Sylvia Frühwirth-Schnatter

# Finite Mixture and Markov Switching Models

## Implementation in MATLAB using the package `bayesf Version 2.0`

December 2, 2008

Springer
Berlin Heidelberg New York
Hong Kong London
Milan Paris Tokyo

# Preface

This package is an update of `Version 1.0` of the MATLAB package `bayesf` released in January 2007. Major changes of `Version 2.0` compared to the previous version are the following:

- Additional distribution families may be selected as component densities in a finite mixture model, namely exponential distributions, univariate and multivariate Student-$t$ distributions with unknown, group specific degrees of freedom and binomial distributions with constant or varying repetition parameter.
- Finite mixtures of generalized linear models have been added and may be based on the Poission distribution, a negative binomial distribution with unknown, group specific degrees of freedom or a binomial distributions with constant or varying repetition parameter.
- For discrete data, it is now possible to allow for the presence of exposures and repeated measurements.

Additionally, some bugs have been fixed. Please be aware that the programs are tested mainly for the data I was interested in. The package may still contain some coding errors and may not work for your data. Please inform me about any problem you have by sending an email to

`sylvia.fruehwirth-schnatter@jku.at`.

Finally, I kindly ask to acknowledge the use of the `bayesf` package if you use results obtained by this package in any research report or in any other means of publication.

Vienna and Linz, December 2008 Sylvia Frühwirth-Schnatter

# Contents

# 1

# Getting Started Quickly

This toolbox has been designed to fit finite mixture models to data using a Bayesian approach based on Markov chain Monte Carlo (MCMC) methods. Such an approach basically has three input parameters, namely the *data*, the *model* and the *prior* and one output parameter, namely the *MCMC draws*. In this package, these parameters are defined through structural arrays.

This chapter shows in Section 1.1 how to get started quickly without bothering too much about prior choices or tuning MCMC. For illustration, Section 1.2 provides six examples, including appropriate MATLAB code. More details appear in later chapters.

## 1.1 Fitting Finite Mixture Distributions

Fitting finite mixture distributions to data using a Bayesian approach basically requires five steps:

1. Defining the model
2. Loading the data
3. Choosing the prior
4. Initializing MCMC
5. Running MCMC

### 1.1.1 Defining the Model

A finite mixture model is a structure array with two obligatory fields: the number of components (field `K`) and the density of the mixture components (field `.dist`). Table 1.1 shows which distribution families are implemented in the current version of this package. The following example defines a mixture of three univariate normal distributions, named `model`:

**Table 1.1.** Mixture distributions implemented in the current version of the package.

| Distribution Family | Abbreviation in field `.dist` |
|---|---|
| univariate normal | `'Normal'` |
| univariate $t$ | `'Student'` |
| multivariate normal | `'Normult'` |
| multivariate $t$ | `'Studmult'` |
| exponential | `'Exponential'` |
| Poisson | `'Poisson'` |
| binomial | `'Binomial'` |

```
model.K=3;
model.dist='Normal';
```

Note that the name `model` may be substituted by any name, e.g. `mymodel`. Chapter 2 describes in more detail how a finite mixture model is defined. Applications to specific distribution families are provided in Chapter 6 for continuous distributions like mixtures of normal, Student-$t$ and exponential distributions and in Chapter 7 for discrete distributions like mixtures of Poisson and binomial distributions.

More general finite mixture models may be fitted to data using this package. Chapter 8 discusses in detail finite mixtures of regression models as well as their extension to mixtures of generalized linear models based on the Poisson, the binomial or the negative binomial distribution. Finally, Chapter 9 discusses finite mixture modeling of time series data using hidden Markov chain models.

### 1.1.2 Loading the Data

The data have to be stored in a structural array named e.g. `data` with one obligatory field, namely `data.y` containing the observations stored by row. More details on data handling appear in Chapter 3. Several data sets are stored under particular names and could be loaded into a structure array using the function `dataget`, see also Subsection 3.1.3. Typing, for instance,

```
data=dataget('fishery');
```

loads the FISHERY DATA plotted in Figure 3.1 into the structure array `data`. Note that the name `data` could be substituted by any name, e.g. `mydata`.

### 1.1.3 Choosing the Prior

The package provides automatic choices of slightly data based proper priors which are explained in detail in the subsequent chapters. Use the function `priordefine` with the model, stored e.g. in `model`, and the data, stored e.g. in `data`, as input arguments to choose this prior:

```
prior = priordefine(data,model);
```

The prior is stored in the structure array `prior`. More details about the fields of this structural array appear in Subsection 4.2.1 and later chapters, however, the package could be run without caring about these details. Note that the name `prior` could be substituted by any name, e.g. `myprior`.

### 1.1.4 Initializing MCMC

Call the function `mcmcstart` with the model, stored e.g. in `model`, and the data, stored e.g. in `data`, as input arguments to initialize MCMC:

```
[data,model,mcmc] = mcmcstart(data,model);
```

This function automatically selects all necessary starting values and tuning parameters for MCMC. Starting values for MCMC are stored in `model` and/or `data`. Tuning parameters for MCMC are stored in the structure array `mcmc`. Note that the name `mcmc` could be substituted by any name, e.g. `mymcmc`. The automatic choice will produce 5000 MCMC draws after a burn-in of 1000 draws. These values could be easily changed after calling `mcmcstart`:

```
[data,model,mcmc] = mcmcstart(data,model);
mcmc.M=10000; mcmc.burinin;
```

More details about the fields of the structural array `mcmc` appear in Section 4.3.1. To get started quickly, the package could be run without caring about these details.

### 1.1.5 Running MCMC

Bayesian inference for finite mixture models using MCMC is carried out by calling the function `mixturemcmc` with four input arguments, namely, the data stored e.g. in `data`, the model stored e.g. in `model`, the prior stored e.g. in `prior`, and the MCMC tuning parameters stored e.g. in `mcmc`:

```
mcmcout= mixturemcmc(data,model,prior,mcmc);
```

The MCMC draws are stored in the structural array `mcmcout`. The fields of this array are explained in detail in Subection 4.3.2. Note that the name `mcmcout` could be substituted by any name, e.g. `mymcmcout`.

### 1.1.6 Bayesian Inference Based on the MCMC Draws

There exist various ways to explore the MCMC draws stored e.g. in `mcmcout`. This section provides a short overview, more details appear in Section 4.5. The function `mcmcplot` could be used to plot the MCMC output:

```
mcmcplot(mcmcout);
```

Most of these figures are trace plots.

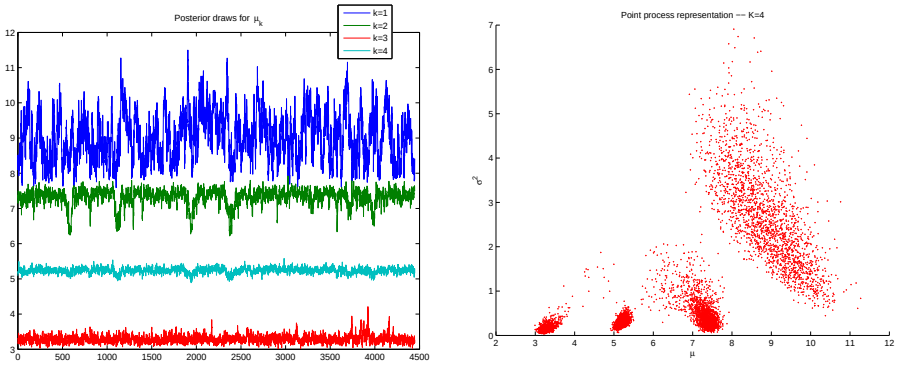


**Fig. 1.1.** FISHERY DATA, running the demo `start_fishery_K4.m`; MCMC draws for $\mu_k$ (left hand side), point process representation (right hand side)

### Parameter Estimation

Call the function `mcmcestimate` for parameter estimation based on the MCMC draws:

```
est = mcmcestimate(mcmcout);
```

Note that `est` could be substituted by any name, e.g. `myest`. `est` is a structural array with various fields containing different parameter estimates. Parameter estimates where a unique labelling has been defined using unsupervised clustering are stored in the field `est.ident` which has several fields corresponding to the various parameters. The estimated weight distribution, for instance, is stored in `est.ident.weight`, while the component parameters are stored in the various fields of `est.ident.par`.

### Clustering the Data

To perform clustering based on the MCMC draws call the function `mcmcclust`:

```
clust = mcmcclust(data,mcmcout);
```

Note that `clust` could be substituted by any name, e.g. `myclust`. `clust` is a structural array with various fields containing different estimators of the unknown allocations. The minimum classification risk estimator, for instance, is stored in `clust.Sident`. Call the function `mcmcclustplot` to plot the clustered data:

```
mcmcclustplot(data,clust);
```

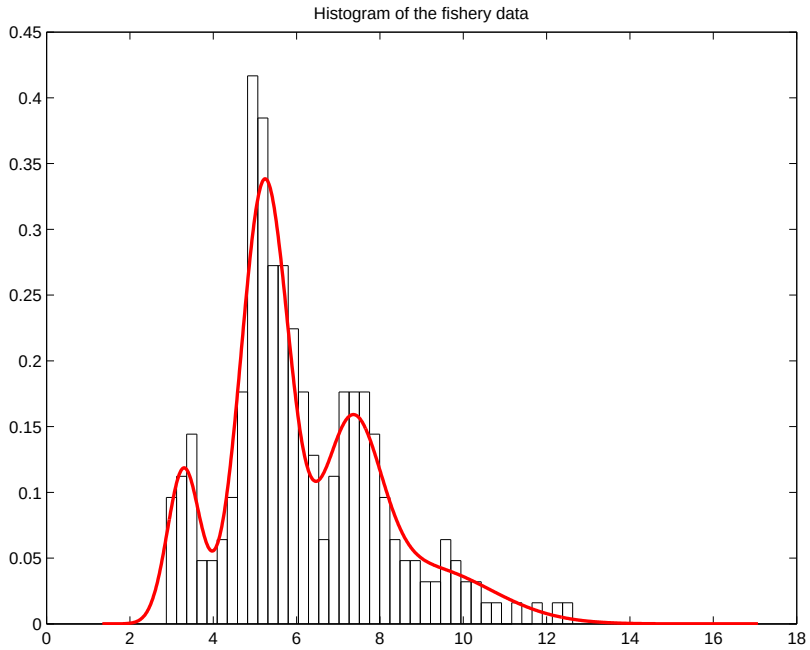


**Fig. 1.2.** FISHERY DATA, running the demo `start_fishery_K4.m`; histogram of the data in comparison to the fitted three component normal mixture distribution

### Computing the Marginal Likelihood

To compute the marginal likelihood of a finite mixture model (with fixed number of components) using the MCMC draws call the function `mcmcbf`:

```
marlik = mcmcbf(data,mcmcout);
```

Note that `marlik` could be substituted by any name, e.g. `mymarlik`. `marlik` is a structural array with various fields containing different estimators of the marginal likelihood. The bridge sampling estimator, for instance, is stored in `marlik.bs`.

## 1.2 Examples

Many demos are included in the package to show how to fit finite mixture models to real and to simulated data. Demos for real data are named `start_dataname`, where `dataname` is the name used in the function `dataget` to load the data, e.g. `fishery`. See Subsection 3.1.3 for a detailed description of the data set analyzed below. Results are stored in MATLAB files named `store_dataname_xxx.mat`.

**Table 1.2.** FISHERY DATA, running the demo `start_fishery.m`; log of the bridge sampling estimator of the marginal likelihood $p(\mathbf{y}|\mathcal{M}_K)$ under the default prior; standard errors in parenthesis

| | $K$ | | | | |
|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 |
| $\hat{p}(\mathbf{y}\|\mathcal{M}_K)$ | -535.11 | -525.68 | -521.50 | **-518.97** | -521.26 |
| | (6.5972e-004) | (0.012) | (0.0089) | (0.0267) | (0.0425) |

All subsequent implementations were carried out using MATLAB (Version 7.3.0) on a notebook with a 2.0 GHz processor.

### 1.2.1 Fishery Data

The program `start_fishery_K4.m` fits a finite mixture of four univariate normal distributions as in Frühwirth-Schnatter (2006, Subsection 6.2.8) to the FISHERY DATA (Titterington et al., 1985) using the default prior (takes about 2 CPU minutes). The program produces 5000 MCMC draws (takes about 30 CPU seconds), plots the MCMC draws, see e.g. Figure 1.1, performs parameter estimation, computes the marginal likelihood of the model (takes about 30 CPU seconds) and plots the fitted mixture, see Figure 1.2.

The program `start_fishery.m` fits finite mixtures of univariate normal distributions with $K = 1$ to $K = 5$ to the FISHERY DATA using the default prior and computes the marginal likelihood $p(\mathbf{y}|\mathcal{M}_K)$ for each model as in Frühwirth-Schnatter (2006, Subsection 7.1.5) (takes about 11 CPU minutes). Table 1.2 shows the log of the bridge sampling estimator of the marginal likelihood $p(\mathbf{y}|\mathcal{M}_K)$. The model with the largest marginal likelihood is a mixture of four normal distributions.

### 1.2.2 Fisher's Iris Data

The program `start_iris_K3.m` fits a finite mixture of three multivariate normal distributions as in Frühwirth-Schnatter (2006, Subsection 6.4.3) to FISHER'S IRIS DATA using the default prior (takes about 3 CPU minutes). The program produces 5000 MCMC draws (takes about 40 CPU seconds), plots the MCMC draws, computes the marginal likelihood of the model (takes about 110 CPU seconds), performs clustering (takes less than 2 CPU seconds) and plots the clustered data, see e.g. Figure 1.3.

The program `start_iris.m` fits finite mixtures of multivariate normal distributions with $K = 1$ to $K = 5$ to the FISHER'S IRIS DATA using the default prior and computes the marginal likelihood $p(\mathbf{y}|\mathcal{M}_K)$ for each model as in Frühwirth-Schnatter (2006, Subsection 7.1.6) (takes about 11 CPU minutes). Table 1.3 shows the log of the bridge sampling estimator of the marginal likelihood $p(\mathbf{y}|\mathcal{M}_K)$. The model with the largest marginal likelihood is a mixture of three multivariate normal distributions.

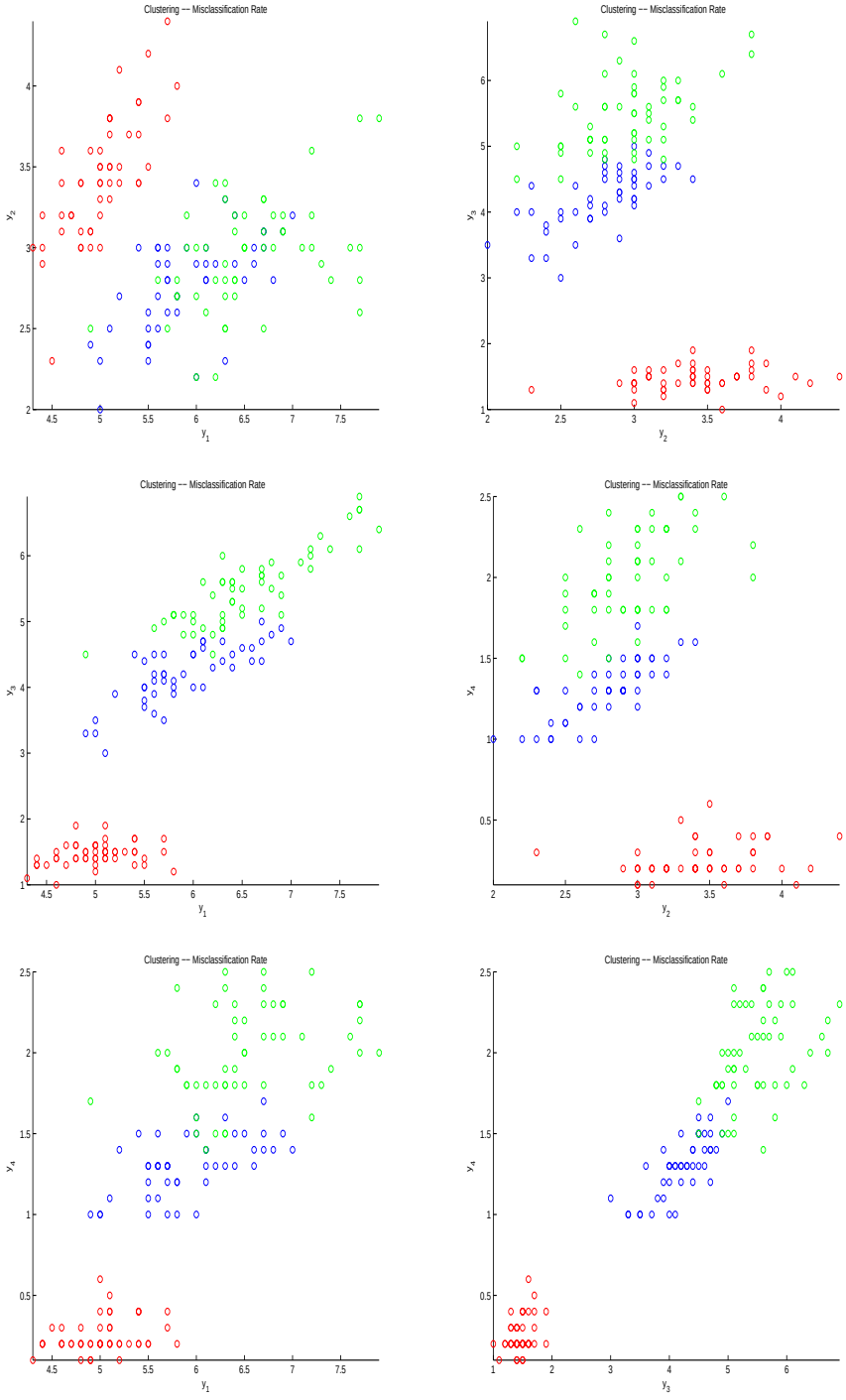


**Fig. 1.3.** FISHER'S IRIS DATA, running the demo `start_iris_K3.m`; clustering of the data into three groups based on the misclassification rate

**Table 1.3.** Fisher's Iris Data, running the demo `start_iris.m`; log of the bridge sampling estimator of the marginal likelihood $p(\mathbf{y}|\mathcal{M}_K)$ under the default prior; standard errors in parenthesis

| | $K$ | | | | |
|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 |
| $\hat{p}(\mathbf{y}\|\mathcal{M}_K)$ | -430.11 | -302.27 | **-294.53** | -297.65 | -307.45 |
| | (0.0026) | (0.0056) | (0.0120) | (0.0353) | (0.0514) |

### 1.2.3 Eye Tracking Data

**Table 1.4.** Eye Tracking Data, running the demo `start_eye.m`; log of the bridge sampling estimator of the marginal likelihood $p(\mathbf{y}|\mathcal{M}_K)$ (standard errors in parenthesis) and corresponding model posterior probabilities $p(\mathcal{M}_K|\mathbf{y})$ under the Poisson prior $K \sim \mathcal{P}(1)$ obtained from two independent MCMC runs

| | $K$ | | | | | | |
|---|---|---|---|---|---|---|---|
| | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| $\hat{p}(\mathbf{y}\|\mathcal{M}_K)$ | -472.9 | -254.2 | -239.8 | -234.5 | -233.4 | -234.8 | -236.2 |
| | (2.3e-006) | (9.9e-004) | (0.012) | (0.014) | (0.034) | (0.025) | (0.027) |
| $\hat{p}(\mathcal{M}_K\|\mathbf{y})$ | 0.0 | 0.0 | 0.0123 | **0.606** | 0.366 | 0.0156 | 0.0005 |
| second MCMC run | 0.0 | 0.0 | 0.0129 | **0.622** | 0.346 | 0.0180 | 0.0006 |

The program `start_eye.m` fits finite mixtures of Poisson distributions with $K = 1$ to $K = 7$ to the Eye Tracking Data (Pauler et al., 1996) and computes the marginal likelihood $p(\mathbf{y}|\mathcal{M}_K)$ for each model (takes about 11 CPU minutes). The prior of the parameters as well as the prior model probabilities are selected as in Frühwirth-Schnatter (2006, Subsection 9.2.4).

Table 1.4 shows the log of bridge sampling estimator of the marginal likelihood $p(\mathbf{y}|\mathcal{M}_K)$ and the corresponding model posterior probabilities $p(\mathcal{M}_K|\mathbf{y})$ under the Poisson prior $K \sim \mathcal{P}(1)$. The last row of this table, showing the model posterior probabilities obtained from a second, independent MCMC run, indicates that the estimators of these probabilities are rather imprecise. Nevertheless, for both MCMC runs the model with the largest posterior probability is a mixture of four Poisson distributions.

Finally, the program `start_eye.m` identifies the mixture of four Poisson distributions using unsupervised clustering as explained in Subsection 4.5.2. The corresponding estimators are given in Table 1.5 and are, apart from relabeling, rather similar to the estimators obtained in Frühwirth-Schnatter (2006, Table 9.3) under the identifiability constraint $\mu_1 < \mu_2 < \mu_3 < \mu_4$, where $\mu_k$ is the group specific mean.

**Table 1.5.** Eye Tracking Data, running the demo `start_eye.m`; identifying a mixture of four Poisson distributions through unsupervised clustering; parameters estimated by posterior means of identified MCMC draws

| | Group $k$ | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| mean $\mu_k$ | 1.25 | 20.11 | 0.028 | 7.89 |
| weight $\eta_k$ | 0.360 | 0.101 | 0.333 | 0.206 |

### 1.2.4 Fabric Fault Data

We reconsider analysis of the Fabric Fault Data (Aitkin, 1996) under various non-Gaussian mixture regression models as in Frühwirth-Schnatter et al. (2009). The response variable $y_i$ is the number of faults in a bolt of length $l_i$. Based on the regressor matrix $(1 \;\; \log l_i)$, we fit a Poisson and a negative binomial regression model as well as finite mixtures of Poisson and negative binomial regression models with $K = 2$ and $K = 3$ groups. Furthermore we consider mixtures of regression models, where the intercept is group specific, while the slope is fixed, both for the Poisson and the negative binomial distribution.

**Table 1.6.** Fabric Fault Data; log marginal likelihoods of various regression models computed as in Frühwirth-Schnatter et al. (2009). Standard errors are given in parentheses.

| Model | $K = 1$ | $K = 2$ | $K = 3$ |
|---|---|---|---|
| Poisson | −101.79 | −99.21 | −100.74 |
| | (0.002) | (0.01) | (0.05) |
| Poisson | −101.79 | −97.46 | −97.65 |
| (fixed slope) | (0.002) | (0.073) | |
| Negative Binomial | −96.04 | −99.05 | −102.21 |
| | (0.007) | (0.027) | (0.038) |

The program `start_fabricfault.m` fits a standard Poisson regression model as well as mixtures of Poisson regression models with $K = 2$ to $K = 3$ under the default prior (takes about 7 CPU minutes). The program `start_fabricfault_mixed_effects.m` fits a Poisson regression model as well as mixtures of Poisson regression models with $K = 2$ to $K = 3$ where the slope is fixed under the default prior (takes about 7 CPU minutes). Finally, the program `start_fabricfault_negbin.m` fits a negative binomial regression model as well as mixtures of negative binomial regression models with $K = 2$ and $K = 3$ (takes about 8 CPU minutes). For the degrees of freedom parameter $\nu_k$, the default prior is changed to match the prior used in Frühwirth-Schnatter et al. (2009).

All programs compute the log of the marginal likelihood which is used to select the best model, see Table 1.6. It turns out that the negative binomial regression model has the smallest marginal likelihood, thus no mixture model is needed for these data. For a Poisson regression model two groups are present, however this model is outperformed by the negative binomial regression model. Figure 1.4 shows the MCMC posterior draws of the degrees of freedom parameter $\nu$ and the posterior density $p(\nu|\mathbf{y})$, estimated through a histogram of the MCMC draws. The degrees of freedom is a finite integer parameter around 10, providing additional evidence for the negative binomial distribution.

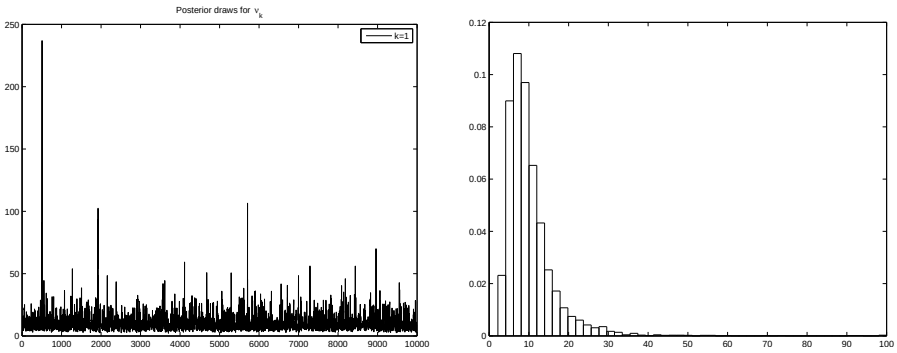


**Fig. 1.4.** Fabric Fault Data, running the demo `start_fabricfault_negbin.m`; MCMC draws for $\nu$ (left hand side) and posterior density $p(\nu|\mathbf{y})$, estimated through a histogram of the MCMC draws (right hand side)

### 1.2.5 Lamb Data

**Table 1.7.** Lamb Data, running the demo `start_lamb.m`; log of the bridge sampling estimator of the marginal likelihood $p(\mathbf{y}|\mathcal{M}_K)$ (standard errors in parenthesis)

| | $K$ | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| $\hat{p}(\mathbf{y}\|\mathcal{M}_K)$ | -204.25 | -184.76 | -178.89 | -178.58 |
| | | (0.0044) | (0.0097) | (0.0337) |

The program `start_lamb.m` fits a Markov mixture of Poisson distributions with increasing number of states ($K = 1$ to $K = 4$) to the Lamb Data (Leroux and Puterman, 1992) and computes the marginal likelihood $p(\mathbf{y}|\mathcal{M}_K)$

for each model (takes about 7 CPU minutes). The default prior is changed to match the prior used in Frühwirth-Schnatter (2006, Subsection 11.7.3).

Table 1.7 shows the log of the bridge sampling estimator of the marginal likelihood $p(\mathbf{y}|\mathcal{M}_K)$. There is no significant difference between $\hat{p}(\mathbf{y}|\mathcal{M}_3)$ and $\hat{p}(\mathbf{y}|\mathcal{M}_4)$, therefore a Markov mixture of three Poisson distributions is selected.

Finally, the program `start_lamb.m` identifies the selected Markov mixture of three Poisson distributions using unsupervised clustering as explained in Subsection 4.5.2. The corresponding estimators are given in Table 1.8 and are rather similar to the estimators obtained in Frühwirth-Schnatter (2006, Table 11.2) under the identifiability constraint $\mu_1 < \mu_2 < \mu_3$, where $\mu_k$ is the state specific mean.

**Table 1.8.** LAMB DATA, running the demo `start_lamb.m`; identifying a Markov mixture of three Poisson distributions through unsupervised clustering; parameters estimated by posterior means of identified MCMC draws

| | State 1 | State 2 | State 3 |
|---|---|---|---|
| mean $\mu_k$ | 0.0695 | 0.497 | 3.120 |
| transition matrix $\boldsymbol{\xi}$ | | | |
| first line | 0.946 | 0.038 | 0.016 |
| second line | 0.045 | 0.943 | 0.012 |
| third line | 0.166 | 0.126 | 0.709 |

### 1.2.6 GDP Data

**Table 1.9.** GDP DATA, running the demos `start_gdp.m` and `start_gdp_swi.m`; log of the bridge sampling estimator of the marginal likelihood $p(\mathbf{y}|\mathcal{M}_K)$

| | AR(p) | Switching AR | | Switching intercept | |
|---|---|---|---|---|---|
| $p$ | $K=1$ | $K=2$ | $K=3$ | $K=2$ | $K=3$ |
| 0 | -199.72 | -194.31 | -193.28 | | |
| 1 | -194.22 | -193.68 | -194.87 | -192.72 | -193.97 |
| 2 | -196.32 | **-191.65** | -194.36 | -194.43 | -195.83 |
| 3 | -197.31 | -193.63 | -196.53 | -194.98 | -196.08 |
| 4 | -199.26 | -195.33 | -199.37 | -196.10 | -196.92 |

The program `start_gdp.m` fits a Markov switching autoregressive model with different number of states ($K = 1$ to $K = 3$) and increasing AR order ($p = 0$ to $p = 4$) to the GDP DATA (Hamilton, 1989). MCMC estimation and computing the marginal likelihood $p(\mathbf{y}|\mathcal{M}_K)$ for each of these 15 models takes in total about 26 CPU minutes. The program `start_gdp_swi.m` fits a

reduced version of these models, namely a Markov switching autoregressive model where only the intercept is switching, again with different number of states ($K = 2$ to $K = 3$) and increasing AR order ($p = 1$ to $p = 4$). MCMC estimation and computing the marginal likelihood $p(\mathbf{y}|\mathcal{M}_K)$ for each of these 8 models takes in total about 17 CPU minutes. In both cases, the default prior is changed to match the prior used in Frühwirth-Schnatter (2006, Subsection 12.2.6).

Table 1.9 compares the log of the bridge sampling estimator of each marginal likelihood $p(\mathbf{y}|\mathcal{M}_K)$. The model with the largest marginal likelihood is a Markov switching autoregressive model with two states of order 2, where both the intercept and all autoregressive coefficients are switching.

Finally, the program `start_gdp.m` identifies the selected model using unsupervised clustering as explained in Subsection 4.5.2. The corresponding estimators are given in Table 1.10 and are very similar to the estimators obtained in Frühwirth-Schnatter (2006, Table 12.2) under the identifiability constraint $\zeta_1 < \zeta_2$, where $\zeta_k$ is the state specific intercept.

**Table 1.10.** GDP Data, running the demos `start_gdp.m`; identifying a Markov switching autoregressive model with two states of order 2 through unsupervised clustering; parameters estimated by posterior means of identified MCMC draws

| Parameter | Contraction ($k = 1$) | Expansion ($k = 2$) |
|---|---|---|
| intercept $\zeta_k$ | -0.571 | 1.069 |
| AR(1) coefficient $\delta_{k,1}$ | 0.234 | 0.281 |
| AR(2) coefficient $\delta_{k,2}$ | 0.462 | -0.116 |
| variance $\sigma^2_{\varepsilon,k}$ | 0.780 | 0.704 |
| transition probability $\xi_{kk'}$ | 0.488 | 0.327 |

# 2

# Finite Mixture Modeling

This toolbox has been designed to fit finite mixture models to data. To this aim it is necessary to specify the mixture model. The general idea how models are defined in this toolbox is described in Section 2.1, whereas Section 2.2 deals with the definition of finite mixture models.

## 2.1 Specifying Models

In this toolbox a model is in general specified as a structure array, named for instance `model`, with various fields defining the model. When defining a model, a distinction is made between the model structure and model parameters.

### 2.1.1 Specifying the Model Structure

To define the model structure, the conditional distribution of the observation $\mathbf{Y}_i$ given the unknown parameters has to be specified in the field `.dist`. Possible distributions are summarized in Table 1.1.

