formula: ~age
[,1]   [,2]
[1,]    0 -2.5182573
[2,]    0  0.5512996
Probalities at zero values of the covariates.
0.9254119 0.07458815
```

Hence at young ages, children have a high probability of incorrect responding in class 1, whereas the prior probability for class 2 increases with age.

3.6. Fixing and constraining parameters

Using package **Rdonlp2**, parameters may be fitted subject to general linear (in-)equality constraints. Constraining and fixing parameters is done using the `conpat` argument to the `depmix.fit`-function, which specifies for each parameter in the model whether it's fixed (0) or free (1 or higher). Equality constraints can be imposed by having two parameters have the same number in the `conpat` vector. When only fixed values are required the `fixed` argument can be used instead of `conpat`, with zeroes for fixed parameters and other values (ones e.g.) for non-fixed parameters. Fitting the models subject to these constraints is handled by the optimization routine `donlp2`.

Parameter numbering When using the `conpat` and `fixed` arguments, complete parameter vectors should be supplied, i.e., these vectors should have length of the number of parameters of the model, which can be obtained by calling `npar(object)`. Parameters are numbered in the following order:

1. the prior model parameters

2. the parameters for the transition models
3. the response model parameters per state (and subsequently per response in the case of multivariate time series)

To see the ordering of parameters use the following:

```
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
```

When fitting constraints it is useful to have good starting values for the parameters and hence we first fit the following model:

```
trst=c(0.9,0.1,0,0,0.1,0.9,0,0)
mod <- depmix(list(rt~1,corr~1),data=speed,transition=~Pacc,
nstates=2,family=list(gaussian(),multinomial()),
trstart=trst,inst=c(.999,0.001))
fm <- fit(mod)
```

After this, we use the fitted values from this model to constrain the regression coefficients on the transition matrix:

```
# start with fixed and free parameters
conpat <- c(0,1,rep(c(0,1),4),1,1,0,1,1,1,0,1)
# constrain the beta's on the transition parameters to be equal
conpat[6] <- conpat[10] <- 2
fm <- fit(fm,equal=conpat)
```

Using `summary` on the fitted model shows that the regression coefficients are now estimated at the same value of 12.66. Note that these arguments provide the possibility for arbitrary constraints, also between, e.g., a multinomial regression coefficient for the transition matrix and the mean of a gaussian response model. Whether such constraints make sense is hence the responsibility of the user.

4. Extending depmixS4

The **depmixS4** package was built with the aim of having the possibility of adding new response distributions (and possibly also other models for the prior and transition probabilities models). Therefore, the EM algorithm simply calls `fit` functions from the separate response models without knowing what they are. As a consequence, adding user-specified response models is straightforward. User-defined distributions should extend the `response`-class and have the following slots:

1. y: the response variable
2. x: the design matrix, possibly only an intercept
3. paramaters: a named list with the coefficients and possibly other parameters, e.g., the standard deviation in the gaussian response model
4. fixed: a vector of logicals indicating whether parameters are fixed
5. npar: numerical indicating the number of parameters of the model

In the `speed` data example, it may be more appropriate to model the response times as an exgaus distribution rather than using the gaussian. We can do so as follows, first we define our own exgaus-class extending the response-class:

```
setClass("exgaus", contains="response")
```

The so-defined class now needs a number of methods:

1. constructor: function to create instances of the class with starting values
2. show method: to print the model to the terminal
3. dens: the function that computes the density of the responses
4. getpars and setpars: to get and set parameters
5. predict: to generate predict'ed values
6. fit: function to fit the model using posterior weights (used by the EM algorithm)

Only the constructor function and the fit function are provided here and the complete code is provided in the helpfile for `makeDepmix`. The `exgaus` example uses the **gamlss.distr** package for computing the `dens`ity and for fitting the parameters.

The constructor function is defined as:

```
setGeneric("exgaus", function(y, pstart = NULL, fixed = NULL, ...)
standardGeneric("exgaus"))
setMethod("exgaus",
signature(y="ANY"),
function(y,pstart=NULL,fixed=NULL, ...) {
y <- matrix(y,length(y))
x <- matrix(1)
parameters <- list()
npar <- 3
if(is.null(fixed)) fixed <- as.logical(rep(0,npar))
if(!is.null(pstart)) {
if(length(pstart)!=npar) stop("length of 'pstart' must be ",npar)
parameters$mu <- pstart[1]
parameters$sigma <- log(pstart[2])
parameters$nu <- log(pstart[3])
```

```
}
mod <- new("exgaus",parameters=parameters,fixed=fixed,x=x,y=y,npar=npar)
mod
}
)
```

The `fit` function is defined as follows:

```
setMethod("fit","exgaus",
function(object,w) {
if(missing(w)) w <- NULL
y <- object@y
fit <- gamlss(y~1,weights=w,family=exGAUS(),
control=gamlss.control(n.cyc=100,trace=FALSE),
mu.start=object@parameters$mu,
sigma.start=exp(object@parameters$sigma),
nu.start=exp(object@parameters$nu))
pars <- c(fit$mu.coefficients,fit$sigma.coefficients,fit$nu.coefficients)
object <- setpars(object,pars)
object
}
)
```

The `fit` function defines a trivial `gamlss` model with only an intercept to be estimated and then sets the fitted parameters back into their respective slots in the 'exgaus' object. See the help for `gamlss.distr` for interpretation of these parameters.

After defining all the necessary methods for the new response model, we can now define the dependent mixture model using this reponse. The function `makeDepmix` is added to **depmixS4** to have full control over model specification, and we need it here.

We first create all the response models that we want as a double list:

```
rModels <- list(
list(
exgaus(rt,pstart=c(5,.1,.1)),
GLMresponse(formula=corr~1,data=speed,family=multinomial(),pstart=c(0.5,0.5))
),
list(
exgaus(rt,pstart=c(6,.1,.1)),
GLMresponse(formula=corr~1,data=speed,family=multinomial(),pstart=c(.1,.9))
)
)
```

Next, we define the transition and prior probability models using the transInit function (which produces a transInit model, which also extends the response class):

```
trstart=c(0.9,0.1,0.1,0.9)
transition <- list()
```

```
transition[[1]] <- transInit(~Pacc,nstates=2,data=speed,pstart=c(0.9,0.1,0,0))
transition[[2]] <- transInit(~Pacc,nstates=2,data=speed,pstart=c(0.1,0.9,0,0))
inMod <- transInit(~1,ns=2,pstart=c(0.1,0.9),data=data.frame(1))
```

Finally, we put everything together using `makeDepmix` and fit the model:

```
mod <- makeDepmix(response=rModels,transition=transition,
prior=inMod,stat=FALSE)
fm <- fit(mod)
```

Using `summary` will print the fitted parameters. Note that the use of `makeDepmix` allows the possibility of, say, fitting a gaussian in one state and an exgaus distribution in another state.

5. Conclusion & future work

What are our next plans?

Adding gradients for speed and computation of the Hessian.

Acknowledgements

Ingmar Visser was supported by an EC Framework 6 grant, project 516542 (NEST). Maarten Speekenbrink was supported by the ESRC Centre for Economic Learning and Social Evolution (ELSE). Han van der Maas provided the speed-accuracy data Dutilh *et al.* (2009) and thereby neccessitated implementing models with time-dependent covariates. Brenda Jansen provided the balance scale data set (Jansen and van der Maas 2002) which was the perfect opportunity to test the covariates on the prior model parameters. The examples in the help files use both of these data sets.

References

Agresti A (2002). *Categorical Data Analysis*. Wiley series in probability and mathematical statistics. Wiley-Interscience, Hoboken, NJ, 2 edition.

Baum LE, Petrie T (1966). "Statistical inference for probabilistic functions of finite state Markov Chains." *Annals of Mathematical Statistics*, **67**, 1554–40.

Dutilh, *et al.* (2009). "sathmm." *Manuscript in preparation*.

Frühwirth-Schnatter S (2006). *Finite Mixture and Markov Switching Models*. Springer Series in Statistics. Springer.

Ghysels E (1994). "On the Periodic Structure of the Business Cycle." *Journal of Business and Economic Statistics*, **12**(3), 289–298.

Jansen BRJ, van der Maas HLJ (2002). "The development of children's rule use on the balance scale task." *Journal of Experimental Child Psychology*, **81**(4), 383–416.

Kim CJ (1994). "Dynamic linear models with Markov-switching." *Journal of Econometrics*, **60**, 1–22.

Krogh A (1998). "An introduction to hidden Markov models for biological sequences." In SL Salzberg, DB Searls, S Kasif (eds.), "Computational methods in molecular biology," chapter 4, pp. 45–63. Elsevier, Amsterdam.

Leroux BG, Puterman ML (1992). "Maximum-Penalized-Likelihood Estimation for Independent and Markov-Dependent Mixture Models." *Biometrics*, **48**, 545–548.

Lystig TC, Hughes JP (2002). "Exact computation of the observed information matrix for hidden Markov models." *Journal of Computational and Graphical Statistics*.

R Development Core Team (2009). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0, URL http://www.R-project.org.

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

Rainer G, Miller EK (2000). "Neural ensemble states in prefrontal cortex identified using a hidden Markov model with a modified EM algorithm." *Neurocomputing*, **32–33**, 961–966.

Schmittmann VD, Visser I, Raijmakers MEJ (2006). "Multiple learning modes in the development of rule-based category-learning task performance." *Neuropsychologia*, **44**(11), 2079–2091.

Siegler RS (1981). *Developmental sequences within and between concepts*. Number 46 in Monographs of the Society for Research in Child Development. SRCD.

Spellucci P (2002). "DONLP2." URL http://www.netlib.org/ampl/solvers/donlp2/.

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

Van de Pol F, Langeheine R, Jong WD (1996). *PANMARK 3. Panel analysis using Markov chains. A latent class analysis program [User manual]*. Voorburg: The Netherlands.

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

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

Affiliation:

Ingmar Visser
Department of Psychology
University of Amsterdam
Roetersstraat 15
1018 WB, Amsterdam
The Netherlands
E-mail: i.visser@uva.nl
URL: http://www.ingmar.org/

Journal of Statistical Software http://www.jstatsoft.org/
published by the American Statistical Association http://www.amstat.org/
Volume VV, Issue II *Submitted:* yyyy-mm-dd
MMMMMM YYYY *Accepted:* yyyy-mm-dd