The most simple model is based on the assumption that the data are i.i.d. replications from a distribution of type `dist` and the parameters in the distribution are assumed to be homogenous over the replications, e.g. for count data the model may read $Y_i \sim \mathcal{P}(\mu)$ with $\mu$ unknown. In this case no further model definition is necessary. To fit such a homogeneity model to data using this toolbox, for instance, it is sufficient to specify the distribution `dist`, no other information is necessary.

For more elaborated models additional fields are needed to define the model. For a finite mixture model, for instance, it is only necessary to specify the number of components by assigning a value to the field `K`, see Subsection 2.2.1. If such a field is missing, it is automatically assumed that $K = 1$.

The finite mixture model is a latent variable models where the conditional distribution of the data depends on a latent variable, namely the hidden indicator. In such a case, a model structure has to be specified for the latent

variables. A default choice is made, namely assuming a hidden multinomial model for the latent indicators. Otherwise, a field has to added to the model definition, which provides an explicit definition of the model for the latent variable, like adding the field `indicmod` to define a Markov switching model, see Subsection 9.2.1.

### 2.1.2 Assigning Parameters

For any statistical model, the conditional distribution of the data depends on unknown model parameters. Such parameters are stored in the field `par`, which is either a single numeric value, a numerical array or a structure array for higher dimensional parameters, depending on the distribution family. In the case of a homogeneous $\mathcal{P}(\mu)$-distribution, for instance, the field `par` is a single numeric value equals $\mu$, see also Section 7.2. For the normal distribution, the field `par` is a structure array with different fields, one of them, namely `mu` defining the mean, whereas the other one, namely `sigma`, defines the variance-covariance matrix of the distribution, see also Section 6.2.1.

For many distribution families there exist different ways to parameterize a distribution. For a multivariate normal distribution, for instance, one may either specify the covariance matrix $\mathbf{\Sigma}$ or the inverse matrix $\mathbf{\Sigma}^{-1}$. These values will be stored in different fields, namely `sigma` and `sigmainv`, see also Section 6.2.1. The package will check which of these fields is defined and use the appropriate value for computation.

### 2.1.3 Unspecified and Fully Specified Models

One has to distinguish between an unspecified, a partially specified, and a fully specified model. For a fully specified model, numerical values are assigned to all parameters in the model. For such a model many characteristics, like the moments of the marginal distribution, may be computed or data may be simulated from that model. When fitting a model to data, the model is typically unspecified, meaning that the parameters of the underlying distribution family are unknown.

## 2.2 Finite Mixture Distributions

Frühwirth-Schnatter (2006, Section 1.2) provides an introduction into finite mixture modelling.

### 2.2.1 Defining a Finite Mixture Distribution

In Frühwirth-Schnatter (2006, Subsection 1.2.1), a random variable $\mathbf{Y}$ with density (2.1)

$$p(\mathbf{y}|\boldsymbol{\vartheta}) = \eta_1 p(\mathbf{y}|\boldsymbol{\theta}_1) + \cdots + \eta_K p(\mathbf{y}|\boldsymbol{\theta}_K), \tag{2.1}$$

where all component densities arise from the same parametric distribution family $\mathcal{T}(\boldsymbol{\theta})$ with density $p(\mathbf{y}|\boldsymbol{\theta})$, indexed by a parameter $\boldsymbol{\theta} \in \Theta$, is said to arise from a (standard) finite mixture of $\mathcal{T}(\boldsymbol{\theta})$ distributions, abbreviated by

$$\mathbf{Y} \sim \eta_1 \mathcal{T}(\boldsymbol{\theta}_1) + \cdots + \eta_K \mathcal{T}(\boldsymbol{\theta}_K).$$

This could be seen as the marginal distribution of a model, where a hidden categorical indicator $S$ is introduced which is assumed to follow a multinomial distribution:

$$\begin{aligned} &\mathbf{Y}|S \sim \eta_1 \mathcal{T}(\boldsymbol{\theta}_S), \\ &S \sim \text{MulNom}\,(1, \eta_1, \ldots, \eta_K)\,. \end{aligned}$$

In the package, a standard finite mixture model is defined as a structure array, named for instance `mix`, containing the following fields:

- The field `dist` shows the parametric distribution family $\mathcal{T}(\boldsymbol{\theta})$ characterized by a string variable. The current version of the package is able to handle the following distribution families:
  - `'Normal'`: normal distribution $\mathcal{N}\left(\mu_k, \sigma_k^2\right)$,
  - `'Normult'`: $r$-variate normal distribution $\mathcal{N}_r\left(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right)$,
  - `'Exponential'`: exponential distribution $\mathcal{E}\left(\lambda_k\right)$,
  - `'Student'`: Student-$t$ distribution $t_{\nu_k}\left(\mu_k, \sigma_k^2\right)$,
  - `'Studmult'`: $r$-variate Student-$t$ distribution $t_{\nu_k}\left(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k\right)$,
  - `'Poisson'`: Poisson distribution $\mathcal{P}\left(\mu_k\right)$,
  - `'Binomial'`: binomial distribution $\text{BiNom}\left(T_i, \pi_k\right)$.

  The package will check just the first six characters, therefore the types may be abbreviated.
- For multivariate mixtures, the field `r` contains the dimension of the realization `y`.
- The field `K` contains the number $K$ of components.
- The field `weight` contains the weight distribution $\boldsymbol{\eta} = (\eta_1, \ldots, \eta_K)$, characterized by a `1 x K` numeric array.
- The field `par` contains the component parameters $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K$. The structure of this field depends on the distribution family and on the dimension of $\boldsymbol{\theta}_k$. For Poisson mixtures, the field `par` is a `1 x K` numeric array, containing the component parameters $\mu_1, \ldots, \mu_K$. For details, how `par` is defined for mixtures of normal distributions, see Subsection 6.2.1.

For $K = 1$ just a single member from the distribution family is used. In this case the fields `K` and `weight` need not be defined.

Other models for the indicators are possible which are defined through the field `indicmod`, see Subsection 9.2.1 for Markov mixture models. If this field is missing in the definition of the mixture model, than it is automatically assumed that a standard finite mixture is considered, where $S$ follows a multinomial distribution with parameter `weight`.

### 2.2.2 Plotting the Density of a Finite Mixture Distribution

To plot the density of a finite mixture distribution, defined by the structure array `mix` as discussed in Subsection 2.2.1, use the function `mixtureplot` which is defined with variable input/output argument, handling figure numbers:

```
mixtureplot(mix); % starts plotting with Figure 1
mixtureplot(mix,nplot); % starts plotting with Figure nplot
nplot=mixtureplot(mix,nplot); % returns the number of the last Figure
```

For a bivariate mixture a surface plot, a contour plot and a colored projection onto the $(y_1, y_2)$ plane is produced. The surface plot is returned in rotate mode, so it may be rotated interactively by means of the mouse.

For multivariate mixtures with $r = 3, 4, 5$, a contour plot is produced for each bivariate marginal density. To visualize higher dimensional densities you have to extract lower dimensional marginal densities using the function `mixturemar`, before applying the function `mixtureplot`, see Subsection 2.2.3.

**Example**

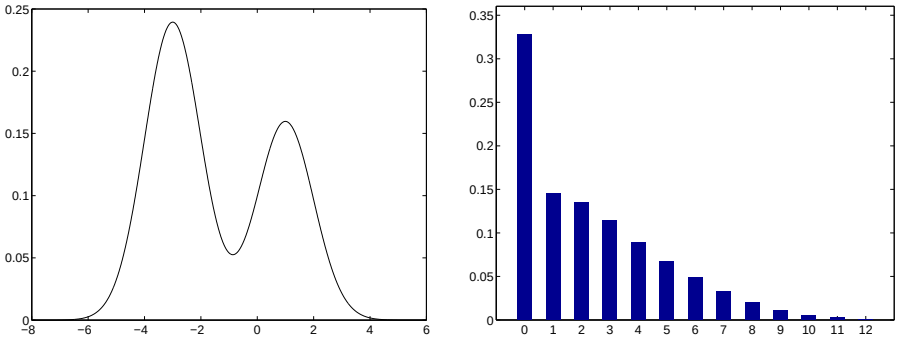


**Fig. 2.1.** Density of a mixture of two normal distributions (left-hand side) and a mixture of three Poisson distributions (right-hand side)

The package contains a MATLAB demo named `demo_figure2_1.m` which produces the plots shown in Figure 2.1. The demo first defines a mixture of two univariate normal distribution as a structure array named `mix` and plots the density shown in the left hand side of using the function `mixtureplot`. Then it defines a mixture of three Poisson distributions as a structure array named `mixpoi`, plots the density shown in the right hand side of Figure 2.1 and provides the following information about `mixpoi`:

```
dist: 'Poisson'
K: 3.00
weight: [0.30 0.40 0.30]
par: [0.10 2.00 5.00]
```

### 2.2.3 Marginal Densities

To obtain the marginal density of the component $Y_j$ of $\mathbf{Y}$ for multivariate mixtures like multivariate normal or $t$ mixtures you may use the function `mixturemar`:

```
mixj= mixturemar(mix,j);
```

The resulting model `mixj` is a univariate mixture model, and may be analyzed as such. The same function may also be used to obtain the marginal density of any selection of components $Y_{j_1}, \ldots, Y_{j_n}$ of $\mathbf{Y}$ as a $n$-variate mixture model:

```
mixn= mixturemar(mix,[j1 j2 ... jn]);
```

### 2.2.4 Moments of a Finite Mixture Distribution

To compute moments of finite mixture distribution, like $\mathrm{E}(Y|\boldsymbol{\vartheta})$, $\mathrm{Var}(Y|\boldsymbol{\vartheta})$ or $\mathrm{E}((Y-\mu)^m|\boldsymbol{\vartheta})$, as described in Frühwirth-Schnatter (2006, Subsection 1.2.4), use the function `moments`

```
mom=moments(mix);
```

where `mix` is a structure array defining a fully specified mixture model
This function produces a structure array with following fields:

- The field `mean` contains the mean (vector) $\mathrm{E}(\mathbf{Y}|\boldsymbol{\vartheta})$ of the mixture distribution. This is a scalar for univariate mixtures and a `r x 1` numerical array for multivariate mixtures, with `r` being the dimension of the data.
- The field `var` contains the variance (covariance matrix) $\mathrm{Var}(\mathbf{Y}|\boldsymbol{\vartheta})$ of the mixture distribution. This is a scalar for univariate mixtures and a `r x r` numerical array for multivariate mixtures,

For finite mixture of normal or $t$ distributions the function `moments` computes higher order moments up to `L=4`. To change the value of `L` call the function `moments` with a second argument as:

```
mom=moments(mix,L);
```

The following additional fields are produced for mixtures of normal or Student-$t$ distributions:

- For $L > 2$, the field `high` contains higher order moments around the mean. For univariate mixtures, this a `1 x L` numerical array, containing the moments $\mathrm{E}((Y-\mu)^m|\boldsymbol{\vartheta})$, for $m = 1, \ldots, L$. For multivariate mixtures, this a `r x L` numerical array, containing the moments $\mathrm{E}((Y_j-\mu_j)^m|\boldsymbol{\vartheta})$ of the marginal density of $Y_j$ for $m = 1, \ldots, L$ for $j = 1, \ldots, r$.
- For $L \geq 3$, the field `skewness` contains the skewness coefficient of each marginal mixture, defined as

$$\frac{\mathrm{E}((Y_j-\mu_j)^3|\boldsymbol{\vartheta})}{\mathrm{Var}(Y_j|\boldsymbol{\vartheta})^{3/2}},$$

  for $j = 1, \ldots, r$. This is a scalar for univariate mixtures and a `r x 1` numerical array for multivariate mixtures.
- For $L \geq 4$, the field `kurtosis` contains the kurtosis coefficient of each marginal mixture of normals, defined as

$$\frac{\mathrm{E}((Y_j-\mu_j)^4|\boldsymbol{\vartheta})}{\mathrm{Var}(Y_j|\boldsymbol{\vartheta})^2},$$

  for $j = 1, \ldots, r$. This is a scalar for univariate mixtures and a `r x 1` numerical array for multivariate mixtures.
- The field `B`, containing the between-group heterogeneity

$$\sum_{k=1}^{K} \eta_k (\boldsymbol{\mu}_k - \mathrm{E}(\mathbf{Y}|\boldsymbol{\vartheta}))(\boldsymbol{\mu}_k - \mathrm{E}(\mathbf{Y}|\boldsymbol{\vartheta}))'.$$

- The field `W`, containing the within-group heterogeneity

$$\sum_{k=1}^{K} \eta_k \boldsymbol{\Sigma}_k.$$

- The coefficient of determination, defined for multivariate mixtures, either by

$$R_t^2(\boldsymbol{\vartheta}) = 1 - \mathrm{tr}\left(\textstyle\sum_{k=1}^{K} \eta_k \boldsymbol{\Sigma}_k\right) / \mathrm{tr}\left(\mathrm{Var}(\mathbf{Y}|\boldsymbol{\vartheta})\right),$$

  and stored in the field `Rtr`, or

$$R_d^2(\boldsymbol{\vartheta}) = 1 - \left|\sum_{k=1}^{K} \eta_k \boldsymbol{\Sigma}_k\right| / |\mathrm{Var}(\mathbf{Y}|\boldsymbol{\vartheta})|,$$

  contained in the field `Rdet`. For a univariate mixture of normals, both definitions reduce to the same scalar value, stored in the field `R`.

For more details, see Frühwirth-Schnatter (2006, Subsection 6.1.1, p.170).

The following additional fields will be produced for discrete mixtures:

- The field `over` is a scalar, containing the overdispersion $\mathrm{Var}(Y|\boldsymbol{\vartheta}) - \mathrm{E}(Y|\boldsymbol{\vartheta})$.
- The field `factorial` is a `1x L` numerical array, containing the first `L` factorial moments $\mathrm{E}(Y!/(Y-j)!|\boldsymbol{\vartheta})$, for $j = 1, \ldots, L$. For mixtures of Poisson distributions these are given by:

$$\mathrm{E}(Y!/(Y-j)!|\boldsymbol{\vartheta}) = \sum_{k=1}^{K} \eta_k \mu_k^j.$$

- The field `zero` contains $\Pr(Y = 0|\boldsymbol{\vartheta})$, the probability to observe 0. For Poisson mixture this is defined as:

$$\Pr(Y = 0|\boldsymbol{\vartheta}) = \sum_{k=1}^{K} \eta_k e^{-\mu_k}.$$

For more details for Poisson mixtures see Subsections 9.2.2 and 9.2.3 in Frühwirth-Schnatter (2006).

### 2.2.5 The Point Process Representation of a Finite Mixture Distribution

To obtain a point process representation of a finite mixture distribution as discussed in Frühwirth-Schnatter (2006, Subsection 1.2.3, p.10) use the function `mixturepoint` which is defined with variable input/output arguments, handling figure numbers:

```
mixturepoint(mix); % starts plotting with Figure 1
mixturepoint(mix,nplot); % starts plotting with Figure nplot
nplot=mixturepoint(mix,nplot); % returns the number of the last Figure
```

For mixtures with univariate component parameter $\theta$, like mixtures of Poisson distributions, $\theta_k$ will be plotted against 0. For mixtures with bivariate component parameter $\boldsymbol{\theta} = (\theta_1, \theta_2)$, $\theta_{1,k}$ will be plotted against $\theta_{2,k}$. For mixtures with multivariate components parameters $\boldsymbol{\theta}$, special point process representations will be generated for each type of mixture models.

#### Example

The package contains a MATLAB demo named `demo-figure2-2.m` which produces the point process representations of two different mixtures of three univariate normal distributions shown in Figure 2.2.

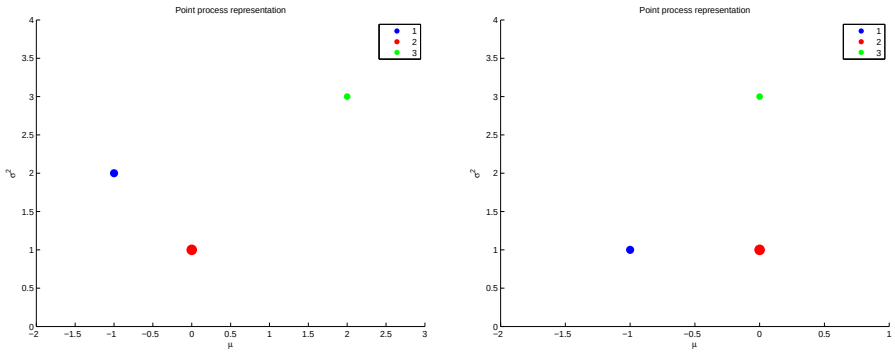


**Fig. 2.2.** Point process representation of two different mixtures of three normal distributions

# 3

# Data Handling

## 3.1 Defining the Data

The data have to be stored in a structure array. Subsequently this array is called `data`, but any other name is possible. Depending on the nature of the data, this structure array will have different fields. This chapter deals with a sequence of $N$ independent observations $\mathbf{y} = (y_1, \ldots, y_N)$. More general data structure will be handled in later chapters, in particular, regression type data in Section 8.1 and time series data in Section 9.1.

### 3.1.1 Data Structures

Data are stored in form of a structure array, named for instance `data`, where

- the field `y` contains the observations.

For univariate data $\mathbf{y} = (y_1, \ldots, y_N)$, the field `y` is a `1 x N` numeric array, where `N` is the number of observations. Thus `data.y(i)` is equal to the $i$th observation $y_i$. If `bycolumn` is true (see below), then `y` is a `N x 1` numeric array.

For multivariate data, $\mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_N)$, the different features are stored by row, thus `y` is a `r x N` numeric array, where `r` is the number of features. Thus `data.y(:,i)` is equal to the $i$th observation $\mathbf{y}_i$. If `bycolumn` is true (see below), then `y` is a `N x r` numeric array.

Optional fields of the structure array `data` are the following:

- The field `name` is the name of the data set, stored as character.
- The field `N` is the number of observations.
- The field `r` is the number of features, i.e. the dimension of a single observation.
- The field `bycolumn` is a logical variable which is true, if the features are stored by column. If this field is missing, then it is assumed that the data are stored by row.

- The field `type` specifies the data type for each feature of the data. `type` is a cell array containing character strings. The following types will be understood by the current version of the package:
  - `'continuous'`: realization of a continuous random variable;
  - `'discrete'`: realization of an integer-valued random variable;

  The package will check just the first three characters, therefore the types may be abbreviated.
- The field `sim` is true, if the data were simulated, see Section 3.3.
- The field `model` may contain information about a data model. For simulated data this could be the model used for simulation. For true data this could be a fitted model.

### 3.1.2 Classified Data

In rare cases, for instance for grouped data, the allocations are known. Such data are called complete or classified data. For such data an additional field called `S` containing the allocations $\mathbf{S} = (S_1, \ldots, S_N)$ has to be added to the structure array containing the data:

- `S` is a `1 x N` numeric array, thus `data.S(i)` is the allocation $S_i$ of the $i$th observation $\mathbf{y}_i$.

Usually, the allocations are unknown. During MCMC estimation, the allocations are recovered from the posterior density.

### 3.1.3 Data Sets Available in the Package

**Table 3.1.** Data sets that may be loaded using the function `dataget('name')`, see Subsection 3.1.3 for a detailed description.

| `name` | Description |
|---|---|
| 'eye' | Eye Tracking Data |
| 'fabricfault' | Fabric Fault Data |
| 'fishery' | Fishery Data |
| 'gdp' | GDP Data |
| 'iris' | Fisher's Iris Data |
| 'lamb' | Lamb Data |

Several data sets are stored under particular names and could be loaded into a structure array using the function `dataget(name)` where name is a string. Typing, for instance,

```
datafish=dataget('fishery')
```

loads the FISHERY DATA plotted in Figure 3.1 into the structure array `datafish` and provides the following information:

```
y: [1x256 double]
N: 256
r: 1
sim: 0
name: 'fishery'
type: 'continuous'
```

If the function `dataget` is called without any argument an overview of all valid data names is returned, see also Table 3.1:

- `'eye'`: a data set counting eye anomalies in 101 schizophrenic patients studied by Pauler et al. (1996) and Escobar and West (1998), where the sample variance shows overdispersion in comparison to the sample mean. The data are reanalyzed e.g. in Frühwirth-Schnatter (2006, Subsection 9.2.4).
- `'fabricfault'`: data on fabric faults analyzed in Aitkin (1996). The response variable $y_i$ is the number of faults in a bolt of length $l_i$. The data are reanalyzed e.g. in Frühwirth-Schnatter et al. (2009).
- `'fishery'`: the data set contains of the length of 256 snappers analyzed in Titterington et al. (1985). The data exhibit unobserved heterogeneity because the age of the fish is unobserved. The data are reanalyzed e.g. in Frühwirth-Schnatter (2006, Subsection 6.2.8 and 7.1.5).
- `'gdp'`: percentage growth rate of the U.S. quarterly real GDP series for the period 1951.II to 1984.IV. This time series was analyzed originally in Hamilton (1989), and reanalyzed, e.g. by McCulloch and Tsay (1994), Chib (1996), Frühwirth-Schnatter (2001) and Frühwirth-Schnatter (2004).
- `'iris'`: this data set consists of 150 four-dimensional observations of three species of iris (*iris setosa*, *iris versicolour*, *iris virginica*). The measurements taken for each plant are *sepal length*, *sepal width*, *petal length* and *petal width*. The data were downloaded from ftp://ftp.ics.uci.edu/pub/machine-learning-databases/iris/iris.names. These data differ from the data presented in Fisher's article; errors in the 35th sample in the fourth feature and in the 38th sample in the second and third features were identified by Steve Chadwick (spchadwick@espeedaz.net). These data are analyzed e.g. in Frühwirth-Schnatter (2006, Subsection 6.3.4 and 7.1.6).
- `'lamb'`: the data are the number of movements by a fetal lamb in $T = 240$ consecutive five-second intervals. This is a time series of count data analyzed originally in Leroux and Puterman (1992), and reanalyzed, e.g. by Chib (1996), Frühwirth-Schnatter (2001) and Frühwirth-Schnatter (2004).

The package contains several MATLAB demos analyzing these data, see for instance `start_fishery_K4.m`. These demos are named `start_name_xxx.m`, where `name` is the name used in calling the function `dataget`.

## 3.2 Data Visualization

Various tools are available for visualizing and exploring the data.

### 3.2.1 Simple Plotting

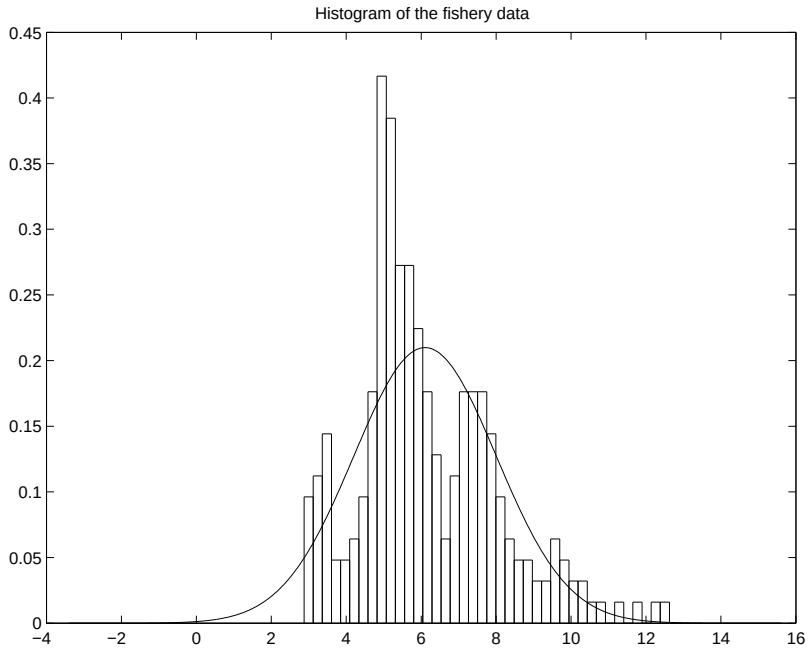


**Fig. 3.1.** Fishery Data, empirical distribution of the observations

To plot a histogram of the data as in Figure 3.1 use the function `dataplot` which is defined with variable input/output argument, handling figure numbers:

```
dataplot(data); % starts plotting with Figure 1
dataplot(data,nplot); % starts plotting with Figure nplot
nplot=dataplot(data,nplot); % returns the number of the last Figure
```

For univariate continuous observations, `dataplot` produces a histogram of the data. For discrete data `dataplot` produces a bar diagram over `[0:max(data.y)]`, with the length of the bar being equal to the absolute frequency of each realization. If the type has not been specified, then the data will be treated as

continuous. If a model has been specified, then the histogram is compared with this model.

For bivariate or multivariate data, `dataplot` produces the following two figures:

- One figure showing a histogram of each feature.
- Another figure showing a scatter plot for each bivariate combination of features.
- For $r > 2$, an additional figure showing a scatter matrix of the data using the intrinsic function `plotmatrix`.

If a model has been specified, then the univariate marginal densities implied by this model are added to the marginal histograms, while contour lines of each bivariate marginal density are added to the scatter plot of each bivariate combination.

### 3.2.2 Empirical Moments

For data stored as a structure array, named `data` for example, the function `datamoments(data)` may be used to compute the sample moments of the data:

```
moments=datamoments(data)
```

This call produces the structure array `moments` with following fields:

- The field `mean` contains the sample mean $\overline{y}$ for univariate data and the vector of sample means $(\overline{y}_1, \ldots, \overline{y}_r)$ for multivariate data. This is a scalar for univariate data and a `r x 1` numerical array for multivariate data, with `r` being the dimension of the data.
- The field `var` contains the sample variance $s_y^2$ for univariate data and the sample covariance matrix $\mathbf{S_y}$ for multivariate data. This is a scalar for univariate data and a `r x r` numerical array for multivariate data.

Note that `moments` could be substituted by any name, e.g. `mymoments`. Additional fields are added for continuous data and for discrete data, respectively.

#### Additional fields for continuous data

The following moments are added to `moments` for continuous data, where `L=4` by default:

- The field `high` contains the first empirical higher order moments around the mean. For univariate data this a `1 x L` numerical array, for multivariate data this is a `r x L` numerical array.
- The field `skewness` contains the empirical skewness coefficient of each feature. This is a scalar for univariate data and a `r x 1` numerical array for multivariate data.

- The field `kurtosis` contains the empirical kurtosis coefficient of each feature. This is a scalar for univariate data and a `r x 1` numerical array for multivariate data.

If the data are multivariate, then the following field are added:

- The field `corr` containing the sample correlation matrix, stored as a `r x r` numerical array.

### Additional fields for discrete data

The following moments are added to `moments` for discrete data:

- The field `factorial` contains the first four factorial sample moments. This is a `4 x 1` numerical array.
- The field `over` contains the sample overdispersion, given by $s_y^2 - \overline{y}$. This is a `4 x 1` numerical array.
- The field `zeros` contains the fractions of zero observations in the sample.

### Additional fields for classified data

If the data are classified, then the following fields are added to `moments`:

- The field `groupmom` containing group specific information. This is a structure array with following fields:
  - `Nk` contains the group sizes, defined by

$$N_k(\mathbf{S}) = \#\{i : S_i = k\}.$$

    This is a 1 x K numerical array.
  - `mean` contains the group averages, defined by

$$\overline{\mathbf{y}}_k(\mathbf{S}) = \frac{1}{N_k(\mathbf{S})} \sum_{i:S_i=k} \mathbf{y}_i.$$

    This is a `1 x K` numeric array.
- `Wk` contains the within-group variability $\boldsymbol{W}_k(\mathbf{S})$, defined by

$$\boldsymbol{W}_k(\mathbf{S}) = \sum_{i:S_i=k} (\mathbf{y}_i - \overline{\mathbf{y}}_k(\mathbf{S}))(\mathbf{y}_i - \overline{\mathbf{y}}_k(\mathbf{S}))'.$$

  For a univariate mixture this is a `1 x K` numeric array. For a multivariate mixture of dimension `r` this is `r x r x K` numeric array.
- `var` contains the within-group (co)variance, defined by

$$\boldsymbol{W}_k(\mathbf{S})/N_k(\mathbf{S}).$$

  For a univariate mixture this is a `1 x K` numeric array. For a multivariate mixture of dimension `r` this is `r x r x K` numeric array.

If the classified data are continuous, then further fields are added to `moments`:

- The field `B`, containing the between-group variance $\boldsymbol{B}(\mathbf{S})$, defined by

$$\boldsymbol{B}(\mathbf{S}) = \sum_{k=1}^{K} N_k(\mathbf{S})(\overline{\mathbf{y}}_k(\mathbf{S}) - \overline{\mathbf{y}})(\overline{\mathbf{y}}_k(\mathbf{S}) - \overline{\mathbf{y}})'.$$

- The field `W`, containing the within-group heterogeneity

$$\boldsymbol{W}(\mathbf{S}) = \sum_{k=1}^{K} \boldsymbol{W}_k(\mathbf{S}).$$

- The field `T`, containing the total variance

$$\mathbf{T} = \sum_{i=1}^{N} (\mathbf{y}_i - \overline{\mathbf{y}})(\mathbf{y}_i - \overline{\mathbf{y}})' = \boldsymbol{W}(\mathbf{S}) + \boldsymbol{B}(\mathbf{S}).$$

- The coefficient of determination, which is defined for multivariate mixtures either by

$$1 - \frac{\mathrm{tr}\,(\boldsymbol{W}(\mathbf{S}))}{\mathrm{tr}\,(\mathbf{T})},$$

  and stored in the field `Rtr`, or by

$$1 - \frac{|\boldsymbol{W}(\mathbf{S})|}{|\mathbf{T}|}.$$

  and stored in the field `Rdet`. For univariate data both definitions reduce to the same scalar value, stored in the field `R`.

### 3.2.3 Examples

#### The Fishery Data data

The package contains a MATLAB demo named `start_fishery_plot.m` which shows how to get access to the Fishery Data data plotted in Figure 3.1, which are then compared with a normal distribution with the same mean and variance as the data.

#### The Eye Tracking Data data

The package contains a MATLAB demo named `start_eye_plot.m` which shows how to get access to the Eye Tracking Data data and how to plot the empirical distribution shown in Figure 3.2. The last two lines compute and display the empirical moments of the data and the theoretical moments of a single Poisson distribution with the same mean as the data:

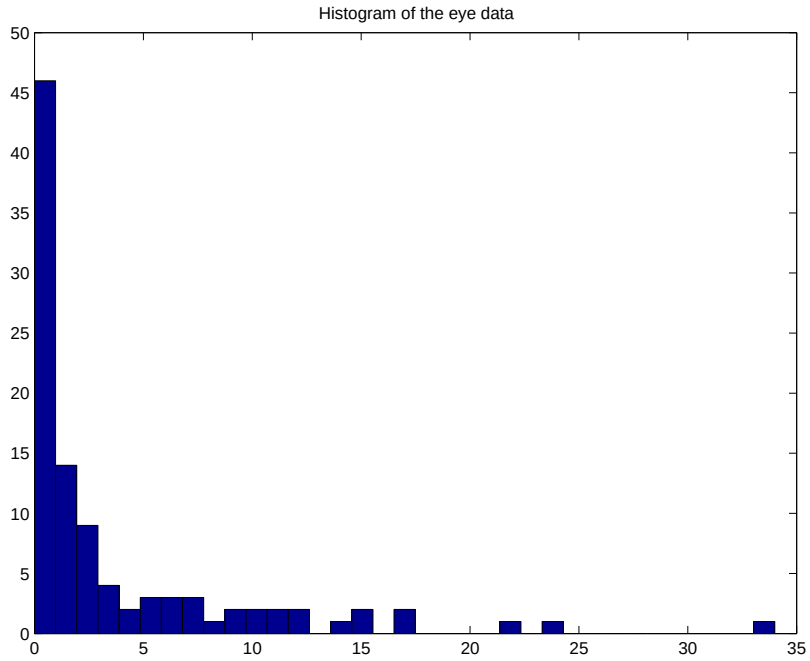


**Fig. 3.2.** Eye Tracking Data, empirical distribution of the observations

```
datamom =
mean: 3.5248
var: 35.5365
over: 32.0118
factorial: [3.5248 44.4356 809.2871 1.7909e+004]
zero: 0.4554

poimom =
mean: 3.5248
var: 3.5248
over: 0
factorial: [3.5248 12.4239]
zero: 0.0295
```

Evidently, the data show a high degree of overdispersion as well as excess zeros compared to a single Poisson distribution.

## 3.3 Data Simulation

To simulate $N$ observations from a model, use the function

```
data=simulate(mymodel,N);
```

Note that `mymodel` has to be a fully specified model, see Subsection 2.1.3. This function produces the structural array `data` with the same fields as in Subsection 3.1.1, including the fields `y`, `N`, `r`, `sim`, `type` and `model`. Note that the data will be stored by row. The field `model` is simply a copy of the structural array `mymodel` used for simulation.

The package contains several MATLAB demos using simulated data, see for instance `demo_mix_normal.m`. Demos using simulated data are all named `demo_xxx.m`.

When plotting simulated data using the function `dataplot`, the true marginal density is added to each histogram. For multivariate data, contours of the marginal bivariate mixture distribution is to each bivariate scatter plot.

# 4

# Statistical Inference for a Finite Mixture Model with Known Number of Components

Statistical inference is concerned with fitting a (finite mixture) model to data. To do so in this toolbox, the data and the model have to be specified. Both the model and the data have to be stored in structural arrays with specific fields, see Section 2.1 and Section 3.1.1, respectively.

## 4.1 Classification for Known Component Parameters

Classification of observations using a fully specified finite mixture model is discussed in Frühwirth-Schnatter (2006, Section 2.2). To obtain for each observation $\mathbf{y}_i$ the probability classification matrix given by Bayes' theorem

$$\Pr(S_i = k|\mathbf{y}_i, \boldsymbol{\vartheta}) = \frac{p(\mathbf{y}_i|\boldsymbol{\theta}_k)\eta_k}{\sum_{j=1}^{K} p(\mathbf{y}_i|\boldsymbol{\theta}_j)\eta_j},$$

call the function

```
class=dataclass(data,mix);
```

where `data` is the structure array containing the data and `mix` is the structure array defining the finite mixture distribution. For classification, `mix` has to be a fully specified model, see Subsection 2.1.3. The function `dataclass` produces a structural array `class` with following fields:

- `prob` is the probability classification matrix, being equal to a `N x K` numerical array, where the rows sum to 1.
- `mixlik` is the logarithm of the mixture likelihood function $\log p(\mathbf{y}|\boldsymbol{\vartheta})$,

$$p(\mathbf{y}|\boldsymbol{\vartheta}) = \prod_{i=1}^{N} \left( \sum_{k=1}^{K} \eta_k p(\mathbf{y}_i|\boldsymbol{\theta}_k) \right),$$

  evaluated at $\boldsymbol{\vartheta}$ equals `mix.par` and `mix.weight`.

- `entropy` is the entropy of the classification, defined by Frühwirth-Schnatter (2006, Subsection 2.2.2, pp. 28)

$$\mathrm{EN}(\boldsymbol{\vartheta}|\mathbf{y}) = -\sum_{i=1}^{N}\sum_{k=1}^{K}\Pr(S_i = k|\mathbf{y}_i, \boldsymbol{\vartheta})\log\Pr(S_i = k|\mathbf{y}_i, \boldsymbol{\vartheta}) \geq 0. \quad (4.1)$$

- `loglikcd` is a `K x 1` numerical array containing the log of the conditional likelihood for each component parameter $\boldsymbol{\theta}_k$. The $k$th element of this array is equal to:

$$\sum_{i:S_i=k}\log p(\mathbf{y}_i|\boldsymbol{\theta}_k).$$

  The conditional likelihood is evaluated at $(\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K)$ equals `mix.par` and $\mathbf{S} = (S_1, \ldots, S_N)$ equals `data.S`.

Note that the name `class` may be substituted by an arbitrary name, e.g. `myclass`.

**Sampling from the classification matrix**

In Section 4.3 a sample $\mathbf{S}^{(m)} = (S_1^{(m)}, \ldots, S_N^{(m)})$ from the probability classification matrix is needed. This is obtained by calling `dataclass` with a second output argument:

```
[class,S]=dataclass(data,mix)
```

The output argument `S` is a `1 x data.N` array containing the simulated allocations. In this case no conditional likelihood is computed, i.e. no field `loglikcd` appears in `class`, but the following new field is added to the structure array `class`:

- `postS` which is equal to the posterior density $p(\mathbf{S}^{(m)}|\mathbf{y}, \boldsymbol{\vartheta})$ of the simulated allocations.

## 4.2 Bayesian Estimation

Statistical inference is concerned with fitting a model to data. Concerning the method used for statistical inference, this toolbox relies on Bayesian estimation which requires the definition of a prior. The hyper parameters of this prior are stored in a structural array called e.g. `myprior`. The precise structure of the array `myprior` depends on the chosen distribution family and usually is a structure array with specific fields which are discussed in more detail in Subsection 4.2.1. Carefully designed default choice are available in this package for any model by calling the function `priordefine` to make Bayesian inference as convenient as possible.

Bayesian inference derives the posterior distribution of the unknown parameters of model `mymodel`, given data `mydata` and the prior `myprior`. In general, one has to distinguish between problems where the posterior distribution is of closed form and problems where this is not the case. Bayesian inference for finite mixture models falls into the second category and relies on Markov chain Monte Carlo (MCMC) inference, see Subsection 4.2.2. Closed form solutions exist only for rare case, e.g. for a finite mixture model with only one component or for fully classified data, where the allocations are known, see Subsection 4.2.3.

### 4.2.1 Choosing the Prior for the Parameter of a Mixture Model

Frühwirth-Schnatter (2006, Section 3.2) discusses in great details how to choose the prior in a mixture model. When fitting a finite mixture model to data, improper prior densities may cause improper mixture posterior densities (Frühwirth-Schnatter, 2006, Subsection 3.2.2). To avoid this, any prior used within the toolbox has to be proper. Error messages and program termination will be the consequence of calling functions with improper priors as input argument.

Users not wanting to specify their own proper prior are recommended to call the function `priordefine` for automatic selection of a slightly data dependent proper prior:

```
prior=priordefine(data,mix);
```

where `data` is a structure array containing the data and `mix` is a structure array defining the mixture distribution which need not be fully specified, only the fields `dist` and `K` are necessary. If the field `K` is missing as well, it is assumed that just a single member from the selected distribution family should be considered. The default prior is invariant to relabeling the components. Details on how this prior is selected will be provided in later chapters for each of the finite mixture models implemented in this toolbox.

Note that it is possible to run the package with this default prior without caring about its structure. Nevertheless some details on this structure are discussed in the remainder of this subsection.

#### Specifying Prior Distributions

In the package, the prior for Bayesian estimation of a particular finite mixture model is stored as a structure array, named for instance `prior`, containing the following fields:

- `weight` specifies the prior for the weight distribution $\boldsymbol{\eta} \sim \mathcal{D}(e_{0,1}, \ldots, e_{0,K})$ which is assumed to be a Dirichlet distribution. This is a `1 x K` numerical array containing the hyper parameters $e_{0,1}, \ldots, e_{0,K}$.

- `par` specifies the prior for each parameter in `mix.par`. The structure of this field depends on the distribution family and on the dimension of $\boldsymbol{\theta}_k$. In general, the field `par` has the same structure as the corresponding field `par` of the structure array defining the mixture distribution.

For a mixture of Poisson distribution, for instance, the prior for the parameter $\mu_k$ reads $\mu_k \sim \mathcal{G}(a_{0,k}, b_{0,k})$. The hyper parameters are stored in `prior.par`. This is again a structural array with the two fields `par.a`, storing $a_{0,1}, \ldots, a_{0,K}$, and `par.b`, storing $b_{0,1}, \ldots, b_{0,K}$. Both fields are `1 x K` numerical arrays.
Additional fields are the following:

- The field `type` specifies the type of the selected prior if various types of priors are implemented in the toolbox, like conditionally conjugate and independence priors for mixtures of normals, see Subsection 6.2.3.

If any of the hyperparameters of an invariant prior is a random parameter with a prior of its own, like $b_0 \sim \mathcal{G}(g_0, G_0)$ in the prior $\mu_k \sim \mathcal{G}(a_0, b_0)$ of a Poisson mixture, then the additional field

- `hier` taking the value `true` is added to the prior specification and additional fields have to be added to the structure array `par`.

For a hierarchical prior for Poisson mixtures, for instance, `par.g` and `par.G` have to be added, containing the parameters $g_0$ and $G_0$ of the Gamma prior $b_{0,k} \sim \mathcal{G}(g_0, G_0)$.

### 4.2.2 Markov Chain Monte Carlo Bayesian Inference

For problems, where no closed form posterior exists, one has to rely on some numerical method to derive the posterior distribution. Finite mixture models typically belong to this class. Nowadays, many researchers rely on Markov chain Monte Carlo (MCMC) methods to obtain draws from the posterior distribution. Contrary to i.i.d. sampling, MCMC sampling starts with an arbitrary starting value, and delivers draws from the posterior distribution only after the so-called burn-in phase.

Bayesian inference using MCMC is carried out by calling the function `mixturemcmc` with four input arguments:

```
mcmcout = mixturemcmc(mydata,mymodel,myprior,mcmc);
```

i.e. the data are stored in `mydata`, the model is stored in `mymodel`, and the prior is stored in `myprior`. The input parameter `mcmc` controls the MCMC sampling procedure and is a structural array with following mandatory fields:

- `burnin` is the length $M_0$ of the burn-in;
- `M` is the number $M$ of stationary draws.

The MCMC draws are stored in the structural array `mcmcout`. Note that the name `mcmcout` could be substituted by any name, e.g. `mymcmcout`. The fields of this array are explained in detail in Section 4.3.2.

### 4.2.3 Closed Form Posterior Distributions

Sometimes a conjugate prior exists, which leads to a posterior distribution from the same distribution family as the prior. For such a conjugate analysis, the function `posterior` may be called to determine the parameters of the posterior distribution:

```
post=posterior(data,model,prior);
```

where the structural array `prior` defines the prior distribution. The procedure returns the parameters of the posterior distribution in `post` which has the same structure as the prior. The function `posterior` checks if a conjugate analysis is possible for this particular model under the given prior. If this is not the case,

- the field `error` is assigned the value `true` and added to the output argument `post`.

Often it is useful to have $M$ random draws from a closed form posterior. These draws could be obtained by first calling the function `posterior` and then calling one of the MATLAB built-in functions which draw from a specific distribution family. Such a strategy, however, is not really recommended. It is far more convenient to call the function `mixturemcmc` even for *conjugate problems*. The function `mixturemcmc` automatically draws from the right posterior distribution even in cases where no MCMC simulation is necessary. Note that the burn in could be set to 0 in such a case.

## 4.3 Parameter Estimation through Data Augmentation and MCMC

Among the estimation methods discussed in Frühwirth-Schnatter (2006, Section 2.4) only Bayesian methods are implemented in the current version of the toolbox for parameter estimation when the allocations are unknown. Bayesian estimation of finite mixtures using data augmentation and MCMC is discussed in great detail in Frühwirth-Schnatter (2006, Section 3.5). MCMC sampling is performed as described in Algorithm 3.4 in Frühwirth-Schnatter (2006, Subsection 3.5.3):

(a) Parameter simulation conditional on a known classification $\mathbf{S}$:
  (a1) Sample $\boldsymbol{\eta} = (\eta_1, \ldots, \eta_K)$ from the Dirichlet distribution $\mathcal{D}(e_1(\mathbf{S}), \ldots, e_K(\mathbf{S}))$.
  (a2) For each $k = 1, \ldots, K$, sample the component parameter $\boldsymbol{\theta}_k$ from the complete-data posterior $p(\boldsymbol{\theta}_k|\mathbf{S}, \mathbf{y})$.

(b) Classification of each observation $\mathbf{y}_i$ conditional on knowing $\boldsymbol{\vartheta}$ by sampling $S_i$ independently for each $i = 1, \ldots, N$ from following discrete distribution:

$$p(S_i = k|\boldsymbol{\vartheta}, \mathbf{y}_i) \propto p(\mathbf{y}_i|\boldsymbol{\theta}_k)\eta_k.$$

Unless stated otherwise (see `mcmc.ranperm` defined in Subsection 4.3.1), each sampling step is concluded by a random permutation step.

It will take several iterations, the so-called burn-in, before the sampler reaches the stationary distribution. After burn-in, $M$ posterior draws $(\boldsymbol{\vartheta}^{(m)}, \mathbf{S}^{(m)}), m = 1, \ldots, M$ are produced by iterating through steps (a) and (b). Both the length of the burn-in as well as the number of draws from the stationary distribution have to be specified by the user, see Subsection 4.3.1. All post burn-in MCMC draws $\boldsymbol{\vartheta}^{(m)}$ are stored in one field of the structural array `mcmcout`. As the dimension of $\mathbf{S}^{(m)}$ is `1 x data.N`, not all draws, but only the last `mcmc.storeS` draws are stored for the allocations, see Subsection 4.3.2.

### 4.3.1 Running MCMC

To run data augmentation and MCMC, call the function

```
mcmcout=mixturemcmc(data,mix,prior,mcmc);
```

where `data` is a structure array containing the data, `mix` is a structure array defining the mixture distribution to be fitted, `prior` is a structure array defining the prior distribution and `mcmc` is a structure array controlling MCMC. Obligatory fields for `mcmc` are:

- `burnin` defining the length $M_0$ of the burn-in;
- `M` defining the number $M$ of stationary draws.

One should be aware that it may take some time to execute MCMC sampling. For the user's convenience, after each minute, the function `mixturemcmc` reports the expected remaining execution time.

MCMC requires the choice of starting values. Usually, a preliminary classification $\mathbf{S}^{(0)}$ is stored in `data.S` and MCMC is started by sampling $\boldsymbol{\vartheta}^{(1)}$ as described in step (a). In this case, the mixture model `mix` need not be fully specified, however, the field `dist` has to be specified in any case. If the field `K` is missing, it is assumed that just a single member from the selected distribution family should be fitted to the data. If sampling of $\boldsymbol{\theta}_k$ involves more than one block, further starting values are needed, which need to be stored in `mix.par` before calling `mixturemcmc`. Furthermore, under a hierarchical prior, starting values have to be selected for the hyper parameters. Under an automatic prior definition using the function `priordefine` introduced in Subsection 4.2.1 such a starting value is automatically provided.

One may delegate the choice of all necessary starting values to the function `mcmcstart` which also defines default choices for all tuning parameters for MCMC:

```
[data,mix,mcmc]=mcmcstart(data,mix);
```

The starting value for the classification is usually obtained through $k$-means clustering using the MATLAB function `kmeans`. The `Warning: Empty cluster`

`created at iteration 1` is sometimes reported by the MATLAB function `kmeans` and may be safely ignored, because the function `mixturemcmc` is able to deal automatically with empty clusters. Further details on how these starting values are selected appear in later chapters for specific mixture models.

The function `mcmcstart` is very convenient, because it allows MCMC for an unspecified mixture model without bothering about starting values. The mixture model may be completely unspecified, only the distribution family (`mix.dist`) of the component density and, if $K > 1$, also `mix.K` have to be specified before calling `mcmcstart`. The function `mcmcstart` defines all necessary starting values needed for MCMC estimation, like starting classifications and starting values for parameters which is very convenient.

The default choice will produce 5000 MCMC draws after a burn-in of 1000 draws and will store the last 500 draws of the allocations. One may easily change this choice by redefining the appropriate fields after calling `mcmcstart`. For instance, to obtain 10000 MCMC draws after a burn-in of 4000 draws and to store the last 1000 draws of the allocations define

```
[data,mix,mcmc]=mcmcstart(data,mix);
mcmc.M=10000;mcmc.burnin=4000;mcmc.storeS=1000;
mcmcout=mixturemcmc(data,mix,prior,mcmc);
```

### Controlling MCMC

`mcmc` is a structure array controlling MCMC having the obligatory fields `burnin` and `M` already described above. Optional fields are the following:

- `startpar` is a logical variable taking the value `true`, if sampling should be started with drawing the allocations $\mathbf{S}$ conditional on a starting value $\boldsymbol{\vartheta}^{(0)}$. The default choice is `startpar=false`.
- `storeS` is an integer variable, causing that the last `storeS` classifications $\mathbf{S}^{(m)}$ to be stored. If the field `storeS` is missing, then the default choice is to store 500 draws. If `storeS` is not positive, then no classifications are stored.
- `storepost` is a logical variable taking the value `true`, if the posterior moments should be stored. The posterior moments are needed to compute the marginal likelihood, see Section 5.3. If `storepost` is not true, then no posterior moments are stored. The default choice is `storepost=true`.
- `ranperm` is a logical variable taking the value `false`, if no random permutation sampling should be performed. If the field `ranperm` is missing or if `ranperm` is true, then random permutation sampling is performed.

The default choice in the package is to start MCMC with an initial classification in which case `startpar=false`. Alternatively, one may select a starting value $\boldsymbol{\vartheta}^{(0)}$ for the parameter and start MCMC by sampling $\mathbf{S}^{(1)}$ as described in step (b). To start MCMC in this way, the mixture model `mix` needs to be fully specified before calling `mixturemcmc`, while `data.S` is unspecified. Again,

the function `mcmcstart` could be used to select starting values for a fully specified mixture model, however, it has to be called with three input arguments to indicate that a starting value for $\boldsymbol{\vartheta}$ is needed rather than a starting value for $\mathbf{S}$:

```
mcmc.startpar=true;
[data,mix,mcmc]=mcmcstart(data,mix,mcmc);
```

### 4.3.2 MCMC Output

`mcmcout` is a structure array containing the MCMC draws and consists of the following fields:

- `M` contains the number of MCMC draws
- `weight` contains the MCMC draws $\boldsymbol{\eta}^{(m)}, m = 1, \ldots, M$ for the weight distribution which are stored in a `M x K` numerical array.
- `par` contains the MCMC draws $(\boldsymbol{\theta}_1^{(m)}, \ldots, \boldsymbol{\theta}_K^{(m)})$, $m = 1, \ldots, M$, for each parameter in `mix.par`. The field `par` has the same structure as the corresponding field `par` of the structure array defining the mixture distribution. For a mixture of Poisson distribution, for instance, `par` is a `M x K` numerical array, storing the posterior draws $\mu_k^{(m)}$, $m = 1, \ldots, M$.
- `ranperm` is a logical variable, which is `true` if the MCMC draws are based on random permutation sampling. Otherwise `ranperm` is false.
- `hyper` is added under a hierarchical prior and contains the MCMC draws for the random hyperparameter.
- `log` stores the logarithm of various function evaluated at the MCMC draws. The field `log` is a structure array containing the following fields, each of them being a `M x 1` numerical array:
  - `mixlik` stores the log of the mixture likelihood, $\log p(\mathbf{y}|\boldsymbol{\vartheta}^{(m)})$, for each MCMC draw $\boldsymbol{\vartheta}^{(m)}$.
  - `mixprior` stores the log of the prior, $\log p(\boldsymbol{\vartheta}^{(m)})$, for each MCMC draw $\boldsymbol{\vartheta}^{(m)}$.
  - `cdpost` stores the log of the (non-normalized) complete data posterior, $\log p(\boldsymbol{\vartheta}^{(m)}, \mathbf{S}^{(m)}|\mathbf{y})$, which is equal to

    $$p(\boldsymbol{\vartheta}^{(m)}, \mathbf{S}^{(m)}|\mathbf{y}) \propto p(\mathbf{y}|\boldsymbol{\vartheta}^{(m)}, \mathbf{S}^{(m)}) p(\mathbf{S}^{(m)}|\boldsymbol{\vartheta}^{(m)}, \mathbf{y}) p(\boldsymbol{\vartheta}^{(m)})$$

    for each MCMC draw $(\boldsymbol{\vartheta}^{(m)}, \mathbf{S}^{(m)})$.
- `entropy` is a `M x 1` numerical array storing the entropy $\mathrm{EN}(\boldsymbol{\vartheta}^{(m)}|\mathbf{y})$, see (4.1), for each MCMC draw.
- `S` is added, if classifications are stored (see `mcmc.storeS` above). The field contains the last `L=mcmc.storeS` MCMC draws of $\mathbf{S}^{(m)}$, stored as a `L x N` numerical array, where `N` are the number of observations.

- `Nk` is added, if posterior moments are stored (see `mcmc.storepost`). This field is a `M x K` numerical array storing the number of observations $N_1(\mathbf{S}), \ldots, N_K(\mathbf{S})$ classified to each group.
- `post` is added, if posterior moments are stored (see `mcmc.storepost` above). These moments are used for computing the marginal likelihood, see Subsection 5.3. `post` is a structure array with the fields `par` and `weight`:
  - `weight` contains the moments $e_1(\mathbf{S}), \ldots, e_K(\mathbf{S})$ of the the posterior Dirichlet distribution $\mathcal{D}(e_1(\mathbf{S}), \ldots, e_K(\mathbf{S}))$ used for simulating the weight distribution $\boldsymbol{\eta}^{(m)}$. `weight` is a `M x K` numerical array.
  - `par` contains certain moments of the complete data posterior distributions $p(\boldsymbol{\theta}_k|\mathbf{S}, \mathbf{y})$, used for simulating the component parameters $(\boldsymbol{\theta}_1^{(m)}, \ldots, \boldsymbol{\theta}_K^{(m)})$.

If $K$ is equal to 1, then a single member from the distribution family `dist` is fitted and redundant fields like `weight`, `S`, `post.weight`, and `ranperm` are not added to `mcmcout`.

Various fields are added which are helpful for postprocessing the MCMC draws:

- `model` contains information about the estimated model and is simply a copy of the calling argument `mix.dist` and `mix.K`.
- `prior` contains the prior used for estimation and is simply a copy of the calling argument `prior`.

You may add the following field to name `mcmcout`:

- `name` which is a character string.

This name will be added to various plots. Use the function

```
mcmcstore(mcmcout);
```

to store the MCMC output. The MCMC output will be stored as a MATLAB file under the name `mcmcout.name`, if such a field is present, and under the name `mcmcout` otherwise.

## 4.4 Parameter Estimation for Known Allocation

In rare cases, for instance for grouped data, the allocations will be known. Such data are called classified data and the allocations are stored in `data.S`, see Subsection 3.1.2.

For a complete-data Bayesian estimation as discussed in Subsection 2.3.3 of Frühwirth-Schnatter (2006), you need first to define a prior on the parameters as in Subsection 4.2.1. For finite mixtures, where a closed form conditional posterior exists, it is possible to compute the moments of the complete-data posterior of the parameters of the finite mixture distribution defined by the

structure array `mix` by calling the functions `posterior`, introduced in Subsection 4.2.3. This produces a structure array `post`, with `par` being the same fields as in the structure array `prior`, see Section 4.2.1.

For practical Bayesian estimation, however, it is more convenient to call the function `mixturemcmc` like for a problem where the allocations are unknown, however, before calling this function,

- the field `indicfix` has to be set equal to `true` and has to be added to the structure array defining the finite mixture model.

This will allow exploring the posterior distribution as in Section 4.5.

## 4.5 Bayesian Inference Using the Posterior Draws

Frühwirth-Schnatter (2006, Section 3.7) discusses in detail how posterior draws could be used for Bayesian inference.

### 4.5.1 Plotting the Posterior Draws

The function

```
mcmcplot(mcmcout);
```

could be used to plot and monitor the MCMC output. The following figures are produced by `mcmcplot`.

#### Trace plots of invariant functionals

A trace plots shows the log mixture likelihood, $\log p(\mathbf{y}|\boldsymbol{\vartheta}^{(m)})$, the log of the prior, $\log p(\boldsymbol{\vartheta}^{(m)})$, the log of the mixture posterior $\log p(\boldsymbol{\vartheta}^{(m)}|\mathbf{y})$, the log of the complete data likelihood, $\log p(\mathbf{y}|\boldsymbol{\vartheta}^{(m)}, \mathbf{S}^{(m)})p(\mathbf{S}^{(m)}|\boldsymbol{\vartheta}^{(m)})$, and the entropy $\mathrm{EN}(\boldsymbol{\vartheta}^{(m)}|\mathbf{y})$ for each MCMC draw over $m = 1, \ldots, M$.

For finite mixture modeling of i.i.d. data additional trace plots are produced that are based on moments of the implied marginal distributions, all of which are invariant to relabelling. Different moments are considered for different distribution families. The moments are computed in the function `mcmcplot` by calling the function `mcmcmargmom`:

```
margmom=mcmcmargmom(mcmcout);
```

which creates a structure array with the same fields as the output from the function `moments`, however, an additional leading dimension is added for each MCMC draw. For multivariate mixtures of normals, for instance, the variance-covariance matrix is stored in the field `var` which a numeric array of dimension `M x r x r`.

### Trace plots of component specific parameters

Component specific parameters are sensitive to labeling switching. If no random permutation sampling has been performed (depending on the tuning parameter `mcmc.ranperm` introduced in Subsection 4.3.1), then trace plot are produced directly for the MCMC draws $\boldsymbol{\vartheta}^{(m)}$. Depending on underlying model, the trace plots are organized in several figures.

If random permutation sampling has been performed (`mcmc.ranperm=true`), then the MCMC draws are identical for all components and direct plotting of the MCMC draws is not really useful. Therefore, the function `mcmcplot` performs model identification prior to plotting using the function `mcmcpermute` described in Subsection 4.5.2. Plots are then produced for the identified MCMC draws. Note that the number of identified MCMC draws could be considerable smaller than the number of total MCMC draws.

### Checking Convergence

The trace plots provided by `mcmcplot` should be roughly stationary. If this is not the case, MCMC should be run again with a longer burn-in period. Alternatively, the first draws could be removed. The function `mcmcsubseq`

```
mcmcsub=mcmcsubseq(mcmcout,indexset);
```

could be used to extract a subsequence of the MCMC draws, defined by the index set `indexset`. To remove the first `it0` draws, for instance, use

```
mcmcsub=mcmcsubseq(mcmcout,[it0+1:mcmcout.M]);
```

To perform spacing, meaning that only every $p$th value should be stored, use

```
p=10;mcmcsub=mcmcsubseq(mcmcout,[1:p:mcmcout.M]);
```

### Sampling Representation of the Mixture Posterior Density

It is sometimes desired to visualize the mixture posterior density $p(\boldsymbol{\vartheta}|\mathbf{y})$, but producing a simple density plot is feasible only for very simple problems, where the unknown parameter $\boldsymbol{\vartheta}$ is at most bivariate. If the dimension of $\boldsymbol{\vartheta}$ exceeds two, draws from the posterior density $p(\boldsymbol{\vartheta}|\mathbf{y})$ are used as a sampling representation of the mixture posterior distribution, which is then visualized in an appropriate manner, see Frühwirth-Schnatter (2006, Subsection 3.7.1).

The function `mcmcmplot` provides such a sampling representation of the posterior draws by calling the function `mcmcsamrep` which is defined with variable input/output argument, handling figure numbers:

```
mcmcsamrep(mcmcout); % starts plotting with Figure 1
mcmcsamrep(mcmcout,nplot); % starts plotting with Figure nplot
nplot=mcmcsamrep(mcmcout,nplot); % returns number of last Figure
```

The function `mcmcsamrep` produces a point process representation of the posterior draws. For mixtures with univariate component parameter $\theta$, $\theta_k^{(m)}$ is plotted against draws from a standard normal distribution. For mixtures with bivariate component specific parameter $\boldsymbol{\theta} = (\theta_1, \theta_2)$, $\theta_{1,k}^{(m)}$ is plotted against $\theta_{2,k}^{(m)}$. For mixtures with multivariate components parameters $\boldsymbol{\theta}$, a specific point process representation is generated for each type of mixture models.

These scatter plots are closely related to the point process representation of the underlying mixture distribution discussed in Subsection 2.2.5. The MCMC draws will scatter around the points corresponding to the true point process representation, with the spread of the clouds representing the uncertainty of estimating the points.

The number of simulations clusters visible in these MCMC draws are helpful for mixtures with unknown number of components, see Subsection 5.1.

### 4.5.2 Estimating the Component Parameters and the Weight Distribution

Inference on the component parameters and the weight distribution is sensitive to label switching, see Frühwirth-Schnatter (2006, Subsection 3.7.6).

The posterior mode is that value of $\boldsymbol{\vartheta}$ which maximizes the nonnormalized mixture posterior density $\log p^\star(\boldsymbol{\vartheta}|\mathbf{y}) = \log p(\mathbf{y}|\boldsymbol{\vartheta}) + \log p(\boldsymbol{\vartheta})$. The posterior mode estimator is the optimal estimator with respect to the 0/1 loss function and is invariant to relabeling. Because $\log p(\mathbf{y}|\boldsymbol{\vartheta}^{(m)})$ and $\log p(\boldsymbol{\vartheta}^{(m)})$ are contained in the MCMC output produced by `mixturemcmc`, the posterior mode may be approximated by the MCMC draws with the largest value of $\log p^\star(\boldsymbol{\vartheta}|\mathbf{y})$. An approximate ML estimator is derived in a similar way.

Ergodic averages of MCMC draws which were generated by unconstrained Gibbs sampling without identification may be sensible to label switching and should be interpreted with great care. Ergodic averages of MCMC draws which were generated by random permutation sampling theoretically are invariant and could be used to check convergence. In the toolbox, model identification based on unsupervised clustering in the point process representation (Frühwirth-Schnatter, 2006, p.96) is performed.

To perform parameter estimation, call the function `mcmcestimate` with a structure array, say `mcmcout` containing the MCMC output after calling the function `mixturemcmc`:

```
est=mcmcestimate(mcmcout);
```

`est` is a structure array containing following estimators of $\boldsymbol{\vartheta}$ with the following fields:

- `pm` is the (approximate) posterior mode estimator of $\boldsymbol{\vartheta}$.
- `ml` is the (approximate) maximum likelihood estimator of $\boldsymbol{\vartheta}$.
- `ident` ergodic average after identification
- `average` ergodic average without identification, if the draws were not generated by the permutation sampler (`mcmcout.ranperm` is `false`).
- `invariant` ergodic average without identification, if the draws were generated by the permutation sampler (`mcmcout.ranperm` is `true`).

Each of these fields is a *fitted mixture* meaning that it is a structure array with the same fields as a mixture model. Therefore many functions, like `mixtureplot` could be applied. To compare the data with the model, you may store a fitted mixture, say `pm` in `data.model` before calling `dataplot`.

If `mcmcestimate` is called with two output arguments,

```
[est,mcmcout]=mcmcestimate(mcmcout);
```

then the estimators and the identified MCMC output will be added to the MCMC output `mcmcout`.

For each estimation method, the estimators of the weight distribution $\eta_1, \ldots, \eta_K$ are stored in the field `weight`, e.g.

- `est.pm.weight` – (approximate) posterior mode estimator,
- `est.ident.weight` – ergodic average after identification.

For each estimation methods, the estimators of the parameters are stored in the field `par`, which has the same structure as for the estimated model.

### Model Identification

Identification is based on unsupervised clustering in the point process representation (Frühwirth-Schnatter, 2006, p.96) and only posterior draws where the resulting classification is a permutation of the group indices are considered. Identification is achieved by calling the function `mcmcpermute`:

```
mcmcout=mcmcpermute(mcmcout);
```

After calling `mcmcpermute` the following fields are added to the MCMC output:

- `perm` is a `MxK` array, containing the classifications resulting from unsupervised clustering
- `isperm` is a `Mx1` logical array, being true iff the classification of the $m$th draw is a permutation
- `nonperm` contains the number of classifications that are not permutations
- `Mperm` contains the number of classifications that are permutations
- `parperm` and `weightperm` contain the identified MCMC output. The field `weightperm` contains the identified MCMC draws for the weight distribution which are stored in a `Mperm x K` numerical array. `parperm` contains the identified MCMC draws for each parameter in `mix.par` and has the

same structure as the corresponding field `par` of the MCMC output, the only difference being that the first dimension is equal to `Mperm`, which might be smaller than `M`.

### 4.5.3 Bayesian Clustering

Model-based clustering using finite mixture models is discussed in great detail in Frühwirth-Schnatter (2006, Section 7.1). Bayesian maximum a posteriori (MAP) classification is based on maximizing the joint posterior

$$p(\boldsymbol{\vartheta}, \mathbf{S}|\mathbf{y}) \propto p(\mathbf{y}|\boldsymbol{\vartheta}, \mathbf{S})p(\mathbf{S}|\boldsymbol{\vartheta})p(\boldsymbol{\vartheta}), \tag{4.2}$$

simultaneously with respect to $\boldsymbol{\vartheta}$ and $\mathbf{S}$, where $p(\mathbf{y}|\boldsymbol{\vartheta}, \mathbf{S})p(\mathbf{S}|\boldsymbol{\vartheta})$ is equal to the classification likelihood, see Frühwirth-Schnatter (2006, Subsection 7.1.3, p.210). An approximation to the Bayesian MAP classifier is determined during data augmentation and Gibbs sampling by evaluating the nonnormalized posterior $p(\boldsymbol{\vartheta}^{(m)}, \mathbf{S}^{(m)}|\mathbf{y})$ for each MCMC draw, and keeping track of the classification that gave the highest posterior density. Note that the Bayesian MAP classifier is invariant to label switching.

Bayesian clustering could be based on loss functions, as discussed in Frühwirth-Schnatter (2006, Subsection 7.1.7). Bayesian clustering based on the misclassification rate is sensitive to label switching and can be carried out only after the mixture has been identified, see also Subsection 4.5.2. This may be a poor estimator, if the mixture model is not identifiable, for instance, because the mixture is overfitting.

An additional estimator based on the posterior similarity matrix is insensitive to label switching, however, it is of order $\mathcal{O}(N)$ and it may take some time to obtain this estimator, if $N$ is large. In the toolbox, $N$ is currently limited to 1000.

To carry out clustering of the observations call the function `mcmcclust`:

```
clust=mcmcclust(data,mcmcout)
```

The structure array `clust` contains various estimators of the allocations stored in the following fields:

- `Smap` is a `1 x N` array containing the (approximate) Bayesian MAP classification.
- `Ssim` is a `1 x N` array containing the estimator based on the posterior similarity matrix and is computed only, if $N \leq 1000$.
- `Sident` is a `1 x N` array containing the estimator minimizing the misclassification rate, which is determined after identification.
- `prob` is a `K x N` array containing the corresponding classification probability matrix, i.e. $\Pr(S_i = k|\mathbf{y})$, $i = 1, \ldots, N$, $k = 1, \ldots, K$.
- `risk` is a `1 x N` array containing the corresponding misclassification rate, i.e. $1 - \sum_{k=1}^{K} \Pr(S_i = k|\mathbf{y})$, $i = 1, \ldots, N$.

If `mcmcclust` is called with two output arguments,

```
[clust, mcmcout]=mcmcclust(data,mcmcout);
```

then the field `clust` having the same structure as above will be added to the MCMC output `mcmcout`.

### Visualizing Bayesian Clustering

To visualize clustering and the estimated probabilities $\Pr(S_i = k|\mathbf{y})$, call the function `mcmcclustplot` in the following way

```
[nfig=]mcmcclustplot(data,clust[,nfig]);
```

where `clust` is the output from calling the function `mcmcclust`. Plotting starts with the input figure number `nfig`, or with figure one, if the input argument `nfig` is missing. The output argument `nfig` reports the number of the last figure and may be omitted.

For univariate data, this function produces a plot of the estimated classification probabilities $\Pr(S_i = k|\mathbf{y}), i = 1, \ldots, N$ and a plot showing the clustering of the data for the different estimators of $\mathbf{S}$ stored in `clust`. For multivariate data, this function produces a plot showing the clustering of the data for each estimator of $\mathbf{S}$ stored in `clust`.

## 4.5.4 Predictive Density Estimation

A quantity that often is of interest when fitting a finite mixture model, is the posterior predictive density $p(\mathbf{y}_f|\mathbf{y})$ of a future realization $\mathbf{y}_f$, given the data $\mathbf{y}$, which is given by

$$p(\mathbf{y}_f|\mathbf{y}) = \int p(\mathbf{y}_f|\boldsymbol{\vartheta})p(\boldsymbol{\vartheta}|\mathbf{y})d\boldsymbol{\vartheta}.$$

This density is estimated from posterior draws which need not be checked for label switching as:

$$\hat{p}(\mathbf{y}_f|\mathbf{y}) = \frac{1}{M}\sum_{m=1}^{M}\left(\sum_{k=1}^{K}\eta_k^{(m)}p(\mathbf{y}_f|\boldsymbol{\theta}_k^{(m)})\right), \tag{4.3}$$

is robust against label switching. To plot the predictive density estimation use the function `mcmcpreddens` which is defined with variable input/output argument, handling figure numbers:

```
[nplot=]mcmcpreddens(data,mcmcout[,nplot]);
```

Plotting starts with the input figure number `nfig`, or with figure one, if the input argument `nfig` is missing. The output argument `nfig` reports the number of the last figure and may be omitted.

### The Posterior Predictive Distribution of a Sequence

To implement Algorithm 3.7 in Frühwirth-Schnatter (2006, Subsection 3.7.3, pp.90) to sample for each MCMC draw in the structure array `mcmcout` a sequence $\mathbf{y}_f^{(m)} = (\mathbf{y}_{f,1}^{(m)}, \ldots, \mathbf{y}_{f,H}^{(m)})$ of length $H \geq 1$ from the posterior predictive distribution $p(\mathbf{y}_f|\mathbf{y})$ of $\mathbf{y}_f$, conditional on the observations $\mathbf{y}$ call the function

```
pred=mcmcpredsam(mcmcout,H);
```

For univariate data `pred` is a `M x H` numerical array, containing the predicted sample, i.e. `pred(m,h)` contains the $h$th observation $y_{f,h}^{(m)}$. For multivariate data `pred` is a `M x r x H` numerical array, containing the predicted sample, where `r` is the dimension of $\mathbf{y}_i$ and `pred(m,:,h)` contains the $h$th observation $\mathbf{y}_{f,h}^{(m)}$.

# 5

# Statistical Inference for Finite Mixture Models Under Model Specification Uncertainty

## 5.1 Mode Hunting in the Mixture Posterior

A couple of informal methods for identifying the number of components are discussed in Frühwirth-Schnatter (2006, Section 4.3). To implement mode hunting in the mixture posterior density as in Frühwirth-Schnatter (2006, Subsection 4.3.1), use the function `mcmcsamrep`, see Subsection 4.5.1.

This function produces point process representation of the posterior draws. If the finite mixture distribution is not overfitting, then $K$ simulation clusters should be present in these figures. If the finite mixture distribution is overfitting, then fewer simulations clusters are present, and a mixture with less components should be fitted to the data. However, some care must be exercised with this interpretation in higher dimensions.

## 5.2 Diagnosing Mixtures Through the Method of Moments and Through Predictive Methods

It is often useful to diagnose mixtures through the method of moments as in Subsection 4.3.3 or through predictive methods as in Subsection 4.3.4 in Frühwirth-Schnatter (2006). For this purpose, a function called `mcmcdiag` is included in the package, which produces various diagnostic plots for the comparison of more than one model. The function may be called simultaneously for more than one MCMC output, in order to compare the different models:

```
[nfig=]mcmcdiag(data,mcmcout1,...,mcmcoutK[,nfig]);
```

where `data` are the data, and `mcmcout1,...,mcmcoutK` is an arbitrary number of structure arrays containing the MCMC output of a certain model. Plotting starts with the input figure number `nfig`, or with figure one, if the input argument `nfig` is missing. The output argument `nfig` reports the number of the last figure and may be omitted.

Several figures compare the posterior distribution of moments of the marginal distribution for the different models. These moments, like the coefficient of determination, may include group specific information as long as the resulting moment is invariant to relabelling. The moments of the marginal distribution of 500 mixture models randomly selected from the MCMC output are computed using the function `mcmcmargmom`:

```
im=randperm(mcmcout.M);
postmom=mcmcmargmom(mcmcsubseq(mcmcout,im(1:500)));
```

which creates a structure array called, for instance, `postmom` with the same fields as the output from the function `moments`, however, an additional leading dimension is added for each selected MCMC draw.

Further plots are standard diagnostic predictive checks, based on the standard sample moments included in the function `datamoments`. The predictive diagnostic checks depend on the nature of the fitted mixture.

For univariate continuous mixtures these predictive diagnostic checks are based on the mean, the variance, the skewness and the kurtosis coefficient. For multivariate continuous mixtures these predictive diagnostic checks are based on the mean, the variance, the skewness and the kurtosis coefficient of each marginal distribution as well as on the correlation coefficients between any two features. For discrete mixture these predictive diagnostic checks are based on the mean, the variance, the overdispersion parameter, the fraction of zeros in the sample and the first up to the fourth factorial moment.

The box plots are based on drawing 200 predictive samples of size $N$. The moments of each predictive sample are computed using the function `mcmcpredmom`:

```
im=randperm(mcmcout.M);
predmom=mcmcpredmom(mcmcsubseq(mcmcout,im(1:200)),data.N,data);
```

which creates a structure array called, for instance, `predmom` with the same fields as the output from the function `datamoments`, however, an additional leading dimension is added for each selected MCMC draw.

### Diagnostic Check Based on an Arbitrary Statistic

To design a diagnostic check based on an arbitrary statistic, say $T(\mathbf{y}_f)$, one has to generate a sample $\mathbf{y}_f^{(1)}, \ldots, \mathbf{y}_f^{(M)}$ from the posterior predictive distribution $p(\mathbf{y}_f|\mathbf{y}, \mathcal{M}_K)$, obtained by Algorithm 3.7 in Frühwirth-Schnatter (2006, Subsection 3.7.3, pp.90) with $H = N$, calling the function `mcmcpredsam(mcmcout,N)`, see also Subsection 4.5.4. Then the statistic $T(\mathbf{y}_f)$ has to be computed for each sample `pred(m,:)` for univariate data, or for each sample `pred(m,:,:)` for multivariate data. The resulting sequence of statistics is then compared with the observed statistics through a histogram or a density plot.

The following code shows how the right-hand side of Figure 4.11 in Frühwirth-Schnatter (2006) has been produced from the MCMC output `mcmcout`:

```
pred=mcmcpredsam(mcmcout,N);
over=var(pred',1)'-mean(pred,2);
overdata=var(data.y,1)'-mean(data.y); % data stored by column
plotdichte(over,'k');
hold on;scatter(overdata,0,50,'k','filled');hold off;
```

## 5.3 Simulation-Based Approximations of the Marginal Likelihood

To implement the material of Frühwirth-Schnatter (2006, Section 5.4), call the function

```
marlik = mcmcbf(data,mcmcout);
```

The structure array `marlik` contains various estimators of the log of the marginal likelihood $\log p(\mathbf{y})$ stored in the following fields:

- `is` is the estimator obtained by importance sampling, see Frühwirth-Schnatter (2006, Subsection 5.4.3);
- `ri` is the estimator obtained by reciprocal importance sampling, see Frühwirth-Schnatter (2006, Subsection 5.4.4);
- `bs` is the estimator obtained by bridge sampling techniques, see Frühwirth-Schnatter (2006, Subsection 5.4.6).

Standard errors are computed as in Chib (1995) and stored in the field `se`:

- `se` is structural array with following fields:
  - `se.bs` is a 1x3 array. `se.bs(1)` contains the standard error of the bridge sampling estimator, `se.bs(2)` contains the standard error of the numerator and `se.bs(3)` contains the standard error of the denominator.
  - `se.is` is a scalar containing the standard error of the importance sampling estimator.
  - `se.ri` is a scalar containing the standard error of the reciprocal importance sampling estimator.

If `mcmcbf` is called with two output arguments,

```
[marlik, mcmcout]=mcmcbf(data,mcmcout);
```

then the field `marlik` having the same structure as above will be added to the MCMC output `mcmcout`.

### 5.3.1 Getting Started Quickly

Several demos are included in the package to demonstrate how to select the number of components through marginal likelihoods. Among them are the following:

- The program `start_fishery.m` may be called to fit finite mixtures of multivariate normal distributions with $K = 1$ to $K = 5$ to Fishery Data and to select the number of components through marginal likelihoods (takes about 11 CPU minutes), see also Subsection 1.2.1.
- The program `start_iris.m` may be called to fit finite mixtures of multivariate normal distributions with $K = 1$ to $K = 5$ to Fisher's Iris Data and to select the number of components through marginal likelihoods (takes about 11 CPU minutes), see also Subsection 1.2.2.

Further demonstrations appear in Subsection 6.2.10, and Subsection 6.3.6.

### 5.3.2 Comparing the Estimators

The different estimators should roughly agree, if $K$ is not too large. Significant differences between `marlik.ri` and the other two estimators may be a sign of nonstationarity in the underlying MCMC draws and a poor choice of the importance density.

Note that `marlik.is` is not sensitive to nonstationarity in the underlying MCMC draws, as long as the tails are not too thin, because the MCMC draws are only used to construct the importance density. Checking convergence of the MCMC draws using the function `mcmcplot` described in Subsection 4.5.1 may be helpful. If the entire MCMC chain is not stationary, an MCMC subchain may be constructed using the function `mcmcsubseq`, see Subsection 4.5.1, before the estimators are determined by the function `mcmcbf`:

```
mcmcsub=mcmcsubseq(mcmcout,[100:mcmcout.M]);
marliksub = mcmcbf(data,mcmcsub);
```

To perform further monitoring in cases where the estimators are extremely different, the relevant functional values are stored in the field `log` of structure array `marlik`. The field `log` is a structural array containing the following fields each of which is a `M x 1` numerical array:

- `loglikmc` contains the log mixture likelihood function evaluated at the MCMC draws
- `priormc` contains the log mixture prior evaluated at the MCMC draws
- `qmc` contains the log of the importance density evaluated at the MCMC draws
- `loglikq` contains the log mixture likelihood function evaluated at the draws from the importance density
- `priorq` contains the log mixture prior evaluated at the draws from the importance density

- `qq` contains the log of the importance density evaluated at the draws from the importance density

The function `mcmcbfplot` may be called to plot the functional evaluations both for the MCMC draws as well as for the draws of the importance density, after having added to the MCMC output as described above:

```
[ifig=]mcmcbfplot(mcmcout[,ifig]);
```

This will produce one figure with a trace plot of all functional evaluations both for the MCMC draws as well as for the draws of the importance density. The MCMC plots should be checked for further signs of non-stationarity of the MCMC draws.

A second plot compares a histogram of the functional evaluation of the mixture prior, the mixture likelihood and the importance density both for the MCMC draws as well as for the draws of the importance density. These histograms should roughly agree. Differences may occur for the histogram of the functional evaluations of the importance density. If this histogram is far less spread out to the left hand side for the MCMC than for the draws from the importance density, then the tails of the importance density are very fat compared to the posterior and reciprocal importance sampling is likely to be unstable. If this histogram is much more spread out to the left hand side for the MCMC than for the draws from the importance density, then the tails of the importance density are very thin compared to the posterior and importance sampling is likely to be unstable.

### 5.3.3 Technical Details

The importance density $q(\boldsymbol{\vartheta})$ is constructed from the $S$ randomly selected MCMC draws, stored in `mcmcout`, as in formula (5.36) in Frühwirth-Schnatter (2006, Subsection 5.4.2) with $S = \min(M_0 K!, S_{\max}, M)$, where $K$ is the number of components in the mixture, $M$ is the number of MCMC draws stored in `mcmcout`, $S_{\max}$ is an upper limit for $S$, and $M_0$ is the expected number of times, that the construction of $q$ will be based on a particular mode of the mixture posterior.

The function `mcmcbf` selects default values for $M_0$ and $S_{\max}$, namely $M_0 = 100$ if $K \leq 3$ and $M_0 = 5$, otherwise, and $S_{\max} = 2000$. To control these parameters, `mcmcbf` may be called with three input arguments:

```
[marlik, mcmcout]=mcmcbf(data,mcmcout,options);
```

where `options` is a structure array with following optional fields:

- `M0` being the value chosen for $M_0$;
- `Smax` being the value chosen for $S_{\max}$.

Every minute, the function `mcmcbf` provides an estimator for the remaining expected execution time. Simulation-based estimators are based on functional evaluations and their computation may be rather time consuming if based on a very long MCMC chain. If the estimated execution time is too long, the estimators could be based on a subsequence of the MCMC draws, which is extracted from the entire MCMC chain using the function `mcmcsubseq`, see Subsection 4.5.1.

#### Functional Evaluations

To compute marginal likelihoods it is necessary to evaluate certain functions. To evaluate the prior $p(\boldsymbol{\vartheta})$ first the parameters is assigned to the fields `par` and `weight` of a mixture model with fields `K` and `dist` being equal to $K$ and the distribution family, respectively, see also Section 2.1. Then the function

```
logprior=prioreval(model,prior);
```

is called where `prior` is a structural array defining the prior. The functional value $\log p(\boldsymbol{\vartheta})$ is returned in `logprior`. To evaluate the likelihood function $p(\mathbf{y}|\boldsymbol{\vartheta})$ first the parameters is assigned to the fields `par` and `weight` of a mixture model with fields `K` and `dist` being equal to $K$ and the distribution family, respectively, see also Section 2.1. Then the function

```
loglik=likelihoodeval(data,model);
```

is called where `data` is a structural array containing the data. The functional value $\log p(\mathbf{y}|\boldsymbol{\vartheta})$ is returned in `loglik`.

## 5.4 Model Choice Criteria

Common model choice criteria are AIC, BIC, and different classification-based information criteria (Frühwirth-Schnatter, 2006, Section 4.4.2, 7.1.4) which are minimized for the optimal model among a set of potential models.

Any of these criteria should be based on the maximum likelihood estimator. In the current version of the toolbox these criteria are either evaluated at the approximate posterior mode estimator or the approximate ML estimator. These approximate estimators are obtained by maximizing the log mixture likelihood function or the log posterior density over the MCMC draws. The corresponding functional values are stored in `mcmcout.log.mixlik` and `mcmcout.log.mixprior`, see Subsection 4.3.2.

To compute these criteria from the MCMC output, call the function `mcmcic`:

```
ic=mcmcic(data,mcmcout)
```

The structure array `ic` contains various model choice criteria stored in the following field:

- `aic` contains the AIC criterion evaluated at the approximate ML estimator.
- `bic` contains the BIC criterion evaluated at the approximate ML estimator.
- `bicpm` contains the BIC criterion evaluated at the approximate posterior mode estimator.
- `iclbic_ml` contains the entropy corrected BIC criterion evaluated at the approximate ML estimator, see Frühwirth-Schnatter (2006, Subsection 7.1.4, p.215).
- `iclbic_pm` contains the entropy corrected BIC criterion evaluated at the approximate posterior mode estimator.
- `loglikmax` contains the (approximate) maximum of the log mixture likelihood function.
- `d` contains the number of parameters in the model.

If `mcmcic` is called with two output arguments,

```
[ic,mcmcout]=mcmcic(data,mcmcout);
```

then the field `ic` having the same structure as above is added to the MCMC output `mcmcout`.

# 6

# Finite Mixture Models for Continuous Data

## 6.1 Data Structures

Data for which continuous mixtures are fitted should be defined as a structure array, called e.g. `data` as described in Subsection 3.1.1. The field `type{j}='continuous'` should be added for each feature, because this automatically allows many additional options for these data.

### Data Sets Available in the Package

For illustration, several data sets are stored under particular names and could be loaded into a structure array using the function `dataget`:

- Fishery Data: `data=dataget('fishery')`
- Fisher's Iris Data: `data=dataget('iris')`

## 6.2 Finite Mixtures of Normal Distributions

Finite mixtures of normal distributions are defined and discussed in detail in Frühwirth-Schnatter (2006, Section 6.1). For univariate observations, it is assumed that the observations $\mathbf{y} = (y_1, \ldots, y_N)$ are independent realization of a random variable $Y$ arising from a following mixture of normal distributions:

$$p(y|\boldsymbol{\vartheta}) = \eta_1 f_N(y; \mu_1, \sigma_1^2) + \cdots + \eta_K f_N(y; \mu_K, \sigma_K^2),$$

with $f_N(y; \mu_k, \sigma_k^2)$ being the density of a univariate normal distribution. For multivariate observations, it is assumed that the observations $\mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_N)$ are independent realization of a random variable $\mathbf{Y}$ arising from a following mixture of multivariate normal distributions:

$$p(\mathbf{y}|\boldsymbol{\vartheta}) = \eta_1 f_N(\mathbf{y}; \boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1) + \cdots + \eta_K f_N(\mathbf{y}; \boldsymbol{\mu}_K, \boldsymbol{\Sigma}_K),$$

with $f_N(\mathbf{y}; \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ being the density of a multivariate normal distribution with mean $\boldsymbol{\mu}_k$ and variance–covariance matrix $\boldsymbol{\Sigma}_k$.

### 6.2.1 Defining Mixtures of Normal Distributions

To define a finite mixture of normal distributions within this toolbox, create a structure array as explained in Subsection 2.2.1. The field `dist` is equal to `'Normal'` for univariate mixtures and equal to `'Normult'` for multivariate mixtures. The field `par` is again a structure array with two fields:

- `mu` contains the component means;
- `sigma` contains the component (co)variances.

For a univariate mixture, both `par.mu` and `par.sigma` are `1 x K` numeric arrays. For a multivariate mixture of dimension `r`, `par.mu` is a `r x K` numeric array and `par.sigma` is `r x r x K` numeric array.

For a mixture of $K$ multivariate normals it is often convenient to work with $\boldsymbol{\Sigma}_k^{-1}$ and $\log|\boldsymbol{\Sigma}_k^{-1}|$, e.g. if the mixture is applied for classification. These quantities are stored as optional fields of `par`, where

- `sigmainv` contains $\boldsymbol{\Sigma}_1^{-1}, \ldots, \boldsymbol{\Sigma}_K^{-1}$ , characterized by a `r x r x K` numeric array;
- `logdet` contains $\log|\boldsymbol{\Sigma}_1^{-1}|, \ldots, \log|\boldsymbol{\Sigma}_K^{-1}|$, characterized by a `1 x K` numeric array.

Table 6.1 summarizes, how the component parameter and the weights are accessed.

Whereas it is mathematically correct to consider a univariate normal mixture as that special case of a multivariate mixture of normals where $r = 1$, a combination of `'Normult'` with `mix.r=1` should be avoided when using this package, because this almost certainly leads to troubles with array sizes when the intrinsic function `squeeze` is applied by one the routines.

**Table 6.1.** Accessing the parameters in a mixture of normals called `mix`

| | | |
|---|---|---|
| $\boldsymbol{\eta}$ | `mix.weight` | `1 x K` |
| $\eta_k$ | `mix.weight(k)` | scalar |
| $\mu_k$ | `mix.par.mu(k)` | scalar |
| $\sigma_k^2$ | `mix.par.sigma(k)` | scalar |
| $\boldsymbol{\mu}_k$ | `mix.par.mu(:,k)` | `r x 1` |
| $\boldsymbol{\Sigma}_k$ | `squeeze(mix.par.sigma(:,:,k))` | `r x r` |
| $\boldsymbol{\Sigma}_k^{-1}$ | `squeeze(mix.par.sigmainv(:,:,k))` | `r x r` |
| $\log\|\boldsymbol{\Sigma}_k^{-1}\|$ | `mix.par.logdet(k)` | scalar |

### 6.2.2 Getting Started Quickly

Several demos are included in the package to demonstrate how to fit mixtures of normals to simulated and real data, see Subsection 3.3 for details on how to simulate data from a finite mixture distribution:

- `start_fishery_K4.m`: fits a finite mixture of four univariate normal distributions to the FISHERY DATA, see also Subsection 1.2.1 (takes about 2 CPU minutes).
- `start_fishery.m`: fits finite mixtures of univariate normal distributions with $K = 1$ to $K = 5$ to the FISHERY DATA (takes about 11 CPU minutes), see also Subsection 1.2.1.
- `start_iris_K3.m`: fits a finite mixture of three multivariate normal distributions to FISHER'S IRIS DATA (takes about 3 CPU minutes), see also Subsection 1.2.2.
- `start_iris.m`: fits finite mixtures of multivariate normal distributions with $K = 1$ to $K = 5$ to FISHER'S IRIS DATA (takes about 11 CPU minutes), see also Subsection 1.2.2.
- `demo_mix_normal.m`: fits a finite mixture of three normal distributions to simulated data (takes about 4 CPU minutes).
- `demo_mix_normal_Kunknown.m`: fits finite mixtures with increasing number of components to simulated data (takes about 4 CPU minutes).
- `demo_mix_multivariate_normal.m`: fits a finite mixture of three bivariate normal distributions to bivariate simulated data (takes about 5 CPU minutes).
- `demo_mix_multivariate_normal_Kunknown.m`: fits finite mixtures of bivariate normal distributions with increasing number of components to bivariate simulated data (takes about 8 CPU minutes), see also Subsection 6.2.10.

### 6.2.3 Choosing the Prior Distribution for Univariate Mixtures of Normals

Bayesian estimation of univariate mixtures of normals is discussed in great detail in Frühwirth-Schnatter (2006, Section 6.2). The choice of prior distributions for mixtures of normal distributions is discussed in Subsections 6.2.2, 6.2.3 and 6.2.6 of Frühwirth-Schnatter (2006). Both conditionally conjugate as well as independence priors are implemented for mixtures of normals in the toolbox.

#### The Structure of the Prior

The prior has to be a structure array as explained in Subsection 4.2.1, including the following fields:

- `type` specifies the prior type. This is one of the following strings:
  - `'concon'` refers to the conditionally conjugate prior

$$\mu_k|\sigma_k^2 \sim \mathcal{N}\left(b_{0,k}, \sigma_k^2/N_{0,k}\right), \qquad \sigma_k^2 \sim \mathcal{G}^{-1}\left(c_{0,k}, C_{0,k}\right). \tag{6.1}$$

  - `'indep'` refers to the independence prior

$$\mu_k \sim \mathcal{N}\left(b_{0,k}, B_{0,k}\right), \qquad \sigma_k^2 \sim \mathcal{G}^{-1}\left(c_{0,k}, C_{0,k}\right). \tag{6.2}$$

- The field `par` contains the hyperparameters and is a structure array with the fields `mu` and `sigma`, respectively:
  - For the conditionally conjugate prior the field `mu` is a structure array with the fields `b` and `NO`, being `1 x K` numerical arrays specifying the parameters $b_{0,k}$ and $N_{0,k}$ of the normal prior (6.1).
  - For the independence prior the field `mu` is a structure array with the fields `b` and `Binv`, being `1 x K` numerical arrays specifying the parameters $b_{0,k}$ and $B_{0,k}^{-1}$ of the normal prior (6.2).
  - For both priors, the field `sigma` is a structure array with the fields `c` and `C`, being `1 x K` numerical arrays specifying the parameters $c_{0,k}$ and $C_{0,k}$ of the prior $\sigma_k^2 \sim \mathcal{G}^{-1}(c_{0,k}, C_{0,k})$.

Note that the arrays defining these fields contain $K$ entries, even if the prior is invariant.

If the hyperparameter $C_0$ of an invariant prior is a random parameter with a prior of its own, $C_0 \sim \mathcal{G}(g_0, G_0)$, then the following additional fields have to be added to the prior specification:

- The field `hier` which is a logical variable taking the value `true`.
- The fields `par.sigma.g` and `par.sigma.G` containing the parameters $g_0$ and $G_0$ of the Gamma prior.

### The Default Choice

The toolbox allows an automatic selection of slightly data dependent, rather noninformative priors by calling the function `priordefine`. This default choice is an invariant hierarchical independence prior (Richardson and Green, 1997) with the hyperparameters selected as in Frühwirth-Schnatter (2006, Subsection 6.2.6):

$$\begin{aligned} b_0 &= m, \qquad B_0 = R^2, \qquad c_0 = 2, \\ g_0 &= 0.5, \qquad G_0 = 100 g_0/(c_0 R^2), \end{aligned} \tag{6.3}$$

where $m$ and $R$ are the midpoint and the length of the observation interval. We choose $g_0 = 0.5$ rather than $g_0 = 0.2$ as in Richardson and Green (1997) for numerical reasons.

It is possible to force the package to use a standard independence prior where $C_0$ is fixed rather than random. This prior is obtained by calling the function `priordefine` in the following way:

```
prior.hier=false;
prior=priordefine(data,mix,prior);
```

The default choice for such a prior is an invariant prior with following hyperparameters (Frühwirth-Schnatter, 2006, Subsection 6.2.2):

$$b_0 = \overline{y}, \qquad B_0 = s_y^2, \qquad c_0 = 2.5, \qquad C_0 = \phi(c_0 - 1)s_y^2, \tag{6.4}$$

where $\overline{y}$ is the sample mean, $s_y^2$ is the sample variance, and $\phi = 0.5$.

Alternatively, it is possible to force the package to use a standard conjugate prior with the hyperparameters selected as in Frühwirth-Schnatter (2006, Subsection 6.2.2):

$$b_0 = \overline{y}, \qquad N_0 = 1, \qquad c_0 = 2.5, \qquad C_0 = \phi(c_0 - 1)s_y^2, \tag{6.5}$$

where $\overline{y}$, $s_y^2$ and $\phi$ are the same as in (6.4). This prior is obtained by calling the function `priordefine` in the following way:

```
prior.hier=false;
prior.type='concon';
prior=priordefine(data,mix,prior);
```

To define a hierarchical conjugate prior which combines the conjugate prior (6.5) with a hierarchical prior where $g_0 = 0.5$ and $G_0 = g_0 C_0^{-1}$ call the function `priordefine` in the following way:

```
prior.hier=true;
prior.type='concon';
prior=priordefine(data,mix,prior);
```

### 6.2.4 Choosing the Prior Distribution for Multivariate Mixtures of Normals

The choice of prior distributions for multivariate mixtures of normal distributions is discussed in Frühwirth-Schnatter (2006, Subsection 6.3.2). Both conditionally conjugate as well as independence priors are implemented in the toolbox.

#### The Structure of the Prior

The prior has to be a structure array as explained in Section 4.2.1, including the following fields:

- `type` specifies the prior type. This is one of the following strings:
  - `'concon'` refers to the conditionally conjugate prior

$$\boldsymbol{\mu}_k | \boldsymbol{\Sigma}_k \sim \mathcal{N}_r\left(\mathbf{b}_0, \boldsymbol{\Sigma}_k / N_{0,k}\right), \qquad \boldsymbol{\Sigma}_k^{-1} \sim \mathcal{W}_r\left(c_{0,k}, \mathbf{C}_{0,k}\right). \tag{6.6}$$

  - `'indep'` refers to the independence prior

$$\boldsymbol{\mu}_k \sim \mathcal{N}_r\left(\mathbf{b}_{0,k}, \mathbf{B}_{0,k}\right), \qquad \boldsymbol{\Sigma}_k^{-1} \sim \mathcal{W}_r\left(c_{0,k}, \mathbf{C}_{0,k}\right). \tag{6.7}$$

- The field `par` contains the hyperparameters of these priors and is a structure array with the fields `mu` and `sigma`, respectively.

- For the conditionally conjugate prior the field `mu` is a structure array with the fields `b` and `N0`, being, respectively, a `r x K` and a `1 x K` numerical array specifying the parameters $\mathbf{b}_{0,k}$ and $N_{0,k}$ of the normal prior (6.6).
- For the independence prior the field `mu` is a structure array with fields `b` and `Binv`, specifying the parameters $\mathbf{b}_{0,k}$ and $\mathbf{B}_{0,k}^{-1}$ of the normal prior (6.7). `b` is a `r x K` numerical array and `Binv` is a `r x r x K` numerical array.
- For both priors, the field `sigma` is a structure array with the fields `c` and `C`, specifying the parameters $c_{0,k}$ and $\mathbf{C}_{0,k}$ of the Wishart prior $\mathbf{\Sigma}_k^{-1} \sim \mathcal{W}_r\left(c_{0,k}, \mathbf{C}_{0,k}\right)$. `c` is a `1 x K` numerical array, `C` is a `r x r x K` numerical array.
- For both priors the field `logdetC`, containing the log of the determinant $\log|\mathbf{C}_{0,k}|$ should be added, because this speeds up the computation of the prior. This field is a `1 x K` numerical array.

Note that the arrays defining these fields contain `K` entries, even if the prior is invariant.

If the hyperparameter $\mathbf{C}_0$ of an invariant prior is a random parameter with a prior of its own, $\mathbf{C}_0 \sim \mathcal{W}_r\left(g_0, \mathbf{G}_0\right)$, then additional fields have to be added to the prior specification:

- The field `hier` which is a logical variable taking the value `true`.
- The fields `par.sigma.g` and `par.sigma.G` containing the parameters $g_0$ and $\mathbf{G}_0$ of the Wishart prior.

### The Default Choice

The toolbox allows an automatic selection of slightly data dependent, rather noninformative priors by calling the function `priordefine`, see Subsection 4.2.1. This default choice is an invariant hierarchical independence prior (Stephens, 1997) with the hyperparameters selected as in Frühwirth-Schnatter (2006, Subsection 6.3.2):

$$\mathbf{b}_0 = \begin{pmatrix} m_1 \\ \vdots \\ m_r \end{pmatrix}, \quad \mathbf{B}_0 = \mathrm{Diag}\big(\, R_1^2 \cdots \, R_r^2 \,\big), \quad \mathbf{G}_0 = \frac{100 g_0}{c_0}\mathbf{B}_0^{-1},$$
$$g_0 = 0.5 + (r-1)/2, \qquad c_0 = \nu_c + (r-1)/2,$$

where $m_l$ and $R_l$ are the midpoint and the length of the observation interval of the $l$th component of $\mathbf{y}_i$.

It is possible to overrule this choice and to select an invariant conjugate default prior (Bensmail et al., 1997) with the hyperparameters selected as in Frühwirth-Schnatter (2006, Subsection 6.3.2):

$$\mathbf{b}_0 = \overline{\mathbf{y}}, \quad N_0 = 1, \quad c_0 = \nu_c + (r-1)/2, \quad \mathbf{C}_0 = \phi(\nu_c - 1)\mathbf{S}_{\mathbf{y}}, \qquad (6.8)$$

where $r = \dim \mathbf{y}$, $\overline{\mathbf{y}}$ is the sample mean vector, $\mathbf{S_y}$ is the sample covariance, $\nu_c = 2.5$, and $\phi = 0.5$. This prior is obtained by calling the function `priordefine` in the following way:

```
prior.hier=false;
prior.type='concon';
prior=priordefine(data,mix,prior);
```

Alternatively, it is possible to select a default standard independence prior which is an invariant prior with following hyperparameters (Frühwirth-Schnatter, 2006, Subsection 6.2.3):

$$\mathbf{b}_0 = \overline{\mathbf{y}}, \qquad \mathbf{B}_0 = \mathbf{S_y}, \qquad c_0 = \nu_c + (r-1)/2, \qquad \mathbf{C}_0 = \phi(\nu_c - 1)\mathbf{S_y},$$

where $\overline{\mathbf{y}}$, $\mathbf{S_y}$, $\nu_c$, and $\phi$ are the same as in (6.8). This prior is obtained by calling the function `priordefine` in the following way:

```
prior.hier=false;
prior=priordefine(data,mix,prior);
```

Finally, it is possible to use a default hierarchical conjugate prior which combines the conjugate prior (6.8) with a hierarchical prior where

$$g_0 = 0.5 + (r-1)/2, \qquad \mathbf{G}_0 = g_0/\mathbf{C}_0^{-1}.$$

This prior is obtained by calling the function `priordefine` in the following way:

```
prior.hier=true;
prior.type='concon';
prior=priordefine(data,mix,prior);
```

### 6.2.5 Bayesian Inference for a Single Normal Distribution

Assume that a single normal distribution, either univariate or multivariate, should be fitted to i.i.d. data and a prior is selected as in Subsection 6.2.3 or 6.2.4.

Under an independence prior or under a hierarchical prior, no closed form posterior is available for the whole parameter vector, and either the mean or the variance have to be fixed, to obtain a closed form conditional distribution. To sample from the posterior distribution under an independence prior, one has to implement a two-step Gibbs sampler. For a hierarchical independence prior, a three-step Gibbs sampler has to be implemented. To run Gibbs sampling under any arbitrary prior call the function `mixturemcmc` explained in Subsection 4.3.

Under a conjugate, non-hierarchical prior, a closed form posterior is available which takes the form a Normal-Gamma or a Normal-Wishart family. To compute the parameters of the posterior distribution, the function `posterior` may be called:

```
post.par=posterior(data,model,prior.par);
```

`post.par` is a structural array containing the same fields as `prior.par`. But even in this case, it is preferable to use the function `mixturemcmc` to sample from the posterior distribution as this allows the application of a lot tools developed for Bayesian inference based on posterior draws, see Section 4.5.

### 6.2.6 Bayesian Parameter Estimation When the Allocations are Known

For data where the allocations are known, the structure array `data` has to include the field `S`, storing the allocations. For a complete-data Bayesian estimation as discussed in Subsection 6.2.1 and Subsection 6.3.1 of Frühwirth-Schnatter (2006), you need first to define a prior on the parameters, stored in a structure array `prior` as described in Subsection 6.2.3 or 6.2.4. Like for a single normal distribution, whether a closed form of the conditional posterior exists or not depends on the nature of the prior.

Under an independence prior or under a hierarchical prior, no closed form posterior is available for the whole parameter vector, even if the allocations are known, and either the mean or the variance have to be fixed, to obtain a closed form conditional distribution.

Under a conjugate, non-hierarchical prior, a closed form posterior is available when the allocations are known which takes the form a Normal-Gamma or a Normal-Wishart family. To compute the parameters of the posterior distribution, the function `posterior` may be called:

```
post.par=posterior(data,model,prior.par);
```

`post.par` is a structural array containing the same fields as `prior.par`. But even in this case, it is preferable to use the function `mixturemcmc` to sample from the posterior distribution.

To run complete-data Gibbs sampling under any arbitrary prior call the function `mixturemcmc` explained in Subsection 4.3, however, the allocations have to fixed beforehand:

```
mix.indicfix=true;
mcmcout=mixturemcmc(data,mix,prior,mcmc);
```

No random permutation will be performed in this case, even if `mcmc.ranperm` is set `true`.

### 6.2.7 Bayesian Parameter Estimation When the Allocations are Unknown

This section concerns parameter estimation when the allocations are unknown. Bayesian estimation of finite mixtures of normal distributions using data augmentation and MCMC is discussed in Frühwirth-Schnatter (2006) for univariate mixtures in Subsection 6.2.4 and for multivariate mixtures in Subsection 6.3.3.

For univariate mixtures, MCMC sampling is performed as described in Frühwirth-Schnatter (2006), *Algorithm 6.1*. Sampling the component parameters $\boldsymbol{\theta}_k = (\mu_k, \sigma_k^2)$ involves the following steps:

(a) Sample $\sigma_k^2$ in each group $k$ from a $\mathcal{G}^{-1}\left(c_k(\mathbf{S}), C_k(\mathbf{S})\right)$-distribution.
(b) Sample $\mu_k$ in each group $k$ from an $\mathcal{N}\left(b_k(\mathbf{S}), B_k(\mathbf{S})\right)$-distribution.

For multivariate mixtures, MCMC sampling is performed as described in Frühwirth-Schnatter (2006), *Algorithm 6.2*. Sampling the component parameters $\boldsymbol{\theta}_k = (\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ involves the following steps:

(a) Sample $\boldsymbol{\Sigma}_k^{-1}$ in each group $k$ from a $\mathcal{W}_r\left(c_k(\mathbf{S}), \mathbf{C}_k(\mathbf{S})\right)$-distribution.
(b) Sample $\boldsymbol{\mu}_k$ in each group $k$ from an $\mathcal{N}_r\left(\mathbf{b}_k(\mathbf{S}), \mathbf{B}_k(\mathbf{S})\right)$-distribution.

To run data augmentation and MCMC for data stored in `data` for the mixture model `mix` under prior `prior`, call the function `mixturemcmc` explained in Subsection 4.3. The structure of the MCMC output is explained in full detail in Subsection 6.2.11.

One may call the function `mcmcstart` explained in Subsection 4.3 before starting MCMC to make use of default starting values. The remainder of this subsection explains, how these starting values are selected.

### Default Starting Values

Unless stated otherwise (see filed `mcmc.startpar` in Subsection 4.3.1), MCMC estimation starts with sampling the parameters and the indicators stored in `data.S` are selected as starting value for the classification $\mathbf{S}^{(0)}$. In higher dimensions, it is usually easier to find a sensible starting value for the classification and the cluster means than for the cluster means and the variance-covariance matrices.

Under the independence prior, sampling of $\boldsymbol{\theta}_k$ involves two blocks, where the first block samples the component (co)variances conditional on the component means. Thus starting values for the means are needed which need to be stored in `mix.par.mu` before calling the function `mixturemcmc`.

The function `mcmcstart` adds starting values for the field `S` in the structure array describing the data and, if necessary, the field `mu` in the structure array describing the mixture model. This function is based on $k$-means clustering of the data stored in `data.y` using the MATLAB function `kmeans`. The resulting cluster means are chosen as starting values for `mu`.

Alternatively, one may start MCMC with sampling the indicators, in which case `mix` has to be a fully specified mixture before calling `mixturemcmc`. To determine starting values for a fully specified mixture, again the function `mcmcstart` may be called, however with the additional input argument `mcmc`, where `mcmc.startpar=true`. For univariate mixtures with $K > 1$, starting values for $\mu_k$ are sampled from $\mathcal{N}\left(\overline{y}, s_y^2\right)$, while all starting values for $\sigma_k^2$ are equal to $s_y^2$. For $K = 1$, $\mu_1 = \overline{y}$. For multivariate mixtures with $K > 1$, starting values for $\boldsymbol{\mu}_k$ are sampled from $\mathcal{N}_r\left(\overline{\mathbf{y}}, \mathbf{S}_\mathbf{y}\right)$, while all starting values

for $\boldsymbol{\Sigma}_k$ are equal to $\mathbf{S}_\mathbf{y}$. For $K = 1$, $\boldsymbol{\mu}_1 = \overline{\mathbf{y}}$. The starting value for the weight distribution is uniform, i.e. $\eta_k = 1/K$.

Finally, under a hierarchical prior the prior parameter `prior.par.G` has to be set to an appropriate starting value, for instance, the mean of the prior put on the random hyperparameter. Automatic prior definition using the function `priordefine`, see Subsection 6.2.3 or 6.2.4, automatically chooses such a starting value for MCMC estimation.

### 6.2.8 Plotting MCMC

The function `mcmcplot`, introduced in Subsection 4.5.1, could be used to plot and monitor the MCMC output.

#### Sampling Representations of the Mixture Posterior Density

To produce sampling representations of the posterior draws as explained in Subsection 4.5.1 function `mcmcplot` calls the function `mcmcsamrep`. For univariate mixtures $\mu_k^{(m)}$ is plotted against $\sigma_k^{(2,m)}$. If the permutation sampler has been used (`mcmcout.ranperm` is `true`), then, additionally, the MCMC draws $\mu_k^{(m)}$ are plotted against $\mu_{k'}^{(m)}$ and the MCMC draws $\sigma_k^{(2,m)}$ are plotted against $\sigma_{k'}^{(2,m)}$.

For the MCMC output of a multivariate mixture of normals, the following point process representation are produced:

- A point process representation of each univariate marginal mixture density of $Y_j$, for each $j = 1, \ldots, r$, by plotting $\mu_{k,j}^{(m)}$ against $\boldsymbol{\Sigma}_{k,jj}^{(m)}$. This plot provides information about similarity of the mixture components in each marginal density.
- A point process representation of each bivariate marginal mixture density of $(Y_j, Y_{j'})$ for all possible combinations $(j, j')$ of elements of $\mathbf{Y}$, by plotting the means $\mu_{k,j}^{(m)}$ of element $j$ against the means $\mu_{k,j'}^{(m)}$ of element $j'$.
- To obtain point process representations for the covariance matrices of each multivariate normal component density, $\log|\boldsymbol{\Sigma}_k^{-1}|^{(m)}$ is plotted against $\mathrm{tr}\,(\boldsymbol{\Sigma}_k)^{(m)}$. Furthermore the largest eigenvalue of $\boldsymbol{\Sigma}_k^{(m)}$ is plotted against the smallest one.

### 6.2.9 Estimating the Component Parameters and the Weight Distribution

To perform parameter estimation, call the function `est=mcmcestimate(mcmcout)` introduced in Subsection 4.5.2. For each estimation method, the estimators of the weight distribution $\eta_1, \ldots, \eta_K$ are stored in the field `weight` while the estimators of the parameters are stored in the field `par`, which has the same structure as for the estimated mixture, e.g.

- `est.pm.par.mu` – (approximate) posterior mode estimator of the groups means $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$ (or $\mu_1, \ldots, \mu_K$),
- `est.pm.par.sigma` – (approximate) posterior mode estimator of the groups variances $\boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\Sigma}_K$ (or $\sigma_1^2, \ldots, \sigma_K^2$),
- `est.ident.par.mu` – ergodic average estimator of the groups means $\boldsymbol{\mu}_1, \ldots, \boldsymbol{\mu}_K$ (or $\mu_1, \ldots, \mu_K$) after identification,
- `est.ident.par.sigma` – ergodic average estimator of the groups variances $\boldsymbol{\Sigma}_1, \ldots, \boldsymbol{\Sigma}_K$ (or $\sigma_1^2, \ldots, \sigma_K^2$) after identification.

### 6.2.10 Model Selection Problems for Mixtures of Normals

To compute the log of the marginal likelihood as in Frühwirth-Schnatter (2006, Subsection 6.4.2), call the function `mcmcbf`, see Section 5.3 for more details.

#### Example

The demo `demo_mix_multivariate_normal_Kunknown.m` fits finite mixtures of multivariate normal distributions with the number of components increasing from $K = 1$ to $K = 4$ to simulated data that where generated by a mixture of three bivariate normal distributions. Table 6.2 reports the various estimators of the log of the marginal likelihood. The standard errors are stored in the field `se`. All estimators select the true number of components. We observe increased numerical instability for overfitting models like $K = 4$. In particular importance sampling and reciprocal importance sampling are instable, while bridge sampling is much more precise.

**Table 6.2.** Running the demo `demo_mix_multivariate_normal_Kunknown.m`; log of various estimates of the marginal likelihood $p(\mathbf{y}|\mathcal{M}_K)$ under the default prior; BS ... bridge sampling, IS ... importance sampling, RI ... reciprocal importance sampling; standard errors in parenthesis

| | $K$ | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| $\hat{p}_{BS}(\mathbf{y}\|\mathcal{M}_K)$ | -2946.63 | -2410.33(0.002) | **-2075.17**(0.002) | -2082.45(0.037) |
| $\hat{p}_{IS}(\mathbf{y}\|\mathcal{M}_K)$ | -2946.63 | -2410.33(0.003) | **-2075.17**(0.003) | -2081.78(0.364) |
| $\hat{p}_{RI}(\mathbf{y}\|\mathcal{M}_K)$ | -2946.63 | -2410.33(0.003) | **-2075.18**(0.004) | -2085.55(0.303) |

### 6.2.11 The Structure of the MCMC Output

The MCMC output is a structure array having the fields defined in Subsection 4.3.2. In this subsection only those fields are described in more details which are specific to normal mixture models.

### The MCMC Output for univariate mixtures of normals

- `par` is a structure array with the fields `mu` and `sigma` containing the MCMC draws for the component parameters:
  - `mu` is a `M x K` numerical array storing the posterior draws $\mu_k^{(m)}$.
  - `sigma` is a `M x K` numerical array storing the posterior draws $\sigma_k^{(2,m)}$.
- `hyper` is added under a hierarchical prior. This is a `M x 1` numerical array containing the MCMC draws $C_0^{(m)}$ for the random hyperparameter.
- `post.par` is a structure array with the fields `mu` and `sigma`. The field `post.par.mu` is a structure array with following fields:
  - `b` is a `M x K` numerical array storing for each group $k$ the mean $b_k(\mathbf{S})$ of the normal posterior $\mathcal{N}(b_k(\mathbf{S}), B_k(\mathbf{S}))$ used for sampling $\mu_k^{(m)}$.
  - `B` is a `M x K` numerical array storing for each group $k$ the variance $B_k(\mathbf{S})$ of this distribution.

  The field `post.par.sigma` is a structure array with following fields:
  - `c` is a `M x K` numerical array storing for each group $k$ the shape parameter $c_k(\mathbf{S})$ of the inverted Gamma posterior $\mathcal{G}^{-1}(c_k(\mathbf{S}), C_k(\mathbf{S}))$ used for sampling $\sigma_k^{(2,m)}$.
  - `C` is a `M x K` numerical array storing for each group $k$ the scale parameter $C_k(\mathbf{S})$ of the same distribution. For a conjugate prior, this is the parameter of the marginal distribution, where the unknown mean is integrated out.

### The MCMC Output for multivariate mixtures of normals

For multivariate mixtures the covariances $\mathbf{\Sigma}_k$ are simulated by drawing $\mathbf{\Sigma}_k^{-1}$ from a Wishart distribution and computing $\mathbf{\Sigma}_k$ as the numerical inverse of $\mathbf{\Sigma}_k^{-1}$. If the posterior draws of $\mathbf{\Sigma}_k$ are needed for further numerical evaluation, like evaluating $p(\mathbf{\Sigma}_k|\mathbf{S}, \mathbf{y})$ or computing eigenvalues, it is often safer to work with $\mathbf{\Sigma}_k^{-1}$. For this reason, both the original draws $(\mathbf{\Sigma}_k^{-1})^{(m)}$ as well as the numerically inverted draws $\mathbf{\Sigma}_k^{(m)}$ are stored. You may prevent storing of $(\mathbf{\Sigma}_k^{-1})^{(m)}$ by calling `mixturemcmc(data,mix,prior,mcmc)` with the option `mcmc.storeinv` being `false`.

To save storage place, for MCMC draws of symmetric matrices of size $r$ only the upper triangular matrix is stored as a vector of length $s = r(r+1)/2$. Various utility functions are available to convert a symmetric matrix into such a vector and to recover a symmetric matrix from such a vector:

- `mat=qinmatr(col)` converts the `s x 1` column vector `col`, where `s=r(r+1)/2` into the `r x r` array `mat`.
- Similarly, `mat=qinmatrmult(col)` converts a sequence of $K$ column vector, stored as `s x K` array `col` into a sequence of `r x r` arrays, stored in the `r x r x K` array `mat`.
- `col= qincol(mat)` converts the `r x r` array `mat` into the `s x 1` column vector `col`, where `s=r(r+1)/2`.

- Similarly, `col= qincolmult(mat)` converts a sequence of $K$ symmetric matrices of dimension $r$, stored in the `r x r x K` array `mat`, into a sequence of $K$ column vector of length $s = r(r+1)/2$, stored as `s x K` array `col`.

In `qinmatr` and `qinmatrmult` a warning will be produced, if no integer $r$ exists such that $s = r(r+1)/2$.

The MCMC output is stored in the structure array `mcmcout` having the fields defined in Subsection 4.3.2. In the following only those fields are described in more details which are specific to multivariate normal mixture models:

- `par` is a structure array with the fields `mu`, `sigma`, `sigmainv`, and `logdet` containing the MCMC draws for the component parameters:
  - `mu` is a `M x r x K` numerical array storing the posterior draws $\boldsymbol{\mu}_k^{(m)}$.
  - `sigma` is a `M x s x K` numerical array storing the posterior draws $\boldsymbol{\Sigma}_k^{(m)}$ as column vectors. To reconstruct as single draw $\boldsymbol{\Sigma}_k^{(m)}$, call the function `qinmatr`, to reconstruct all covariance matrices $\boldsymbol{\Sigma}_1^{(m)}, \ldots, \boldsymbol{\Sigma}_K^{(m)}$ for the $m$th draw call the function `qinmatrmult`:

    ```
    mix.par.sigma(:,:,k)=qinmatr(mcmcout.par.sigma(m,:,k)')
    mix.par.sigma=qinmatr(mcmcout.par.sigma(m,:,:))
    ```

  - `sigmainv` is a `M x s x K` numerical array storing the posterior draws $(\boldsymbol{\Sigma}_k^{-1})^{(m)}$ as column vectors, unless `mcmc.storeinv` is `false`. To reconstruct a single draw $(\boldsymbol{\Sigma}_k^{-1})^{(m)}$, call the function `qinmatr`, to reconstruct all covariance matrices $(\boldsymbol{\Sigma}_1^{-1})^{(m)}, \ldots, (\boldsymbol{\Sigma}_K^{-1})^{(m)}$ for the $m$th draw call the function `qinmatrmult`:

    ```
    mix.par.sigmainv(:,:,k)=qinmatr(mcmcout.par.sigmainv(m,:,k)')
    mix.par.sigmainv=qinmatr(mcmcout.par.sigmainv(m,:,:))
    ```

  - `logdet` is a `M x K` numerical array, storing $\log|(\boldsymbol{\Sigma}_k^{-1})^{(m)}|$.
- `hyper` is added under a hierarchical prior. This is a `M x s` numerical array containing the MCMC draws of the (symmetric) random hyperparameter matrices $\mathbf{C}_0^{(m)}$, stored as column vectors.
- `post.par` is a structure array with the fields `mu` and `sigma`. The field `post.par.mu` is a structure array with following fields:
  - `b` is a `M x r x K` numerical array storing for each group $k$ the mean $\mathbf{b}_k(\mathbf{S})$ of the normal posterior $\mathcal{N}_r\left(\mathbf{b}_k(\mathbf{S}), \mathbf{B}_k(\mathbf{S})\right)$ used for sampling $\boldsymbol{\mu}_k^{(m)}$.
  - `B` is a `M x r x r x K` numerical array storing for each group $k$ the variance $B_k(\mathbf{S})$ of the same distribution.

  The field `post.par.sigma` is a structure array with following fields:
  - `c` is a `M x K` numerical array storing for each group $k$ the shape parameter $c_k(\mathbf{S})$ of the Wishart posterior $\mathcal{W}_r\left(c_k(\mathbf{S}), \mathbf{C}_k(\mathbf{S})\right)$ used for sampling $(\boldsymbol{\Sigma}_k^{-1})^{(m)}$.

- `C` is a `M x s x K` numerical array storing for each group $k$ the (symmetric) scale matrix $\mathbf{C}_k(\mathbf{S})$ of the same distribution as a vector of size `s=r(r+1)/2`. For a conjugate prior, this is the parameter of the marginal distribution, where the unknown mean is integrated out.
- `logdetC` is a `M x K` numerical array storing for each group $k$ the log of the determinant $\log|\mathbf{C}_k(\mathbf{S})|$.

## 6.3 Finite Mixtures of Student-$t$ Distributions

Finite mixtures of Student-$t$ distributions are defined and discussed in detail in Frühwirth-Schnatter (2006, Section 7.3). For univariate observations, it is assumed that the observations $\mathbf{y} = (y_1, \ldots, y_N)$ are independent realizations of a random variable $Y$ arising from the following mixture of univariate Student-$t$ distributions:

$$Y \sim \eta_1 t_{\nu_1}\left(\mu_1, \sigma_1^2\right) + \cdots + \eta_K t_{\nu_K}\left(\mu_K, \sigma_K^2\right). \tag{6.9}$$

For multivariate observations, it is assumed that the observations $\mathbf{y} = (\mathbf{y}_1, \ldots, \mathbf{y}_N)$ are independent realizations of a random variable $\mathbf{Y}$ arising from the following mixture of multivariate Student-$t$ distributions:

$$\mathbf{Y} \sim \eta_1 t_{\nu_1}\left(\boldsymbol{\mu}_1, \boldsymbol{\Sigma}_1\right) + \cdots + \eta_K t_{\nu_K}\left(\boldsymbol{\mu}_K, \boldsymbol{\Sigma}_K\right). \tag{6.10}$$

### 6.3.1 Defining Mixtures of Student-$t$ Distributions

To define a finite mixture of Student-$t$ distributions within this toolbox, create a structure array as explained in Subsection 2.2.1. The field `dist` is equal to `'Student'` for univariate mixtures and equal to `'Stumult'` for multivariate mixtures. The field `par` is a structure array with three fields:

- `mu` contains the component specific location parameters $\mu_k$ or $\boldsymbol{\mu}_k$;
- `sigma` contains the component specific scale parameters $\sigma_k^2$ or $\boldsymbol{\Sigma}_k$;
- `df` contains the component specific degrees of freedoms $\nu_1, \ldots, \nu_K$.

For a univariate mixture, `par.mu`, `par.sigma` and `par.df` are `1 x K` numeric arrays. For a multivariate mixture of dimension `r`, `par.mu` is a `r x K` numeric array, `par.sigma` is `r x r x K` numeric array and `par.df` is a `1 x K` numeric array.

For a mixture of $K$ multivariate Student-$t$ distributions it is often convenient to work with $\boldsymbol{\Sigma}_k^{-1}$ and $\log|\boldsymbol{\Sigma}_k^{-1}|$, e.g. if the mixture is applied for classification. These quantities are stored as optional fields of `par`, where

- `sigmainv` contains $\boldsymbol{\Sigma}_1^{-1}, \ldots, \boldsymbol{\Sigma}_K^{-1}$, characterized by a `r x r x K` numeric array;
- `logdet` contains $\log|\boldsymbol{\Sigma}_1^{-1}|, \ldots, \log|\boldsymbol{\Sigma}_K^{-1}|$, characterized by a `1 x K` numeric array.

Whereas it is mathematically correct to consider a univariate Student-$t$ mixture as that special case of a multivariate mixture where $r = 1$, a combination of `'Stumult'` with `mix.r=1` should be avoided when using this package, because this almost surely leads to troubles with array sizes when the intrinsic function `squeeze` is applied by one the routines.

### Simulated Data

Data are simulated from a finite mixture of Student-$t$ distributions by calling the function `simulate`, see Subsection 3.3. To simulate the data, the following hierarchical representation of a finite mixture of Student-$t$ distributions is used, where the distributions

$$S_i \sim \text{MulNom}\left(\eta_1, \ldots, \eta_K\right),$$
$$\omega_i | S_i = k \sim \mathcal{G}\left(\nu_k/2, \nu_k/2\right),$$

are combined with

$$Y_i | S_i = k, \omega_i \sim \mathcal{N}\left(\mu_k, \sigma_k^2/\omega_i\right), \tag{6.11}$$

for univariate mixtures and with

$$\mathbf{Y}_i | S_i = k, \omega_i \sim \mathcal{N}_r\left(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k/\omega_i\right), \tag{6.12}$$

for multivariate mixtures. The "true" scaling factors $\omega_i$ used in the simulation are added for simulated data as additional field to the structural array defining the data:

- The field `omega` is a `1 x N` array containing the "true" scaling factors $\omega_1, \ldots, \omega_N$ used in the simulation of the data.

## 6.3.2 Getting Started Quickly

Several demos are available, that demonstrate how to fit mixtures of Student-$t$ distributions to simulated data:

- `demo_mix_student.m`: fits a mixture of two univariate Student-$t$ distributions to simulated data (takes about 6 CPU minutes), see also Subsection 6.3.6.
- `demo_mix_student_Kunknown.m`: fits mixtures of univariate Student-$t$ distributions with increasing number of components to simulated data (takes about 11 CPU minutes).
- `demo_mix_multivariate_student.m`: fits a finite mixture of three bivariate Student-$t$ distributions to bivariate simulated data (takes about 7 CPU minutes).
- `demo_mix_multivariate_student_Kunknown.m`: fits mixtures of bivariate Student-$t$ distributions with increasing number of components to bivariate simulated data (takes about 25 CPU minutes), see also Subsection 6.2.10.

### 6.3.3 Choosing the Prior Distribution

An attractive feature of a Bayesian approach is estimating the degrees of freedom $\nu_k$ along with all other unknown quantities. Bayesian estimation is based on assuming prior independence between $\nu_k$ and the remaining component specific parameters. For univariate mixtures, the following prior is used:

$$p(\mu_k, \sigma_k^2, \nu_k) = p(\mu_k, \sigma_k^2)p(\nu_k), \tag{6.13}$$

where $p(\mu_k, \sigma_k^2)$ is the same prior as in discussed in Subsection 6.2.3. For multivariate mixtures, the following prior is used:

$$p(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \nu_k) = p(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)p(\nu_k), \tag{6.14}$$

where $p(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ is the same prior as in discussed in Subsection 6.2.4.

The prior on $\nu_k$ has to be selected carefully in order to avoid improper posteriors, see e.g. Geweke (1993) and Fonseca et al. (2008). In this package following translated prior is used:

$$p(\nu_k) \propto \frac{(\nu_k - c)^{a_0 - 1}}{(\nu_k - c + d)^{a_0 + b_0}} I_{\{c,\infty\}}(\nu_k), \tag{6.15}$$

where $a_0$, $b_0$, $c$ and $d$ are hyperparameters selected by the user. Choosing $c > 0$ shifts the prior away from 0, as it is advisable to avoid values for $\nu_k$ that are close to 0, see Fernández and Steel (1999).

#### The Structure of the Prior

The prior has the same structure as for mixtures of normal distributions, see Subsection 6.2.3 and 6.2.4, respectively. The field `par` has an additional field to define the prior of $\nu_k$:

- The field `df` defines the prior for $\nu_1, \ldots, \nu_K$. It is a structural array with following fields:
  - The field `type` defines the type of prior used for $p(\nu_k)$ and is equal to `'inhier'` for prior (6.15).
  - The field `trans` defines the hyperparameter $c$;
  - the field `a0` defines the hyperparameter $a_0$;
  - the field `b0` defines the hyperparameter $b_0$;
  - the field `d` defines the hyperparameter $d$.

#### The Default Choice

The toolbox allows the automatic selection of a slightly data dependent, rather noninformative prior by calling the function `priordefine`. This default choice combines the prior $p(\nu_k)$ defined in (6.15), where $a_0 = 2$, $b_0 = 2$, $c = 1$ and $d = 9$, with the same default choice for $(\mu_k, \sigma_k^2)$ or $(\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$ as for mixtures

of normals, see Subsection 6.2.3 and 6.2.4, respectively. For this prior choice, the prior median of $\nu_k$ is equal to 10, while the prior mean is equal to 20. This particular choice guarantees that the posterior distribution is proper and that the marginal posterior distribution of $\nu_k$ has a finite expectation. Thus the average of the MCMC draws may be used to estimate $\nu_k$.

### 6.3.4 Bayesian Parameter Estimation When the Allocations are Unknown

This section concerns parameter estimation when the allocations are unknown. Bayesian estimation of finite mixtures of Student-$t$ distributions using data augmentation and MCMC is discussed in Frühwirth-Schnatter (2006, Subsection 7.3.1). Bayesian estimation of mixtures of Student-$t$ distributions is based on the representation of the Student-$t$ distribution as an infinite scale mixture of normal distributions, see (6.11) and (6.12), respectively. Thus a mixture of Student-$t$ distributions may be regarded as a mixture of normal distributions, where all group members have the same expectation $\boldsymbol{\mu}_k$, however, within each group there exists variance heterogeneity, captured by the scaling factor $\omega_i$, with smaller values of $\omega_i$ causing larger variances.

The sampling scheme presented in Subsection 6.2.7 for normal mixtures has to be extended, see Frühwirth-Schnatter (2006), *Algorithm 7.1*. MCMC sampling is performed as described in Frühwirth-Schnatter (2006), *Algorithm 6.1* and an additional step has to be added to sample $\nu_1, \ldots, \nu_K$.

Depending on the degree of data augmentation in the conditional density $p(\nu_1, \ldots, \nu_K|\cdot)$, different Metropolis-Hastings steps to sample $\nu_k$ result. The fastest algorithm is sampling $\nu_k$ conditional on knowing the scaling parameters $\boldsymbol{\omega} = (\omega_1, \ldots, \omega_N)$, by drawing from the full conditional posterior $p(\nu_k|\boldsymbol{\omega}, \mathbf{S}, \mathbf{y})$ by means of a Metropolis–Hastings algorithm as Lin et al. (2007) did. However, this works only, if the degree of freedom is small in all components. Tremendous inefficiency factors may be observed, if some of the $\nu_k$s were larger than about 10.

Sampling $\nu_k$ from $p(\nu_k|\mu_k, \sigma_k^2, \mathbf{S}, \mathbf{y})$ or $p(\nu_k|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \mathbf{S}, \mathbf{y})$ where $\boldsymbol{\omega}$ is integrated out increases efficiency considerably. Additional efficiency is gained by sampling $\nu_k$ without conditioning on $\mathbf{S}$ and $\boldsymbol{\omega}$ from $p(\nu_k|\boldsymbol{\theta}_{-k}, \boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k, \boldsymbol{\eta}, \mathbf{y})$ where $\boldsymbol{\theta}_{-k}$ denotes all component specific parameters expect $\boldsymbol{\theta}_k$. However, this sampler is the most time consuming one because it involves the computation of the mixture likelihood $p(\mathbf{y}|\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K, \boldsymbol{\eta})$.

Nevertheless, this is the default choice in the package. The corresponding Metropolis-Hastings algorithm is based on the uniform log random walk proposal

$$\log(\nu_k^{new} - 1) \sim \mathcal{U}\left[\log(\nu_k - 1) - c_{\nu_k}, \log(\nu_k - 1) + c_{\nu_k}\right], \tag{6.16}$$

with fixed width parameter $c_{\nu_k}$. The width parameter $c_{\nu_k}$ has to be selected by the user prior to running MCMC, see below.

To run data augmentation and MCMC for data stored in `data` for the Student-$t$ mixture model defined in `mix` under prior `prior`, call the function `mixturemcmc` explained in Subsection 4.3. The structure of the MCMC output is explained in Subsection 6.3.7.

One may call the function `mcmcstart` explained in Subsection 4.3 before starting MCMC to make use of default starting values. The function `mcmcstart` *does not choose* the width parameter $c_{\nu_k}$ of the random walk Metropolis-Hastings algorithm used for sampling $\nu_k$. The width parameters $c_{\nu_1}, \ldots, c_{\nu_1}$ have to be stored prior to calling `mixturemcmc` by the user in an additional field of the array `mcmc` controlling MCMC:

- The field `mh.tune.df` is numerical array of size `1 x K` defining the width parameters $c_{\nu_1}, \ldots, c_{\nu_K}$ of the uniform log random walk proposals for $\nu_1, \ldots, \nu_K$ defined in (6.16).

### Default Starting Values

The remainder of this subsection explains how these starting values are selected.

Unless stated otherwise (see filed `mcmc.startpar` in Subsection 4.3.1), MCMC estimation starts with sampling the parameters and the indicators stored in `data.S` are selected as starting value for the classification $\mathbf{S}^{(0)}$. Additionally, for Student-$t$ mixtures starting values are needed for the scaling factors $\omega_1, \ldots, \omega_N$ which need to be stored in `data.omega`. The function `mcmcstart` selects the starting values $\omega_i = 1, i = 1, \ldots, N$. These values may be changed after calling `mcmcstart` simply by reassigning other starting values to the array `data.omega`.

Under a conjugate prior, sampling of the component specific parameter $\boldsymbol{\theta}_k$ involves two blocks, where the first block samples the component means and the component (co)variances conditional on the component degrees of freedom. Thus starting values of the degrees of freedom are needed which need to be stored in `mix.par.df` before calling the function `mixturemcmc`. Under the independence prior, sampling of $\boldsymbol{\theta}_k$ involves three blocks, where the first block samples the component (co)variances conditional on the component means and the component degrees of freedom. Thus starting values for the means and the degrees of freedom are needed which need to be stored in `mix.par.mu` and `mix.par.df` before calling the function `mixturemcmc`.

The function `mcmcstart` adds starting values for the fields `S` and `omega` in the structure array describing the data, for the degrees of freedom and, if necessary, for the field `mu` in the structure array describing the mixture model. This function is based on $k$-means clustering of the data stored in `data.y` using the MATLAB function `kmeans`. The resulting cluster means are chosen as starting values for `mu`. The starting value for $\nu_k$ is equal to 10. It may be easily changed after calling `mcmcmstart` just by setting the array `mix.par.df` to different values.

Alternatively, one may start MCMC with sampling the indicators, in which case `mix` has to be a fully specified mixture before calling `mixturemcmc`, including the field `par` and, for $K > 1$, the field `weight`. In this case, `data.S` may be unspecified. To determine starting values for a fully specified mixture, again the function `mcmcstart` may be called, however with the additional input argument `mcmc`, where `mcmc.startpar=true`. Starting values for $\mu_k$ and $\sigma_k^2$ or $\boldsymbol{\mu}_k$ and $\boldsymbol{\Sigma}_k$ are determined exactly as for a normal mixture, see Subsection 6.2.7, while $\nu_k$ is equal to 10.

### 6.3.5 Plotting MCMC

The function `mcmcplot`, introduced in Subsection 4.5.1, could be used to plot and monitor the MCMC output.

### 6.3.6 Model Selection Problems for Mixtures of Student-$t$ distributions

To compute the log of the marginal likelihood as in Frühwirth-Schnatter (2006, Subsection 7.3.2), call the function `mcmcbf`, see Section 5.3 for more details. The importance density for $\nu_k$ is constructed through a kernel density estimator applied to the MCMC draws of $\nu_k$. The stability of the various estimators is much smaller than for mixtures of normal distributions.

#### Examples

The demo `demo_mix_student.m` fits a finite mixtures of two univariate Student-$t$ distributions to data that were simulated from such a mixture. Table 6.3 reports the various estimators of the log of the marginal likelihood of this model for two independent MCMC runs (5000 draws after a burn-in of 2000 draws). For each MCMC chain, marginal likelihood estimation was performed twice independently. It is evident that bridge sampling is very stable over independent runs. Importance sampling and reciprocal importance sampling are rather unstable.

**Table 6.3.** Running the demo `demo_mix_student.m`; log of various estimates of the marginal likelihood $p(\mathbf{y}|\mathcal{M}_2)$ for a mixture of two Student-$t$ distributions under the default prior; BS ... bridge sampling, IS ... importance sampling, RI ... reciprocal importance sampling; standard errors in parenthesis

| | first MCMC run | | second MCMC run | |
|---|---|---|---|---|
| | estimator 1 | estimator 2 | estimator 1 | estimator 2 |
| $\hat{p}_{BS}(\mathbf{y}\|\mathcal{M}_K)$ | -3548.55(0.02) | -3548.52(0.02) | -3548.53(0.02) | -3548.37(0.02) |
| $\hat{p}_{IS}(\mathbf{y}\|\mathcal{M}_K)$ | -3538.47(0.41) | -3535.94(0.89) | -3539.24(0.47) | -3537.05(0.65) |
| $\hat{p}_{RI}(\mathbf{y}\|\mathcal{M}_K)$ | -3595.03(0.99) | -3588.77(0.99) | -3586.84(0.95) | -3589.51(0.99) |

The demo `demo_mix_multivariate_student_Kunknown.m` fits finite mixtures of multivariate Student-$t$ distributions with the number of components increasing from $K = 1$ to $K = 4$ to simulated data that where generated by a mixture of three bivariate Student-$t$ distributions. Table 6.4 reports the various estimators of the log of the marginal likelihood. The standard errors are stored in the field `se`. The bridge sampling and importance sampling estimator select the true number of components, while reciprocal importance sampling selects the wrong model. We observe increased numerical instability for overfitting models like $K = 4$. Importance sampling and reciprocal importance sampling are instable, while bridge sampling is much more precise.

**Table 6.4.** Running the demo `demo_mix_multivariate_student_Kunknown.m`; log of various estimates of the marginal likelihood $p(\mathbf{y}|\mathcal{M}_K)$ under the default prior; BS ... bridge sampling, IS ... importance sampling, RI ... reciprocal importance sampling; standard errors in parenthesis

| | $K$ | | | |
|---|---|---|---|---|
| | 1 | 2 | 3 | 4 |
| $\hat{p}_{BS}(\mathbf{y}\|\mathcal{M}_K)$ | -6078.81(0.01) | -4923.18(0.01) | **-4412.82**(0.02) | -4424.91(0.04) |
| $\hat{p}_{IS}(\mathbf{y}\|\mathcal{M}_K)$ | -6077.82(0.07) | -4922.01(0.08) | **-4401.11**(0.85) | -4418.70(0.45) |
| $\hat{p}_{RI}(\mathbf{y}\|\mathcal{M}_K)$ | -6079.58(0.05) | **-4924.08**(0.06) | -4432.25(0.84) | -4449.24(0.99) |

### 6.3.7 The Structure of the MCMC Output

The MCMC output is a structure array having the same fields as a mixture of normal distribution, see Subsection 6.2.11. The following fields are added for mixtures of Student-$t$ distributions:

- The field `mh` provides details about the Metropolis-Hastings algorithm. It is a structural array with two fields:
  - The field `tune.df` is a `1 x K` array containing the tuning parameters for the log random walk Metropolis Hastings algorithm, see Subsection 6.3.4.
  - The field `acc.df` is a `1 x K` array containing the acceptance rates for the log random walk Metropolis Hastings algorithm for the degrees of freedom parameters $\nu_1, \ldots, \nu_K$.
- The field `par` has the fields described in Subsection 6.2.11 and, additionally, a field containing the MCMC draws for $\nu_1, \ldots, \nu_K$:
  - The field `df` is a `M x K` numerical array storing the posterior draws $\nu_k^{(m)}$.

## 6.4 Finite Mixtures of Exponential Distributions

It is often assumed that nonnegative observations are realizations of a random variable $Y$ arising from a finite mixture of exponential distributions:

$$Y \sim \eta_1 \mathcal{E}(\lambda_1) + \cdots + \eta_K \mathcal{E}(\lambda_K), \tag{6.17}$$

where $\mathcal{E}(\lambda_k)$ is an exponential distribution with mean $1/\lambda_k$. Mixtures of exponential distributions are discussed in Frühwirth-Schnatter (2006, Section 9.1) and in Wagner (2007).

### 6.4.1 Defining Mixture of Exponential Distributions

To define a finite mixture of exponential distributions within this MATLAB package, create a structure array as explained in Section 2.2.1, where the field `par` is a `1 x K` numeric array containing the component parameters $\lambda_1, \ldots, \lambda_K$.

### 6.4.2 Getting Started Quickly

A demo is available that demonstrates how to fit mixtures of exponential distributions to simulated data, see Subsection 3.3 for details on how to simulate data from a finite mixture distribution:

- `demo_mix_exponential.m`: fits a mixture of two exponential distributions to simulated data (takes less than 1 CPU minute).

### 6.4.3 Choosing the Prior for Bayesian Estimation

The choice of prior distributions for mixtures of exponential distributions is discussed in detail in Wagner (2007). The standard choice is the conditionally conjugate prior $\lambda_k \sim \mathcal{G}(a_{0,k}, b_{0,k})$.

#### The Structure of the Prior

The prior has to be a structure array as explained in Section 4.2.1. For mixtures of exponential distributions the field `par` is a structure array with two fields, storing the prior parameters:

- `a` is a `1 x K` numerical arrays storing $a_{0,1}, \ldots, a_{0,K}$.
- `b` is a `1 x K` numerical arrays storing $b_{0,1}, \ldots, b_{0,K}$.

#### The Default Choice

The toolbox allows an automatic selection of a slightly data dependent, rather noninformative prior by calling the function `priordefine`, see Subsection 4.2.1. The parameter $a_0 = 0.1$ is chosen to be a small value as in Wagner (2007). The parameter $b_0$ is chosen in such a way that the prior mean is matched to the mean of the data, i.e. $b_0 = a_0 \overline{y}$.

### 6.4.4 Parameter Estimation When the Allocations are Unknown

Bayesian estimation of finite mixtures of exponential distributions using data augmentation and MCMC is discussed in Frühwirth-Schnatter (2006, Subsection 9.1.2). Sampling the component parameter $\lambda_k$ involves the following step:

(a) For each $k = 1, \ldots, K$, sample $\lambda_k$ from a $\mathcal{G}\left(a_k(\mathbf{S}), b_k(\mathbf{S})\right)$-distribution.

To run data augmentation and MCMC call the function `mixturemcmc` explained in Subsection 4.3. The structure of the MCMC output is explained in Subsection 6.4.7. One may call the function `mcmcstart` explained in Subsection 4.3 before starting MCMC to make use of default starting values.

#### Default Starting Values

Unless stated otherwise (see filed `mcmc.startpar` in Subsection 4.3.1), MCMC estimation starts with sampling the parameters and the indicators stored in `data.S` are selected as starting value for the classification $\mathbf{S}^{(0)}$. The function `mcmcstart` adds starting values for the field `S` in the structure array describing the data. This function is based on $k$-means clustering of the data stored in `data.y` using the MATLAB function `kmeans`. Clustering is applied to the transformed data $z_i = \sqrt{y_i}$.

Alternatively, one may start MCMC with sampling the indicators, in which case `mix` has to be a fully specified mixture before calling `mixturemcmc`. To determine starting values for a fully specified mixture, again the function `mcmcstart` may be called, however with the additional input argument `mcmc`, where `mcmc.startpar=true`. For $K > 1$, starting values for $\lambda_k$ are defined as $\lambda_k = \exp(z_k)/\overline{y}$, where $z_k \sim \mathcal{N}\left(0, 0.5^2\right)$, while $\lambda_1 = 1/\overline{y}$ for $K = 1$. The starting value for the weight distribution is uniform, i.e. $\eta_k = 1/K$.

### 6.4.5 Plotting MCMC

The function `mcmcplot`, introduced in Subsection 4.3.2, could be used to plot and monitor the MCMC output.

### 6.4.6 Model Selection Problems for Mixtures of Exponentials

To compute the log of the marginal likelihood as in Wagner (2007) call the function `mcmcbf`, see Section 5.3 for more details.

### 6.4.7 The MCMC Output for Mixtures of Exponentials

The MCMC output is a structure array having the fields defined in Subsection 4.3.2. In this subsection only those fields are described in more details which are specific to mixtures of exponential distributions:

- `par` is a `M x K` numerical array storing the posterior draws $\lambda_k^{(m)}$.
- `post.par` is a structure array with following fields:
  - `a` is a `M x K` numerical array storing for each group $k$ the shape parameter $a_k(\mathbf{S})$ of the posterior $\mathcal{G}\left(a_k(\mathbf{S}), b_k(\mathbf{S})\right)$ used for sampling $\lambda_k^{(m)}$.
  - `b` is a `M x K` numerical array storing for each group $k$ the scale parameter $b_k(\mathbf{S})$ of the same distribution.

# 7

# Finite Mixture Models for Discrete-Valued Data

This chapter deals with finite mixture modelling of discrete-valued or categorical data.

## 7.1 Data Handling

The data are defined as a structure array as described in Subsection 3.1.1. The field `type='discrete'` should be added, because this automatically allows additional options for these data. If exposures should be included in the analysis or if the data result from repeated measurements, then an additional field has to be added to the structure array defining the data:

- `Ti` is usually a numerical array of size `1 x data.N`, but could also be a single integer number.

### Data Sets Available in the Package

For illustration, several data sets are stored under particular names and could be loaded into a structure array using the function `dataget`:

- Eye Tracking Data: `data=dataget('eye');`

### Plotting the Data

Use the function `dataplot(data)`, described in Subsection 3.2.1, to plot the data.

## 7.2 Finite Mixtures of Poisson Distributions

A popular model for describing the distribution of count data is the Poisson mixture model, where it is assumed that $y_1, \ldots, y_N$ are independent realization of a random variable $Y$ arising from a mixture of Poisson distributions:

$$Y \sim \eta_1 \mathcal{P}(\mu_1) + \cdots + \eta_K \mathcal{P}(\mu_K), \tag{7.1}$$

with $\mathcal{P}(\mu_k)$ being a Poisson distribution with mean $\mu_k$ (Frühwirth-Schnatter, 2006, Section 9.2). If exposures $e_1, \ldots, e_N$ are available, then the mixture model reads:

$$Y_i \sim \eta_1 \mathcal{P}(e_i \mu_1) + \cdots + \eta_K \mathcal{P}(e_i \mu_K). \tag{7.2}$$

### 7.2.1 Defining Mixtures of Poisson Distributions

To define a finite mixture of Poisson distributions create a structure array as explained in Section 2.2.1, where the field `par` is a `1 x K` numeric array containing the component parameters $\mu_1, \ldots, \mu_K$.

#### Including Exposures

Exposures $e_1, \ldots, e_N$ are stored in the field `Ti` of the structure array defining the data, see Subsection 7.1. For all functions in the package, where both the mixture model and the data appear as input argument, it is assumed implicitly, that the repetition parameter or the exposures of the data and the model are the same.

Only if a function is called, where only the structure array defining the mixture model appears as input argument, then a field `Ti` has to be added explicitly to the array defining the model before calling this function. Plotting the mixture density as discussed in Subsection 2.2.2 and computing moments of the finite mixture distributions as discussed in Subsection 2.2.4 is possible only, if the number of exposures is the same for all observations.

#### Simulated Data

When data are simulated from a Poisson distribution using the function `simulate` introduced in Subsection 3.3, then it is usually assumed that no exposures are available. It is, however, possible to simulate data for a given sequence of exposures, that is stored in the field `Ti` of the structure array defining the data, say `mydata`. To this aim, call the function `simulate` with three input arguments, for instance:

```
mydata.Ti=[103 26 31 40 62 71 93 80 35 76];
data=simulate(mymodel,10,mydata);
```

### 7.2.2 Getting Started Quickly

A demo is available, that demonstrate how to fit mixtures of Poisson distributions to real data:

- `start_eye.m`: fits finite mixtures of Poisson distributions with $K = 1$ to $K = 7$ to the Eye Tracking Data (takes about 11 CPU minutes), see also Subsection 1.2.3.

### 7.2.3 Choosing the Prior for Bayesian Estimation

The choice of prior distributions for Poisson mixtures is discussed in Frühwirth-Schnatter (2006, Subsection 9.2.1).

#### The Structure of the Prior

The prior is based on the conditionally conjugate priors $\mu_k \sim \mathcal{G}(a_{0,k}, b_{0,k})$ and is defined as a structure array as explained in Section 4.2.1. The field `par` is a structure array with two fields:

- `a` is a `1 x K` numerical arrays storing $a_{0,1}, \ldots, a_{0,K}$.
- `b` is a `1 x K` numerical arrays storing $b_{0,1}, \ldots, b_{0,K}$.

If hyperparameters $b_0$ of an invariant prior, where $\mu_k \sim \mathcal{G}(a_0, b_0)$ for all $k = 1, \ldots, K$, is a random parameter with prior $b_0 \sim \mathcal{G}(g_0, G_0)$, then the following fields have to be added to the structure array defining the prior:

- The field `hier` taking the value `true`.
- The field `g` containing $g_0$
- The field `G` containing $G_0$.

For the hierarchical prior, the values stored in field `b` act as a starting values prior to MCMC estimation and are updated during MCMC sampling.

#### The Default Choice

The toolbox allows an automatic selection of a slightly data dependent, rather noninformative hierarchical prior by calling the function `priordefine`, see Subsection 4.2.1.

The tuning of the automatic prior is based on moment matching (Frühwirth-Schnatter, 2006, Subsection 9.2.1). $a_0$ is derived from matching second order moments:

$$a_0 = \frac{\overline{y}^2}{s_y^2 - \overline{y}}.$$

The parameter $b_0$ is chosen in such a way that the prior mean $\mathrm{E}(Y|\boldsymbol{\vartheta}) = a_0/b_0$ is matched to the mean of the data:

$$b_0 = \frac{a_0}{\overline{y}}.$$

The larger the overdispersion in the data, the smaller $a_0$ will be chosen. If $s_y^2 - \overline{y} \leq 0$, then $a_0 = 10$. If overdispersion is small, then $a_0$ is large and $\mu_k$ is strongly shrunken toward $\overline{y}$. To avoid the inclusion of too much prior information, the default prior is a hierarchical prior, where $b_0 \sim \mathcal{G}(g_0, G_0)$ with $g_0 = 0.5$. Matching $\mathrm{E}(b_0) = g_0/G_0$ to $a_0/\overline{y}$ yields:

$$G_0 = \frac{g_0 \overline{y}}{a_0}.$$

One may overrule certain default choices. To define a standard conjugate prior, where $a_0$ and a fixed hyper parameter $b_0$ are selected automatically, call the function `priordefine` in the following way:

```
prior.hier=false;
prior=priordefine(data,mix,prior);
```

### 7.2.4 Parameter Estimation When the Allocations are Unknown

Bayesian estimation of finite mixtures of Poisson distributions using data augmentation and MCMC is discussed in Frühwirth-Schnatter (2006, Subsection 3.5.2). Sampling the component parameter $\mu_k$ involves the following step:

(a) For each $k = 1, \ldots, K$, sample $\mu_k$ from a $\mathcal{G}\left(a_k(\mathbf{S}), b_k(\mathbf{S})\right)$-distribution.

To run data augmentation and MCMC call the function `mixturemcmc` explained in Subsection 4.3. The structure of the MCMC output is explained in detail in Subsection 7.2.8. One may call the function `mcmcstart` explained in Subsection 4.3 before starting MCMC to make use of default starting values.

#### Default Starting Values

Under a hierarchical prior, the prior parameter `prior.par.G` has to be set to an appropriate starting value, for instance, the mean of the prior put on the random hyperparameter $b_0$. Automatic prior definition using the function `priordefine` automatically chooses such a starting value for MCMC estimation.

Unless stated otherwise (see filed `mcmc.startpar` in Subsection 4.3.1), MCMC estimation starts with sampling the parameters and the indicators stored in `data.S` are selected as starting value for the classification $\mathbf{S}^{(0)}$. The function `mcmcstart` adds starting values for the field `S` in the structure array describing the data. This function is based on $k$-means clustering of the data stored in `data.y` using the MATLAB function `kmeans`. Clustering is applied to the transformed data $z_i = \sqrt{y_i}$.

For the sake of comparison, one may start MCMC with sampling the indicators, in which case `mix` has to be a fully specified mixture before calling `mixturemcmc`. To determine starting values for a fully specified mixture, again the function `mcmcstart` may be called, however with the additional input argument `mcmc`, where `mcmc.startpar=true`. For $K > 1$, starting values for $\mu_k$ are defined as $\mu_k = \max(0.1, \overline{y} \exp(z_k))$, where $z_k \sim \mathcal{N}\left(0, 0.5^2\right)$, while $\mu_1 = \max(0.1, \overline{y})$ for $K = 1$. The starting value for the weight distribution is uniform, i.e. $\eta_k = 1/K$.

### 7.2.5 Unknown number of components

To compute the log of the marginal likelihood call the function `mcmcbf`, see Section 5.3 for more details.

### 7.2.6 Bayesian Fitting of a Single Poisson Distribution

Assume that a single Poisson distribution should be fitted to i.i.d. data and a prior is selected as in Subsection 7.2.3. Under a hierarchical prior, no closed form posterior is available and either the mean or the hyperparameter has to be fixed to obtain a closed form conditional distribution. To sample from the posterior distribution, one has to implement a two-step Gibbs sampler by calling the function `mixturemcmc` explained in Subsection 4.3.

Under a conjugate, non-hierarchical prior, a closed form posterior is available which takes the form a Gamma-distribution. To compute the parameters of the posterior distribution, the function `posterior` may be called:

```
post.par=posterior(data,model,prior.par);
```

`post.par` is a structural array containing the same fields as `prior.par`. But even in this case, it is preferable to use the function `mixturemcmc` to sample from the posterior distribution as this allows the application of a lot tools developed for Bayesian inference based on posterior draws, see Section 4.5.

### 7.2.7 Bayesian Parameter Estimation When the Allocations are Known

For data where the allocations are known, the structure array `data` has to include the field `S`, storing the allocations. A prior is selected as in Subsection 7.2.3. Under a hierarchical prior, no closed form posterior is available, even if the allocations are known. To sample from the posterior distribution, one has to implement a two-step Gibbs sampler by calling the function `mixturemcmc` explained in Subsection 4.3, however, the allocations have to fixed beforehand:

```
mix.indicfix=true;
mcmcout=mixturemcmc(data,mix,prior,mcmc);
```

No random permutation will be performed in this case, even if `mcmc.ranperm` is set `true`.

Under the conjugate prior, the complete-data posterior is the product of $K$ gamma distributions. To compute the moments of this posterior distribution, simply call the function `posterior`:

```
post=posterior(data,mix,prior);
```

The structure array `post` will have the same fields as the prior. But even in this case, it is preferable to use the function `mixturemcmc` in combination with `mix.indicfix=true` to sample from the posterior distribution.

### 7.2.8 The structure of the MCMC Output

The MCMC output is stored in the structure array `mcmcout` having the fields defined in Subsection 4.3.2. In this subsection only those fields are described in more details which are specific to Poisson mixture models:

- `par` is a `M x K` numerical array storing the posterior draws $\mu_k^{(m)}$.
- `hyper` is added under a hierarchical prior. This is a `M x 1` numerical array containing the MCMC draws $b_0^{(m)}$ for the random hyperparameter $b_0$.
- `post.par` is a structure array with following fields:
  - `a` is a `M x K` numerical array storing for each group $k$ the shape parameter $a_k(\mathbf{S})$ of the posterior $\mathcal{G}\left(a_k(\mathbf{S}), b_k(\mathbf{S})\right)$ used for sampling $\mu_k^{(m)}$.
  - `b` is a `M x K` numerical array storing for each group $k$ the scale parameter $b_k(\mathbf{S})$ of the same distribution.

## 7.3 Finite Mixtures of Binomial Distributions

For binomial mixtures the component densities arise from $\text{BiNom}\left(T, \pi\right)$-distributions, where the repetition parameter $T$ is assumed to be known, whereas the component-specific probabilities $\pi$ are unknown and heterogeneous:

$$Y \sim \eta_1 \text{BiNom}\left(T, \pi_1\right) + \cdots + \eta_K \text{BiNom}\left(T, \pi_K\right).$$

The density of this mixture is given by

$$p(y|\boldsymbol{\vartheta}) = \sum_{k=1}^{K} \eta_k \binom{T}{y} \pi_k^y (1-\pi_k)^{T-y}, \tag{7.3}$$

with $\boldsymbol{\vartheta} = (\pi_1, \ldots, \pi_K, \eta_1, \ldots, \eta_K)$. Binomial mixtures are not necessarily identifiable, see Frühwirth-Schnatter (2006, Section 9.3.1). A necessary and sufficient condition is $T \geq 2K - 1$.

Finite mixtures of binomial distributions may be extended to the case where the repetition parameter $T_i$ varies between the realizations $y_1, \ldots, y_N$:

$$p(y_i|\boldsymbol{\vartheta}) = \sum_{k=1}^{K} \eta_k \binom{T_i}{y_i} \pi_k^{y_i} (1-\pi_k)^{T_i - y_i}.$$

### 7.3.1 Defining Mixtures of Binomial Distributions

To define a mixture of binomial distributions create a structure array as explained in Section 2.2.1 where the field `par` is a `1 x K` numeric array containing the component parameters $\pi_1, \ldots, \pi_K$.

The repetition parameter is usually defined through the data, by adding the field `Ti` to the structure array defining the data, see Subsection 7.1. If $T$ is fixed over all observations, then `Ti` is a single integer containing $T$. If the repetition parameter varies across the observations, then `Ti` is a array of the same size as the field `y`, i.e. `1 x N`, containing $T_1, \ldots, T_N$. Whenever both the mixture model and the data appear as input argument of a function in this package, it is assumed implicitly that the repetition parameter of the data and the model is the same.

Whenever calling a function, where only the structure array defining the mixture model appears as input argument the field `Ti` has to be added explicitly to the array defining the model before calling this function. Note that plotting the mixture density as discussed in Subsection 2.2.2 and computing moments of the finite mixture distribution as discussed in Subsection 2.2.4 is possible only, if the repetition parameter is the same for all observations.

#### Simulated Data

When data are simulated from a mixture of binomial distribution using the function `simulate` introduced in Subsection 3.3, then the repetition parameter has to be stored in the field `Ti` of the structure array defining the data, say `mydata`, before calling `simulate`. Furthermore, the function `simulate` has to be called with three input arguments:

```
mydata.Ti=[103 26 31 40 62 71 93 80 35 76];
data=simulate(mymodel,10,mydata);
```

If the function `simulate` is called with only two input arguments, then it is assumed that $T_i = 1$ for all observations, i.e. that the data are binary.

### 7.3.2 Getting Started Quickly

A demo is available that demonstrate how to fit a mixture of binomial distributions to simulated data:

- `demo_mix_binomial.m` fits a mixture of two binomial distributions to simulated data (takes less than 1 CPU minute).

### 7.3.3 Choosing the Prior for Bayesian Estimation

The choice of prior distributions for mixtures of binomial distributions is discussed in Frühwirth-Schnatter (2006, Subsection 9.3.2).

### The Structure of the Prior

The prior is based on the conditionally conjugate prior $\pi_k \sim \mathcal{B}(a_{0,k}, b_{0,k})$ and is defined as a structure array as explained in Section 4.2.1 with the field `par` being equal to a structure array with two fields:

- `a` is a `1 x K` numerical arrays storing $a_{0,1}, \ldots, a_{0,K}$;
- `b` is a `1 x K` numerical arrays storing $b_{0,1}, \ldots, b_{0,K}$.

### The Default Prior

The toolbox allows an automatic selection of a prior by calling the function `priordefine`, see Subsection 4.2.1. For all components this default choice is a uniform prior, i.e. $a_{0,k} = b_{0,k} = 1$.

## 7.3.4 Parameter Estimation When the Allocations are Unknown

Bayesian estimation of finite mixtures of binomial distributions using data augmentation and MCMC is discussed in Frühwirth-Schnatter (2006, Subsection 9.3.1). Sampling the component parameter $\theta_k = \pi_k$ involves the following step:

(a) For each $k = 1, \ldots, K$, sample $\pi_k$ from a $\mathcal{B}(a_k(\mathbf{S}), b_k(\mathbf{S}))$-distribution.

To run data augmentation and MCMC call the function `mixturemcmc` explained in Subsection 4.3. The structure of the MCMC output is explained in Subsection 7.3.6. One may call the function `mcmcstart` explained in Subsection 4.3 before starting MCMC to make use of default starting values.

### Default Starting Values

Unless stated otherwise (see filed `mcmc.startpar` in Subsection 4.3.1), MCMC estimation starts with sampling the parameters and the indicators stored in `data.S` are selected as starting value for the classification $\mathbf{S}^{(0)}$. The function `mcmcstart` determines starting values for $\mathbf{S}^{(0)}$ in the following way. If the range of the data is large enough, i.e. if $\max(y_i) - \min(y_i) \geq 2K$, then $k$-means clustering using the MATLAB function `kmeans` is applied to the transformed data $z_i = \sqrt{y_i}$. If the range of the data is smaller than $2K$, then a random classification is applied.

Alternatively, one may start MCMC with sampling the indicators, in which case `mix` has to be a fully specified mixture before calling `mixturemcmc`. To determine starting values for a fully specified mixture, again the function `mcmcstart` may be called, however with the additional input argument `mcmc`, where `mcmc.startpar=true`. For $K > 1$, starting values for $\pi_k$ are defined as $\pi_k = \min(\max(0.1, \overline{h} \exp(z_k)), 0.9)$, where $h_i = y_i / T_i$ and $z_k \sim \mathcal{N}\left(0, 0.2^2\right)$, while $\pi_1 = \min(\max(0.1, \overline{h}), 0.9)$ for $K = 1$. The starting value for the weight distribution is uniform, i.e. $\eta_k = 1/K$.

### 7.3.5 Unknown number of components

To compute the log of the marginal likelihood call the function `mcmcbf`, see Section 5.3 for more details.

### 7.3.6 The structure of the MCMC Output

The MCMC output is stored in the structure array `mcmcout` having the fields defined in Subsection 4.3.2. The following fields are specific to mixtures of binomial distributions:

- `par` is a `M x K` numerical array storing the posterior draws $\pi_k^{(m)}$.
- `post.par` is a structural array with following fields:
  - `a` is a `M x K` numerical array storing for each group $k$ the shape parameter $a_k(\mathbf{S})$ of the posterior $\mathcal{B}\left(a_k(\mathbf{S}), b_k(\mathbf{S})\right)$ used for sampling $\pi_k^{(m)}$.
  - `b` is a `M x K` numerical array storing for each group $k$ the scale parameter $b_k(\mathbf{S})$ of the same distribution.

# 8

# Finite Mixtures of Regression Models

## 8.1 Data Handling

Data to which a regression model is fitted are defined as a structure array in the following way:

- The field `y` contains the dependent observations. This is a `1 x N` numeric array, where `N` is the number of observations.
- The field `X` contains the independent variables, where each row corresponds to a certain covariate. This is a `s x N` numeric array, where `s` is the number of independent variables.

Note that all variables are stored by row. If `bycolumn` is true (see below), then `y` is a `N x 1` numeric array and `X` is a `N x s` numeric array.

Optional fields are the following:

- The field `name` is the name of the data set, stored as character.
- The field `N` is the number of observations.
- The field `bycolumn` is a logical variable which is true, if the variables are stored by column. If this field is missing, then it is assumed that the data are stored by row.

### Data Sets Available in the Package

For illustration, several data sets are stored under particular names and could be loaded into a structure array using the function `dataget`:

- Star Cluster Data: `data=dataget('starclust')`
- Fabric Fault Data: `data=dataget('fabricfault')`

### Visualization

Use the function `dataplot(data)` to plot the data.

## 8.2 Finite Mixture of Multiple Regression Models

Finite mixtures of multiple regression models and their statistical inference are discussed in detail in Frühwirth-Schnatter (2006, Section 8.2 and 8.3). In this section we will discuss regression modelling based on normal errors, for more general distributions see Section 8.4.

A finite mixture regression model assumes that a set of $K$ regression models characterized by the parameters $(\boldsymbol{\beta}_1, \sigma^2_{\varepsilon,1}), \ldots, (\boldsymbol{\beta}_K, \sigma^2_{\varepsilon,K})$ exists, and that for each observation pair $(Y_i, \mathbf{x}_i)$ a hidden random indicator $S_i$ chooses one among these models to generate $Y_i$:

$$Y_i = \mathbf{x}_i \boldsymbol{\beta}_{S_i} + \varepsilon_i, \qquad \varepsilon_i \sim \mathcal{N}\left(0, \sigma^2_{\varepsilon,S_i}\right). \tag{8.1}$$

$\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_K$ as well as $\sigma^2_{\varepsilon,1}, \ldots, \sigma^2_{\varepsilon,K}$ are unknown parameters that need to be estimated from the data.

### 8.2.1 Defining a Finite Mixture Regression Model

#### Specifying the Model Structure

To define a standard finite mixture regression model, create a structure array, named for instance `mixreg`, containing the following fields:

- The field `dist` defined the parametric distribution family of the regression model. In this section we will discuss only data from a normal distribution, thus `dist='Normal'`, for more general distributions see Section 8.4.
- The field `d` defines the dimension of the regression parameter.
- The field `K` contains the number $K$ of regimes.

If the field `d` is missing, this model definition reduces to a standard finite mixture of univariate normal distributions. If the field `K` is missing, this model definition reduces a standard regression model.

#### Assigning Parameters

For a fully specified finite mixture regression model, values have to be assigned to all model parameters:

- The field `par` contains the coefficients of the regression model in each regime. This is a structure array with following fields:
  - `beta` is a `d x K` numeric array containing the regression parameters;
  - `sigma` is a `1 x K` numeric array containing the error variances.
- The field `weight` contains the weight distribution $\boldsymbol{\eta} = (\eta_1, \ldots, \eta_K)$, characterized by a `1 x K` numeric array. This field is missing, if $K = 1$.

### 8.2.2 Getting Started Quickly

A demo is available, that demonstrate how to fit mixtures of regression models to simulated data, see Subsection 8.5.1 for details on how to simulate data from a finite mixture of regression models:

- `demo_mixreg`: fits a standard regression model and finite mixtures of regression models with $K = 2$ and $K = 3$ to data that are simulated from a finite mixture of two regression models and selects and evaluates the model with the largest marginal likelihood (takes about 5 CPU minutes).

### 8.2.3 Choosing the Prior Distribution

The choice of prior distributions for mixtures of regression models is discussed in Subsection 8.3.3 of Frühwirth-Schnatter (2006). In the current version of the toolbox only the independence prior, where

$$\boldsymbol{\beta}_k \sim \mathcal{N}_d\left(\mathbf{b}_{0,k}, \mathbf{B}_{0,k}\right), \qquad \sigma^2_{\varepsilon,k} \sim \mathcal{G}^{-1}\left(c_{0,k}, C_{0,k}\right), \tag{8.2}$$

is implemented for mixtures of regression models, no conjugate prior is available. It is possible (and recommended) to use a hierarchical prior, where the hyperparameter $C_{0,k} \equiv C_0$ is a random variable with a prior of its own, $C_0 \sim \mathcal{G}\left(g_0, G_0\right)$.

#### The Default Prior Choice

The toolbox allows an automatic selection of a slightly data dependent, rather noninformative proper prior by calling the function `priordefine`:

```
prior=priordefine(data,mixreg);
```

The selected prior is a hierarchical independence prior, where

$$\begin{aligned} &\mathbf{b}_{0,k} = \mathbf{b}_0, \qquad \mathbf{B}_{0,k} = 10\mathbf{I}_d, \\ &c_{0,k} = \nu_c, \qquad g_0 = 0.5, \qquad \mathbf{G}_0 = g_0\phi(\nu_c - 1)s^2_y, \end{aligned} \tag{8.3}$$

where $b_{0,j} = \overline{y}$ with $\overline{y}$ being the sample mean of the dependent variable if $\beta_{k,j}$ is a switching intercept, and $b_{0,j} = 0$, otherwise. $s^2_y$ is the sample variance of the dependent variable, $\nu_c = 2.5$, and $\phi = 0.5$.

#### The Structure of the Prior

The prior is a structure array as explained in Section 4.2.1. For mixtures of regression models the field `par` is a structure array with two fields:

- The field `beta` is a structure array with the fields `b` and `Binv`, being a `d x K` and a `d x d x K` numerical array specifying the parameters $\mathbf{b}_{0,k}$ and $\mathbf{B}^{-1}_{0,k}$ of prior (8.2).

- The field `sigma` is a structure array with the fields `c` and `C`, being `1 x K` numerical arrays specifying the parameters $c_{0,k}$ and $C_{0,k}$ of prior (8.2).

Note that the arrays defining these fields have to contain $K$ entries, even if the prior is invariant. For a hierarchical prior, the following additional fields have to be added to the prior specification:

- The field `hier` which is a logical variable taking the value `true`.
- The fields `par.sigma.g` and `par.sigma.G` containing the parameters $g_0$ and $G_0$ of the Gamma prior.

### 8.2.4 Bayesian Inference When the Allocations Are Unknown

Typically, the allocations are unknown and MCMC estimation of both the parameters and the allocations is carried out using data augmentation and Gibbs sampling using Algorithm 8.1 described in Frühwirth-Schnatter (2006, Subsection 8.3.4). To run data augmentation and MCMC for data stored in `data` for the mixture regression model `mixreg` under prior `prior`, call the function `mixturemcmc` explained in Subsection 4.3:

```
mcmcout=mixturemcmc(data,mixreg,prior,mcmc);
```

The structure of the MCMC output is explained in full detail in Subsection 8.2.5. One may call the function `mcmcstart` explained in Subsection 4.3 before starting MCMC to make use of default starting values. The remainder of this subsection explains, how this starting values are selected.

#### Default Starting Values

Unless stated otherwise (see filed `mcmc.startpar` in Subsection 4.3.1), MCMC estimation starts with sampling the parameters and the indicators stored in `data.S` are selected as starting value for the classification $\mathbf{S}^{(0)}$. Under the independence prior, sampling of $\boldsymbol{\beta}_k$ and $\sigma^2_{\varepsilon,k}$ involves two blocks, where the first block samples the regression parameters conditional on the error variances. Thus starting values for the error variances are needed which have to be stored in `mixreg.par.sigma` before calling the function `mixturemcmc`.

The function `mcmcstart` determines these starting values in the following way. Using the MATLAB function `kmeans`, $k$-means clustering is applied to the multivariate data where the data in `data.y` are merged with the regressors in `data.X`. All starting values for $\sigma^2_{\varepsilon,k}$ are equal to $s^2_y$.

For the sake of comparison, one may start MCMC with sampling the indicators, in which case `mix` has to be a fully specified mixture before calling `mixturemcmc`. To determine starting values for a fully specified mixture, again the function `mcmcstart` may be called, however with the additional input argument `mcmc`, where `mcmc.startpar=true`. The starting values for all regression coefficients except the coefficient $\beta_{k,j}$ corresponding to the intercept are

set equal to 0. For $K > 1$, starting values for $\beta_{k,j}$ are sampled from $\mathcal{N}\left(\overline{y}, s_y^2\right)$, while all starting values for $\sigma_{\varepsilon,k}^2$ are equal to $s_y^2$. For $K = 1$, $\beta_{1,j} = \overline{y}$. The starting value for the weight distribution is uniform, i.e. $\eta_k = 1/K$.

### 8.2.5 The Structure of the MCMC Output

The MCMC output is stored in the structure array `mcmcout` having the fields defined in Subsection 4.3.2:

- `par` is a structure array with the fields `beta` and `sigma` containing the MCMC draws for the regression parameters:
  - `beta` is a `M x d x K` numerical array storing the posterior draws $\boldsymbol{\beta}_k^{(m)}$.
  - `sigma` is a `M x K` numerical array storing the posterior draws $(\sigma_{\varepsilon,k}^2)^{(m)}$.
- `hyper` is added under a hierarchical prior. This is a `M x 1` numerical array containing the MCMC draws $C_0^{(m)}$ for the random hyperparameter.
- `post.par` is a structure array with the fields `beta` and `sigma`. The field `post.par.beta` is a structure array with following fields:
  - `b` is a `M x d x K` numerical array storing for each group $k$ the mean $\mathbf{b}_k(\mathbf{S})$ of the normal posterior $\mathcal{N}\left(\mathbf{b}_k(\mathbf{S}), \mathbf{B}_k(\mathbf{S})\right)$ used for sampling $\boldsymbol{\beta}_k^{(m)}$.
  - `B` is a `M x d x d x K` numerical array storing for each group $k$ the variance $\mathbf{B}_k(\mathbf{S})$ of the same posterior.

  The field `post.par.sigma` is a structure array with following fields:
  - `c` is a `M x K` numerical array storing for each group $k$ the shape parameter $c_k(\mathbf{S})$ of the inverted Gamma posterior $\mathcal{G}^{-1}\left(c_k(\mathbf{S}), C_k(\mathbf{S})\right)$ used for sampling $(\sigma_{\varepsilon,k}^2)^{(m)}$.
  - `C` is a `M x K` numerical array storing for each group $k$ the scale parameter $C_k(\mathbf{S})$ of the same distribution.

## 8.3 Mixed-Effects Finite Mixtures of Regression Models

A mixed-effects model allows us to combine regression coefficients that are fixed across all realizations $(Y_i, \mathbf{x}_i)$ with regression coefficients that are allowed to change:

$$Y_i = \mathbf{x}_i^f \boldsymbol{\alpha} + \mathbf{x}_i^r \boldsymbol{\beta}_{S_i} + \varepsilon_i, \qquad \varepsilon_i \sim \mathcal{N}\left(0, \sigma_{\varepsilon,S_i}^2\right), \tag{8.4}$$

where $\mathbf{x}_i^f$ are the fixed effects, whereas $\mathbf{x}_i^r$ are the random effects, see Frühwirth-Schnatter (2006, Section 8.4). A necessary condition for identifiability is that the columns of the design matrix defined by

$$\mathbf{X} = \begin{pmatrix} \mathbf{x}_1^f & \mathbf{x}_1^r \\ \vdots & \vdots \\ \mathbf{x}_N^f & \mathbf{x}_N^r \end{pmatrix}$$

are linearly independent.

### 8.3.1 Defining a Mixed-Effects Finite Mixture Regression Model

#### Specifying the Model Structure

To define a standard finite mixture regression model, create a structure array, named for instance `mixreg`, containing the following fields:

- The field `dist` defines the parametric distribution family of the regression model. In the current version of the package only dependent data from a normal distribution are considered, thus `dist='Normal'`.
- The field `K` contains the number $K$ of regimes.
- The field `d` defines the dimension of the regression parameter.
- The field `indexdf` is a `fd x 1` integer array defining which columns of the design matrix (i.e. which rows of the data matrix stored in field `X`) correspond to the fixed effects.

If the field `indexdf` is missing, this model definition reduces to a finite mixture of multiple regression models, see Subsection 8.2, where all regression parameters are switching. If the field `K` is missing, this model definition reduces a standard regression model.

#### Assigning Parameters

For a fully specified finite mixture regression model, values have to be assigned to all model parameters:

- The field `par` contains the coefficients of the regression model. This is a structure array with following fields:
  - `beta` is a `(d-fd) x K` numeric array containing the switching regression parameters;
  - `alpha` is a `fd x 1` numeric array containing the fixed regression parameters;
  - `sigma` is a `1 x K` numeric array containing the error variances.
- The field `weight` contains the weight distribution $\boldsymbol{\eta} = (\eta_1, \ldots, \eta_K)$, characterized by a `1 x K` numeric array. This field is missing, if $K = 1$.

### 8.3.2 Getting Started Quickly

A demo is available that demonstrate how to fit mixed-effects mixtures of regression models to simulated data, see Subsection 8.5.1 for details on how to simulate data from a finite mixture of regression models:

- `demo_mixreg_mixeffects`: fits a standard regression model and mixed-effects finite mixtures of regression models with $K = 2$ and $K = 3$ to data that are simulated from a mixed-effects finite mixture of two regression models (takes about 5 CPU minutes).

### 8.3.3 Choosing Priors for Bayesian Estimation

It is assumed that the priors of all parameters but $\boldsymbol{\alpha}$ are the same as in Subsection 8.2.3, whereas

$$\boldsymbol{\alpha} \sim \mathcal{N}_r\left(\mathbf{a}_0, \mathbf{A}_0\right). \tag{8.5}$$

$\boldsymbol{\alpha}$ and $\boldsymbol{\beta}_k$ are assumed to be pairwise independent a priori. Thus the joint prior on $\boldsymbol{\alpha}^* = (\boldsymbol{\alpha}, \boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_K)$ is a normal prior, $\boldsymbol{\alpha}^* \sim \mathcal{N}_{r^*}\left(\mathbf{a}_0^*, \mathbf{A}_0^*\right)$, where $r^* = r + Kd$ and $\mathbf{a}_0^*$ and $\mathbf{A}_0^*$ are derived from $\mathbf{a}_0, \mathbf{A}_0$, $\mathbf{b}_0$, and $\mathbf{B}_0$ in an obvious way.

#### The Default Prior

The toolbox allows an automatic selection of a slightly data dependent, rather noninformative proper prior by calling the function `priordefine` introduced in Subsection 4.2.1. The prior for $\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_K$ is the same as in (8.3) and the following hyperparameters are added for the prior on $\boldsymbol{\alpha}$: $\mathbf{A}_0 = 10\mathbf{I}_r$, and $a_{0,j} = \overline{y}$ with $\overline{y}$ being the sample mean of the dependent variable, if $\alpha_j$ is a constant intercept, and $a_{0,j} = 0$, otherwise.

#### The Structure of the Prior

The prior is a structure array as in Subsection 8.2.3, where the field `par` is a structure array with fields `alpha`, `beta` and `sigma`, respectively. The field `sigma` is exactly the same as in Subsection 8.2.3:

- The field `beta` is a structure array with the fields `b` and `Binv` being, respectively, a `(d-fd) x K` and a `(d-fd) x (d-fd) x K` numerical array specifying the parameters $\mathbf{b}_{0,k}$ and $\mathbf{B}_{0,k}^{-1}$ of prior (8.2).
- The field `alpha` is a structure array with the fields `a` and `Ainv` being, respectively, a `fd x 1` and a `fd x fd` numerical array specifying the parameters $\mathbf{a}_0$ and $\mathbf{A}_0^{-1}$ of prior (8.5).

### 8.3.4 Bayesian Inference When the Allocations Are Unknown

In the package MCMC estimation is carried out using data augmentation and Gibbs sampling using Algorithm 8.2 described in Frühwirth-Schnatter (2006, Subsection 8.4.4)). Under the normal prior on the regression coefficients $\boldsymbol{\alpha}^* = (\boldsymbol{\alpha}, \boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_K)$, $\boldsymbol{\alpha}^* \sim \mathcal{N}_{r^*}\left(\mathbf{a}_0^*, \mathbf{A}_0^*\right)$ discussed in Subsection 8.3.3, the joint posterior of $\boldsymbol{\alpha}^*$, conditional on knowing the variance parameters $\sigma_{\varepsilon,1}^2, \ldots, \sigma_{\varepsilon,K}^2$, is again a normal distribution: $\boldsymbol{\alpha}^* | \sigma_{\varepsilon,1}^2, \ldots, \sigma_{\varepsilon,K}^2, \mathbf{y}, \mathbf{S} \sim \mathcal{N}_{r^*}\left(\mathbf{a}_N^*, \mathbf{A}_N^*\right)$. $\mathbf{a}_N^*$ and $\mathbf{A}_N^*$ are given by:

$$(\mathbf{A}_N^*)^{-1} = (\mathbf{A}_0^*)^{-1} + \sum_{i=1}^{N} \frac{1}{\sigma^2_{\varepsilon,S_i}} \boldsymbol{Z}_i' \boldsymbol{Z}_i, \tag{8.6}$$

$$\mathbf{a}_N^* = \mathbf{A}_N^* \left( (\mathbf{A}_0^*)^{-1} \mathbf{a}_0^* + \sum_{i=1}^{N} \frac{1}{\sigma^2_{\varepsilon,S_i}} \boldsymbol{Z}_i' y_i \right), \tag{8.7}$$

where $\boldsymbol{Z}_i = (\mathbf{x}_i^f \ \mathbf{x}_i^r D_{i1} \ \cdots \ \mathbf{x}_i^r D_{iK})$ and $D_{ik} = I_{\{S_i=k\}}$.

To run data augmentation and MCMC for data stored in `data` for the mixed-effects mixture regression model `mixreg` under prior `prior`, call the function `mixturemcmc` explained in Subsection 4.3:

```
mcmcout=mixturemcmc(data,mixreg,prior,mcmc);
```

The structure of the MCMC output is explained in full detail in Subsection 8.3.5. One may call the function `mcmcstart` explained in Subsection 4.3 before starting MCMC to make use of default starting values which are determined exactly as in Subsection 8.2.4.

### 8.3.5 MCMC Output

The MCMC output is stored in the structure array `mcmcout` having similar fields as in Subsection 8.2.5 with following modifications:

- The field `alpha` is added to `par`. This is a `M x fd` numerical array storing the posterior draws $\boldsymbol{\alpha}^{(m)}$.
- `beta` is a `M x (d-fd) x K` numerical array storing the posterior draws $\boldsymbol{\beta}_k^{(m)}$.

`post.par` is a structure array with the fields `alpha` , `beta` and `sigma`. The field `post.par.sigma` is the same as in Subsection 8.2.5. To reduce the dimension of the covariance matrix of the $\mathcal{N}_{r^*}(\mathbf{a}_N^*, \mathbf{A}_N^*)$ posterior derived in (8.6) for sampling $\boldsymbol{\alpha}^* = (\boldsymbol{\alpha}, \boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_K)$, the moments of the marginal distributions of $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}_k$ rather than the moments of the joint distribution are stored. The field `post.par.alpha` is a structure array with following fields:

- `a` is a `M x fd` numerical array storing that part of the mean $\mathbf{a}_N^*$ of the normal posterior $\mathcal{N}_{r^*}(\mathbf{a}_N^*, \mathbf{A}_N^*)$ that corresponds to $\boldsymbol{\alpha}$.
- `A` is a `M x fd x fd` numerical array storing that part of the covariance matrix $\mathbf{A}_N^*$ of the normal posterior $\mathcal{N}_{r^*}(\mathbf{a}_N^*, \mathbf{A}_N^*)$ that corresponds to the marginal distribution of $\boldsymbol{\alpha}$.

The field `post.par.beta` is a structure array with following fields:

- `b` is a `M x (d-fd) x K` numerical array storing for each group $k$ that part of the mean $\mathbf{a}_N^*$ of the normal posterior $\mathcal{N}_{r^*}(\mathbf{a}_N^*, \mathbf{A}_N^*)$ that corresponds to $\boldsymbol{\beta}_k$.
- `B` is a `M x (d-fd) x (d-fd) x K` numerical array storing for each group $k$ that part of the covariance matrix $\mathbf{A}_N^*$ of the normal posterior $\mathcal{N}_{r^*}(\mathbf{a}_N^*, \mathbf{A}_N^*)$ that corresponds to the marginal distribution of $\boldsymbol{\beta}_k$.

## 8.4 Finite Mixtures of Generalized Linear Models

Finite mixtures of generalized linear models (GLMs) extend the finite mixture of regression models discussed in Section 8.2 and 8.3 to nonnormal data.

### Finite Mixtures of Poisson Regression Models

Let $Y_i$ denote the $i$th response variable, observed in reaction to covariates $\mathbf{x}_i$, including 1 for the intercept. It is assumed that the marginal distribution of $Y_i$ follows a mixture of Poisson distributions,

$$Y_i \sim \sum_{k=1}^{K} \eta_k \mathcal{P}\left(\mu_{k,i}\right), \tag{8.8}$$

where $\mu_{k,i} = \exp(\mathbf{x}_i \boldsymbol{\beta}_k)$. If exposure data $e_i$ are available for each subject, then $\mu_{k,i} = e_i \exp(\mathbf{x}_i \boldsymbol{\beta}_k)$. If $\mathbf{x}_i = 1$, a finite mixture of Poisson distributions with $\mu_k = \exp(\boldsymbol{\beta}_k)$ results; if $K = 1$, the standard Poisson regression model results.

### Finite Mixtures of Negative Binomial Regression Models

It is assumed that $Y_i$ follows a mixture of Poisson distributions with random intercept,

$$Y_i | S_i = k \sim \mathcal{P}\left(\mu_{k,i} \lambda_i\right), \tag{8.9}$$

where

$$\lambda_i | S_i = k \sim \mathcal{G}\left(\delta_k, \delta_k\right), \tag{8.10}$$

and $\mu_{k,i} = \exp(\mathbf{x}_i \boldsymbol{\beta}_k)$. If exposure data $e_i$ are available for each subject, then $\mu_{k,i} = e_i \exp(\mathbf{x}_i \boldsymbol{\beta}_k)$.

The marginal distribution is a mixture of negative binomial distributions:

$$p(y_i | \boldsymbol{\vartheta}) = \sum_{k=1}^{K} \eta_k \binom{\delta_k + y_i - 1}{\delta_k - 1} \left(\frac{\delta_k}{\delta_k + \mu_{k,i}}\right)^{\delta_k} \left(\frac{\mu_{k,i}}{\delta_k + \mu_{k,i}}\right)^{y_i}. \tag{8.11}$$

If $\mathbf{x}_i = 1$, a finite mixture of negative binomial distributions with $\mu_k = \exp(\boldsymbol{\beta}_k)$ results; if $K = 1$, the standard negative binomial regression model results.

**Finite Mixture Regression Models for Binary and Binomial Data**

Let $Y_{i,t}$ denote a binary variable, observed on $T_i$ occasions in reaction to covariates $\mathbf{x}_i$, including 1 for the intercept. Define $Y_i = \sum_{t=1}^{T_i} Y_{i,t}$. It is assumed that the marginal distribution of $Y_i$ follows a mixture of binomial distributions,

$$Y_i \sim \sum_{k=1}^{K} \eta_k \text{BiNom}\left(T_i, \pi_{k,i}\right), \tag{8.12}$$

where logit $\pi_{k,i} = \mathbf{x}_i \boldsymbol{\beta}_k$.

### 8.4.1 Defining a Finite Mixture of GLMs

If exposure data $e_1, \ldots, e_N$ should be included in the analysis or if the data result from repeated measurements $T_1, \ldots, T_N$, then as in Section 7.1 this information should be stored in the field `Ti` of the structure array defining the data.

**Specifying the Model Structure**

To define a finite mixture of GLMs, create a structure array, named for instance `mixglm`, containing the following fields:

- The field `dist` defines the parametric distribution family of the GLM. In the current version of the package following distributions are available:
  - `'Poisson'`: Poisson distribution as in (8.8);
  - `'Binomial'`: binomial distribution as in (8.12);
  - `'Negative Binomial'`: negative binomial distribution as in (8.11).
- The field `K` contains the number $K$ of regimes.
- The field `d` defines the dimension of the regression parameter.
- The field `indexdf` is a `fd x 1` integer array defining which columns of the design matrix correspond to the fixed effects.

If the field `indexdf` is missing, this model definition reduces to a finite mixture of GLMs, where all regression parameters are switching. If the field `K` is missing, this model definition reduces a standard GLM.

**Assigning Parameters**

For a fully specified model, values have to be assigned to all model parameters:

- The field `par` contains the coefficients of the regression model. This is a structure array with following fields:
  - `beta` is a `(d-fd) x K` numeric array containing the switching regression parameters;

  - `alpha` is a `fd x 1` numeric array containing the fixed regression parameters, if present.

  For mixture of negative binomial distributions an additional field has to added to `par`, which defines the parameter $\delta_k$ in (8.10):
  - `df` a `1 x K` numeric array containing the parameters $\delta_1, \ldots, \delta_K$.
- The field `weight` contains the weight distribution $\boldsymbol{\eta} = (\eta_1, \ldots, \eta_K)$, characterized by a `1 x K` numeric array. This field is missing, if $K = 1$.

### 8.4.2 Getting Started Quickly

Several demos are available, that demonstrate how to fit mixtures of GLMs to real and simulated data, see Subsection 8.5.1 for more details how the data are simulated:

- `start_fabricfault.m` fits a Poisson regression model as well as mixtures of Poisson regression models with $K = 2$ to $K = 3$ to the Fabric Fault Data (takes about 7 CPU minutes), see also Subsection 1.2.4.
- `start_fabricfault_mixed_effects.m` fits a Poisson regression model as well as mixtures of Poisson regression models with $K = 2$ to $K = 3$ where the slope is fixed to the Fabric Fault Data (takes about 7 CPU minutes), see also Subsection 1.2.4.
- `start_fabricfault_negbin.m` fits a negative binomial regression model as well as mixtures of negative binomial regression models with $K = 2$ and $K = 3$ to the Fabric Fault Data (takes about 8 CPU minutes), see also Subsection 1.2.4.
- `demo_regression_mix_binomial.m` fits a mixture of two binomial regression models to simulated data (takes about 3 CPU minutes).

### 8.4.3 Choosing Priors for Bayesian Estimation

It is assumed that the prior for $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}_k$ are the same as in Subsection 8.2.3 and 8.3.3, respectively.

For the negative binomial distribution, additionally a prior for $\delta_k$ has to be selected. Bayesian estimation is based on assuming prior independence between $\delta_k$ and the remaining component specific parameters. The prior $p(\delta_k)$ has to be selected carefully, in order to avoid improper posteriors. In this package following prior is used:

$$p(\delta_k) \propto \frac{\delta_k^{a_0-1}}{(\delta_k + d)^{a_0+b_0}}, \tag{8.13}$$

where $a_0$, $b_0$, and $d$ are hyperparameters selected by the user.

### The Default Prior

The toolbox allows an automatic selection of a slightly data dependent, rather noninformative proper prior by calling the function `priordefine` introduced in Subsection 4.2.1. For the regression parameters this prior is the same as in Subsection 8.2.3 and 8.3.3.

For the negative binomial distribution, the default choice for the prior $p(\delta_k)$ defined in (8.13) is $a_0 = b_0 = 2$, and $d = 10$. For this prior choice, the prior median of $\delta_k$ is equal to 10, while the prior mean is equal to 20. This particular choice guarantees that the posterior distribution is proper and that the marginal posterior distribution of $\delta_k$ has a finite expectation. Thus the average of the MCMC draws may be used to estimate $\delta_k$.

### The Structure of the Prior

The prior is a structure array as in Subsection 8.2.3 and 8.3.3, however, no field `sigma` appears, because no unknown error variance is present in the model.

For the negative binomial distribution, the structure array `par` has an additional field to define the prior of $\delta_k$:

- The field `df` defines the prior for $\delta_1, \ldots, \delta_K$. It is a structural array with following fields:
  - The field `type` defines the type of prior used for $p(\delta_k)$ and is equal to `'hier'` for prior (8.13).
  - the field `a0` defines the hyperparameter $a_0$;
  - the field `b0` defines the hyperparameter $b_0$;
  - the field `d` defines the hyperparameter $d$.

## 8.4.4 Bayesian Inference When the Allocations Are Unknown

Various proposals have been put forward on how to estimate the unknown parameter $\boldsymbol{\vartheta}$ for finite mixtures of GLMs using MCMC under the assumption of a multivariate normal prior for the fixed and group specific regression parameters.

As the likelihood $p(\mathbf{y}|\boldsymbol{\vartheta})$ is available in closed form, one may use a single-move random walk Metropolis–Hastings algorithm as in Viallefont et al. (2002) or a multivariate random walk Metropolis–Hastings algorithm as is Hurn et al. (2003) to sample from the marginal posterior distribution $p(\boldsymbol{\vartheta}|\mathbf{y})$. To avoid time-consuming tuning of the underlying proposal densities, MCMC estimation is carried out in this package using data augmentation and auxiliary mixture sampling as described in Frühwirth-Schnatter et al. (2009).

For the negative binomial distribution, an additional step is added to sample $\delta_1, \ldots, \delta_K$. A partially marginalized sampler is used to draw $\delta_k$ by means of a Metropolis–Hastings algorithm from the marginal distribution $p(\delta_k|\mathbf{S}, \mathbf{y})$ which is available in closed form. The corresponding Metropolis-Hastings algorithm is based on the uniform log random walk proposal

$$\log(\delta_k^{new} - 1) \sim \mathcal{U}\left[\log(\delta_k - 1) - c_{\delta_k}, \log(\delta_k - 1) + c_{\delta_k}\right],$$

with fixed width parameter $c_{\delta_k}$. The width parameter $c_{\delta_k}$ has to be selected by the user prior to running MCMC, see below.

To run data augmentation and MCMC for a GLM call the function `mixturemcmc` explained in Subsection 4.3:

```
mcmcout=mixturemcmc(data,mixglm,prior,mcmc);
```

The structure of the MCMC output is explained in full detail in Subsection 8.4.5. One may call the function `mcmcstart` explained in Subsection 4.3 before starting MCMC to make use of default starting values. The function `mcmcstart` *does not choose* the variance $c_k$ of the random walk Metropolis-Hastings algorithm used for sampling $\delta_k$. These variances have to be stored prior to calling `mixturemcmc` by the user in an additional field of the array `mcmc` controlling MCMC:

- The field `mh.tune.df` is numerical array of size `1 x K` defining the variances $c_1, \ldots, c_K$ of the uniform log normal random walk proposals for $\delta_1, \ldots, \delta_K$.

### Default Starting Values

Unless stated otherwise (see filed `mcmc.startpar` in Subsection 4.3.1), MCMC estimation starts with sampling the parameters and the indicators stored in `data.S` are selected as starting value for the classification $\mathbf{S}^{(0)}$. For the negative binomial distribution starting values are needed for $\delta_1, \ldots, \delta_K$.

The function `mcmcstart` determines these starting values in the following way. Using the MATLAB function `kmeans`, $k$-means clustering is applied to the multivariate data where the data in `data.y` are merged with the regressors in `data.X`.

For the sake of comparison, one may start MCMC with sampling the indicators, in which case `mix` has to be a fully specified mixture before calling `mixturemcmc`. To determine starting values for a fully specified mixture, again the function `mcmcstart` may be called, however with the additional input argument `mcmc`, where `mcmc.startpar=true`. The starting values for all regression coefficients except the coefficient $\beta_{k,j}$ corresponding to the intercept are set equal to 0. For $K > 1$, starting values for $\beta_{k,j}$ are sampled from $\mathcal{N}\left(\overline{y}, s_y^2\right)$, while $\beta_{1,j} = \overline{y}$ for $K = 1$. The starting value for the weight distribution is uniform, i.e. $\eta_k = 1/K$.

For the negative binomial distribution, the starting values for $\delta_1, \ldots, \delta_K$ are equal to 5 for both ways of running MCMC.

### 8.4.5 MCMC Output

The MCMC output is stored in the structure array `mcmcout` and has similar fields as in Subsections 8.2.5 and 8.3.5, however neither `par` nor `post.par`

contain a field named `sigma`, because no unknown error variance is present in the model. Additionally, no field `hyper` is present.

For the negative binomial distribution, the following fields are added:

- The field `mh` provides details about the Metropolis-Hastings algorithm. It is a structural array with two fields:
  - The field `tune` is a `1 x K` array containing the tuning parameters for the log random walk Metropolis Hastings algorithm, see Subsection 8.4.4.
  - The field `acc` is a `1 x K` array containing the acceptance rates for the log random walk Metropolis Hastings algorithm for the degrees of freedom parameters $\delta_1, \ldots, \delta_K$.
- The field `par` has an additional field containing the MCMC draws for $\delta_1, \ldots, \delta_K$:
  - The field `df` is a `M x K` numerical array storing the posterior draws $\delta_k^{(m)}$.

## 8.5 Further Issues

### 8.5.1 Simulate from a Finite Mixture of Multiple Regression Models

To simulate $N$ observations $\mathbf{y} = (y_1, \ldots, y_N)$ from a finite mixture regression model use the function `simulate`, introduced in Subsection 3.3. The finite mixture regression model has to be fully specified model named, e.g. `mixreg`, and is defined as structural array as described in Subsection 8.2.1 for finite mixture regression models, in Subsection 8.3.1 for mixed-effect finite mixture regression models and in Subsection 8.4.1 for mixtures of GLMs.

The way the function `simulate` is called depends on whether a design matrix is available or nor. If a design matrix is available, then the function is called as

```
data=simulate(mixreg,N,data),
```

where the design matrix has to be stored in `data.X` by row meaning that `data.X` is a `mixreg.d x N` numerical array where each row correspond to a certain covariate. If the function is called without a design, a random design is simulated, where all covariates are drawn from uniform distribution $\mathcal{U}[a, b]$ and the last column of the design matrix corresponds to the intercept. For a normal regression model $a = 0, b = 1$, for the Poisson and the negative binomial distribution $a = 0.5, b = 1$, and for the binomial distribution $a = -1, b = 2$.

The function `simulate` produces the structural array `data` with the same fields as empirical data, see Subsection 3.1.1, including the field `y`, `N`, `X`, `r`, `sim`, `type` and `model`. Note that the data and the design will be generated as stored by row. The field `model` is simply a copy of the structural array `mixreg` used

for simulation. If $K > 1$, an additional field called `S` will be added containing the true allocations $\mathbf{S} = (S_1, \ldots, S_N)$. This is a `1 x N` numeric array, thus `data.S(i)` is the allocation $S_i$ of the $i$th observation $\mathbf{y}_i$.

### 8.5.2 Plotting MCMC

The function `mcmcplot` explained in Subsection 4.5.1 could be used to plot and monitor the MCMC output. The following sampling representations of the posterior draws are produced. For each possible combinations $(j, j')$, the simulated regression parameter $\beta_{k,j}^{(m)}$ is plotted against $\beta_{k,j'}^{(m)}$ for $k = 1, \ldots, K$.

### 8.5.3 Simulation-Based Approximations of the Marginal Likelihood

To compute the log of the marginal likelihood for a finite mixture of regression models (with or without mixed-effects) as discussed in Frühwirth-Schnatter (2006, Subsection 8.3.6) call the function

```
est=mcmcbf(data,mcmcout);
```

see Section 5.3 for more details. The marginal likelihood is available both for normal and for generalized linear mixture regression models.

### 8.5.4 Parameter Estimation

To perform parameter estimation, call the function `mcmcestimate` introduced in Subsection 4.5.2 after calling the function `mixturemcmc` with a structure array, say `mcmcout`, containing the MCMC output as input argument:

```
est=mcmcestimate(mcmcout);
```

The estimators of the weight distribution $\eta_1, \ldots, \eta_K$ are stored in

- `est.pm.weight` – (approximate) posterior mode estimator.
- `est.ml.weight` – (approximate) maximum likelihood estimator.
- `est.ident.weight` – ergodic average after identification
- `est.average.weight` – ergodic average without identification, if the draws were not generated by the permutation sampler (`mcmcout.ranperm` is `false`).
- `est.invariant.weight` – ergodic average without identification, if the draws were generated by the permutation sampler (`mcmcout.ranperm` is `true`).

The estimators of the switching regression parameters $\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_K$ are stored in

- `est.pm.par.beta` – (approximate) posterior mode estimator.

- `est.ml.par.beta` – (approximate) maximum likelihood estimator.
- `est.ident.par.beta` – ergodic average after identification
- `est.average.par.beta` – ergodic average without identification, if the draws were not generated by the permutation sampler (`mcmcout.ranperm` is `false`).
- `est.invariant.par.beta` – ergodic average without identification, if the draws were generated by the permutation sampler (`mcmcout.ranperm` is `true`).

The estimators of the fixed regression parameter $\boldsymbol{\alpha}$, if any, are stored in

- `est.pm.par.alpha` – (approximate) posterior mode estimator.
- `est.ml.par.alpha` – (approximate) maximum likelihood estimator.
- `est.ident.par.alpha` – ergodic average after identification
- `est.average.par.alpha` – ergodic average without identification, if the draws were not generated by the permutation sampler (`mcmcout.ranperm` is `false`).
- `est.invariant.par.alpha` – ergodic average without identification, if the draws were generated by the permutation sampler (`mcmcout.ranperm` is `true`).

For regression models based on the normal distribution, the estimators of the groups variances $\sigma_1^2, \ldots, \sigma_K^2$ are stored in

- `est.pm.par.sigma` – (approximate) posterior mode estimator.
- `est.ml.par.sigma` – (approximate) maximum likelihood estimator.
- `est.ident.par.sigma` – ergodic average after identification
- `est.average.par.sigma` – ergodic average without identification, if the draws were not generated by the permutation sampler (`mcmcout.ranperm` is `false`).
- `est.invariant.par.sigma` – ergodic average without identification, if the draws were generated by the permutation sampler (`mcmcout.ranperm` is `true`).

### 8.5.5 Clustering

To carry out clustering of the observations, call the function

```
clust=mcmcclust(data,mcmcout);
```

introduced in Subsection 4.5.3. To visualize clustering call the function

```
mcmcclustplot(data,clust,[nfig]);
```

introduced in the same subsection. A special plot is produced for mixtures of regression models. Clustering is visualized by plotting each observed regressor against the observed $y_i$ and marking group membership through group specific colors.

### 8.5.6 Bayesian Inference When the Allocations Are Known

In rare cases, e.g. for grouped data, will the allocations be known. In this case, the structure array `data` storing the data has to include the field `S`, storing the allocations, see also Subsection 4.4. In the context of regression modeling of such data, it is possible to assume that all regression coefficients are group-specific as in Section 8.2.1 or that some regression coefficients are the same in all groups as in Section 8.3. It is also possible to fit GLMs as In Section 8.4.1.

For a complete-data Bayesian estimation as discussed in Frühwirth-Schnatter (2006, Subsection 8.3.2 and 8.4.3) you need first to define a prior on the parameters, stored in a structure array `prior` as described in Subsection 8.3.3, Subsection 8.2.3 and Subsection 8.4.3, respectively.

Under these priors no closed form posterior is available, even if the allocations are known. To sample from the posterior distribution one could run complete-data Gibbs sampling using the function `mixturemcmc`, however, the allocations have to fixed beforehand:

```
mixreg.indicfix=true;
mcmcout=mixturemcmc(data,mixreg,prior,mcmc);
```

The allocations will not be updated during MCMC and no random permutation will be performed, even if `mcmc.ranperm` is set `true`. Starting values are selected automatically as described in Subsection 8.2.4, Subsection runmcmcregmix and Subsection 8.4.4, respectively.

Even, if the allocations are known, simulation-based approximations of the appropriate marginal likelihood could be computed using the function `mcmcbf`.

# 9

# Markov Switching Models for Time Series Data

The toolbox allows to fit finite Markov mixture models to time series data. To this aim, it is necessary to specify the Markov mixture model.

## 9.1 Data Handling

Let $\{y_t, t = 1, \ldots, T\}$ denote a time series of $T$ univariate observations taking values in a sampling space $\mathcal{Y}$ which may be either discrete or continuous. Time series are stored in form of a structure array where the field `y` contains the observations. For univariate time series `y` is a `1 x N` numeric array, where `N` is the number of observations. If the structure array is named, for instance, `data`, then `data.y(t)` is equal to the $t$th observation $y_t$. If `bycolumn` is true (see below), then `y` is a `N x 1` numeric array. To distinguish time series data from a sequence of observations, the optional field

- `istimeseries` could be added and set to true.

If this field is missing, it is assumed that the data are not a time series. Further optional fields are the same as in Subsection 3.1.1.

### Time Series Available in the Package

The following time series are stored under particular names and could be loaded into a structure array using the function `data=dataget(name)`:

- LAMB DATA: `data=dataget('lamb')`;
- GDP DATA: `data=dataget('gdp')`.

### Simple Plotting

If the field `istimeseries` is true, then the function `dataplot` introduced in Subsection 3.2.1 produces a time series plot and a plot of the autocorrelation of the time series itself and the squared time series, additionally to the marginal distribution.

#### Empirical Moments

For a time series stored in structural array named `timeseries`, for example, with fields as in Section 9.1, the function `datamoments(timeseries)` described in Subsection 3.2.2 may be called to compute sample moments of the data:

```
moments=datamoments(timeseries)
```

This function returns a structural array with the same fields as for non-time series data, however, following fields are added for a time series data:

- `ac`, the empirical autocorrelation function up to 20 lags;
- `acsqu`, the empirical autocorrelation function of the squared process up to 20 lags.

## 9.2 Finite Markov Mixture Models

Frühwirth-Schnatter (2006, Chapter 10) provides an introduction into finite Markov mixture modelling. Let $\{y_t, t = 1, \ldots, T\}$ denote a time series of $T$ univariate observations taking values in a sampling space $\mathcal{Y}$ which may be either discrete or continuous. The time series $\{y_t, t = 1, \ldots, T\}$ is considered to be the realization of a stochastic process $\{Y_t\}_{t=1}^{T}$ where the probability distribution of $Y_t$ depends on the realizations of a hidden discrete stochastic process $S_t$.

For each $t \geq 1$, the distribution of $Y_t$ arises from one out of $K$ distributions $\mathcal{T}(\boldsymbol{\theta}_1), \ldots, \mathcal{T}(\boldsymbol{\theta}_K)$, depending on the state of $S_t$. Whereas the specification of the conditional distribution of $Y_t$ given $S_t$ is closely related to previous chapters, the distribution of $S_t$ has now to be specified explicitly.

The stochastic properties of $S_t$ are sufficiently described by the $(K \times K)$ transition matrix $\boldsymbol{\xi}$, where each element $\xi_{jk}$ of $\boldsymbol{\xi}$ is equal to the transition probability from state $j$ to state $k$:

$$\xi_{jk} = \Pr(S_t = k | S_{t-1} = j), \qquad \forall j, k \in \{1, \ldots, K\},$$

see Frühwirth-Schnatter (2006, Section 10.2).

### 9.2.1 Defining Finite Markov Mixture Models

#### Specifying the Model Structure

A finite Markov mixture is defined as a structure array as described in Subsection 2.2.1, named for instance `model`, however, the field `indicmod` has to be added to specify the probability distribution of $S_t$:

- The field `dist` shows the parametric distribution family $\mathcal{T}(\boldsymbol{\theta})$ characterized by a string variable. The current version of the package is able to handle the following distribution families:
  - `'Poisson'`: Poisson distribution $\mathcal{P}(\mu_k)$,
  - `'Normal'`: normal distribution $\mathcal{N}\left(\mu_k, \sigma_k^2\right)$.

  The package will check just the first six characters, therefore the types may be abbreviated.
- The field `K` contains the number $K$ of states of the hidden indicator $S_t$.
- The field `indicmod` specifies the distribution of $S_t$ and is a structural array with the following fields:
  - The field `dist` specifies the distribution of $S_t$. The following options are available:
    - `'Markovchain'`: $S_t$ a hidden Markov chain with unknown transition matrix `indicmod.xi` (see below);
    - `'Multinomial'`: $S_t$ is an i.i.d. sequence with unknown distribution `weight`.

    Because the package checks just the first six characters, the types may be abbreviated. Under the option `'Multinomial'` the model reduces to a finite mixture model.
  - The field `init` specifies the initial distribution of $S_0$. The following options are available, see Frühwirth-Schnatter (2006, Section 10.3.4):
    - `'ergodic'`: the initial distribution is equal to the ergodic distribution;
    - `'uniform'`: the initial distribution is equal to the uniform distribution;

    Because the package checks just the first three characters, the types may be abbreviated. The default choice is the option `'ergodic'`, if this field is missing.

The model reduces to a standard finite mixture model, if the field `indicmod` is missing.

### Assigning Parameters

Parameters are assigned in the following way:

- The field `par` contains the component parameters $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_K$. The structure of this field depends on the distribution family and on the dimension of $\boldsymbol{\theta}_k$. For Poisson Markov mixtures, the field `par` is a `1 x K` numeric array, containing the component parameters $\mu_1, \ldots, \mu_K$. For Markov mixtures of normal distributions, `par` is defined as in Subsection 6.2.1.
- The transition matrix $\boldsymbol{\xi}$ is a parameter of the indicator model specified in the field `indicmod` and therefore added to that field. Thus the field `indicmod.xi` contains the transition matrix $\boldsymbol{\xi}$, characterized by a `K x K` numeric array.

### 9.2.2 Getting Started Quickly

Several demos are available that demonstrate how to fit finite Markov mixtures models to real data:

- `start_lamb.m`: fits a Markov mixture of Poisson distributions to the LAMB DATA (takes about 7 CPU minutes), see also Subsection 1.2.5.
- `start_gdp_marmix.m`: fits a Markov mixture of normal distributions to the GDP DATA (takes about 7 CPU minutes).

Bayesian estimation using MCMC and prior choices are discussed in Section 9.7.

### 9.2.3 Simulate from a Finite Markov Mixture Distribution

To simulate a time series of length $N = T$ observations from a finite Markov mixture distribution define the Markov mixture through a structure array, say `marmix`, as described in Subsection 9.2.1 and use the function `simulate`, introduced in Subsection 3.3:

```
timeseries=simulate(marmix,N);
```

The initial value $S_0$ is simulated from the initial distribution specified by the field `indicmod.init`. It is also possible to fix the initial value $S_0$ by choosing `indicmod.init` in the following way before calling `simulate`:

- `'fixX'`: the initial value $S_0$ is equal to `X`.

The function `simulate` creates a structural array `timeseries` with the same fields as an empirical time series, see Subsection 9.1, including the field `y`, `N`, `r`, `sim`, `type` and `model`. Note that the data will be generated as stored by row. The field `model` is simply a copy of the structural array `marmix` used for simulation. Two additional fields are added for simulated data:

- `S` contains the true states $\mathbf{S} = (S_1, \ldots, S_N)$. This is a `1 x N` numeric array, thus `data.S(t)` is the state of the hidden Markov chain $S_t$ of the $t$th observation $y_t$.
- `S0` contains the true state $S_0$. This is a single numerical value.

### 9.2.4 Some Descriptive Features of Finite Markov Mixture Distributions

Because the unconditional distribution of a random process $Y_t$, being generated by a Markov mixture of $\mathcal{T}(\boldsymbol{\theta})$-distribution is a standard finite mixture of $\mathcal{T}(\boldsymbol{\theta})$-distribution with the ergodic probabilities acting as weights, all functions defined in Chapter 2 are applicable to finite Markov mixtures. To determine the invariant probability distribution $\boldsymbol{\eta}$ for a given transition matrix $\boldsymbol{\xi}$, see e.g. Frühwirth-Schnatter (2006, p.306), the utility function `eta=marstat(xi)` is called.

**Moments and Autocorrelation Functions**

The moments of the marginal distribution are obtained by calling the function `moments` as discussed in Subsection 2.2.4, see also Frühwirth-Schnatter (2006, Subsection 10.2.3). The following additional fields are added for a Markov mixture model, when calling the function `moments`:

- `ergodic`, containing the ergodic distribution, stored as `1 x K`;
- `eigen`, containing the eigenvalues of $\boldsymbol{\xi}$, stored as `1 x K`;
- `duration`, expected duration of each state, stored as `1 x K`; see also Frühwirth-Schnatter (2006, Subsection 10.2.2).
- `ac`, the autocorrelation function up to 20 lags; see also Frühwirth-Schnatter (2006, Subsection 10.2.4).
- `acsqu`, the autocorrelation function of the squared process up to 20 lags; see also Frühwirth-Schnatter (2006, Subsection 10.2.5).

**Visualisation**

To plot the density of the unconditional distribution of a finite Markov mixture distribution, defined by the structure array `marmix` use the function `mixtureplot` described in Subsection 2.2.2. This function will also produce a bar plot of the autocorrelation function of $y_t$ and $y_t^2$.

## 9.3 The Markov Switching Regression Model

### 9.3.1 Defining the Markov Switching Regression Model

The Markov switching regression model is an extension of finite mixtures of regression models to time series data. For continuous data, the model reads:

$$Y_t = \mathbf{x}_t \boldsymbol{\beta}_{S_t} + \varepsilon_t, \qquad \varepsilon_t \sim \mathcal{N}\left(0, \sigma^2_{\varepsilon,S_t}\right), \tag{9.1}$$

where $S_t$ is a hidden Markov chain and $\mathbf{x}_t$ is a row vector of explanatory variables including the constant, see Frühwirth-Schnatter (2006, Subsection 10.3.2) for more details. A similar extension may be formulated for non-Gaussian time series.

The mixed-effects regression model, considered for continuous data in Section 8.3 could be extended in a similar way:

$$Y_t = \mathbf{x}_t^f \boldsymbol{\alpha} + \mathbf{x}_t^r \boldsymbol{\beta}_{S_t} + \varepsilon_t, \qquad \varepsilon_t \sim \mathcal{N}\left(0, \sigma^2_{\varepsilon,S_t}\right), \tag{9.2}$$

where $\mathbf{x}_t^f$ are the fixed effects, whereas $\mathbf{x}_t^r$ are the random effects, see Frühwirth-Schnatter (2006, Section 8.4). Again a similar extension may be formulated for non-Gaussian time series.

Both models are defined as a structural array exactly in the same way as for a hidden indicator $S_t$ which is multinomial, see Subsection 8.2.1 or Subsection 8.3.1 for more details. However, as for basic Markov mixture distributions, the field `indicmod` has to be added to specify the probability distribution of $S_t$, see again Subsection 9.2.1.

### 9.3.2 Getting Started Quickly

Several demos are available that demonstrate how to fit a Markov switching regression model to simulated data:

- `demo_msreg`: fits a standard regression model and Markov mixtures of regression models with $K = 2$ and $K = 3$ to data that are simulated from a Markov mixture of two regression models and selects and evaluates the model with the largest marginal likelihood (takes about 11 CPU minutes).
- `demo_msreg_mixeffects`: fits a standard regression model and a mixed-effects Markov mixture of regression models with $K = 2$ and $K = 3$ to data that are simulated from a mixed-effects Markov mixture of two regression models and selects and evaluates the model with the largest marginal likelihood (takes about 11 CPU minutes).

Bayesian estimation using MCMC and prior choices are discussed in Section 9.7.

### 9.3.3 Simulate from a Markov Switching Regression Model

To simulate $N = T$ observations $\mathbf{y} = (y_1, \ldots, y_N)$ from a Markov switching regression model, with or without fixed effects, define the model through a structure array, say `msreg` as described above and call

```
timeseries=simulate(msreg,N);
```

The hidden Markov chain is simulated as described in Subsection 9.2.3. Conditional on the indicators, simulation proceeds exactly as for finite mixtures of regression models, see Subsection 8.5.1.

## 9.4 The Markov Switching Autoregressive Model

Markov switching autoregressive (MSAR) models are discussed in Frühwirth-Schnatter (2006, Section 12.2) for univariate time series on continuous observations. In its most general form the MSAR model allows that the autoregressive coefficients are affected by the hidden Markov chain $S_t$:

$$Y_t = \zeta_{S_t} + \delta_{S_t,1} Y_{t-1} + \cdots + \delta_{S_t,p} Y_{t-p} + \varepsilon_t. \tag{9.3}$$

$\varepsilon_t$ is an error term with switching variance. A more specific model is obtained, if the autoregressive coefficients are state independent:

$$Y_t = \zeta_{S_t} + \delta_1 Y_{t-1} + \cdots + \delta_p Y_{t-p} + \varepsilon_t. \tag{9.4}$$

### 9.4.1 Defining the Markov Switching Autoregressive Model

An MSAR model is defined as a finite Markov mixture model through a structural array, named for instance `msarmodel`, with the same fields as described in Subsection 9.2.1. The field `ar` is added to specify the order of the model.

For statistical inference, an MSAR model is treated as a Markov switching regression model where some or all regression coefficients are switching, see again Subsection 9.3.

#### Switching AR Coefficients

If no further assumptions are made, then it is assumed that the AR coefficients are switching. For such a model, the model structure is specified in the following way:

- The field `dist` defines the parametric distribution family $\mathcal{T}(\boldsymbol{\theta})$ of $p(y_t|\boldsymbol{\theta}_k, y^{t-1})$. The current version of the package is able to handle the following distribution families:
  - `'Normal'`: normal distribution $\mathcal{N}\left(\mu_{k,t}, \sigma_k^2\right)$.

  The package will check just the first six characters, therefore the types may be abbreviated.
- The field `K` defines the number of states of $S_t$.
- The field `indicmod` specifies the distribution of $S_t$ as in Subsection 9.2.1.
- The field `ar` specifies the order of the autoregressive part.

The model reduces to a basic Markov mixture model, if the field `ar` is missing. Parameter values are assigned in the following way:

- `par` specifies the state specific parameters. This is a structural array with following fields:
  - `beta` contains the intercept and the switching AR coefficients. For a hidden Markov chain with $K$ states this is a `(1+ar) x K` numerical array.
  - `indexar` is a `ar x 1` array of indices defining which elements of `beta` correspond to the AR coefficients. If this field is missing, then it is assumed that the first element of `beta`, i.e. `beta(1,:)` corresponds to the switching intercept, whereas the remaining elements, i.e. `beta(2:end,:)` correspond to the switching AR coefficients.
  - The switching variance is stored in the field `sigma` being a `1 x K` numerical array.
- `indicmod.xi` contains the transition matrix $\boldsymbol{\xi}$ as in Subsection 9.2.1.

#### State Independent AR Coefficients

For an MSAR model where the autoregressive coefficients are state independent, the model structure is specified in the following way:

- The field `dist` defines the parametric distribution family $\mathcal{T}(\boldsymbol{\theta})$ of $p(y_t|\boldsymbol{\theta}_k, y^{t-1})$. The current version of the package is able to handle the following distribution families:
  - `'Normal'`: normal distribution $\mathcal{N}\left(\mu_{k,t}, \sigma_k^2\right)$.

  The package will check just the first six characters, therefore the types may be abbreviated.
- The field `K` defines the number of states of $S_t$.
- The field `indicmod` specifies the distribution of $S_t$ as in Subsection 9.2.1.
- The field `arf` specifies the order of the state independent autoregressive part.

The model reduces to a basic Markov mixture model, if the field `arf` is missing. Parameter values are assigned in the following way:

- `par` specifies the model parameters. This is a structural array with following fields:
  - `beta` contains the switching intercept. For a hidden Markov chain with $K$ states this is a `1 x K` numerical array.
  - `alpha` is a `arf x 1` numerical array containing the state independent AR coefficients.
  - The switching variance is stored in the field `sigma` being a `1 x K` numerical array.
- `indicmod.xi` contains the transition matrix $\boldsymbol{\xi}$ as in Subsection 9.2.1.

### Mixture AR Models and Standard AR Models

If in the above model definition the number of states is equal to one, then a standard AR model results. A finite mixture AR model results, if the indicator model is substituted by a standard finite mixture model, e.g. by leaving the field `indicmod` unspecified. These models may be tested against a Markov switching model through comparing marginal likelihoods, see Subsection 9.8.6.

## 9.4.2 Getting Started Quickly

Several demos are available, that demonstrate how to fit a Markov switching autoregressive model to real data:

- The program `start_gdp.m` fits Markov switching autoregressive models with different number of states and increasing AR order where all parameters are switching to the GDP Data (takes about 26 CPU minutes), see also Subsection 1.2.6.
- The program `start_gdp_swi.m` fits Markov switching autoregressive model with different number of states and increasing AR order where only the intercept is switching to the GDP Data (takes about 17 CPU minutes), see also Subsection 1.2.6.

Bayesian estimation using MCMC and prior choices are discussed in Section 9.7.

### 9.4.3 Simulate from a Markov Switching Autoregressive Model

To simulate a time series $y_1, \ldots, y_N$ of length $N$ from a Markov switching AR model define a fully specified model through a structure array, say `marmix`, as described in Subsection 9.4.1 and use the function `simulate` as in Subsection 9.2.3:

```
timeseries=simulate(marmix,N);
```

The starting value $S_0$ is simulated as described as Subsection 9.2.3. The structural array `timeseries` has the same structure as in Subsection 9.2.3.

For a Markov switching autoregressive model, starting values $y_{1-p}, \ldots, y_0$ are needed to simulate $y_1, \ldots, y_N$. If $y_{1-p}, \ldots, y_0$ are known values, like in simulation based forecasting, see Subsection 9.9, the structural array `timeseries` has to be created before calling `simulate` and the starting values have to be stored in the field

- `'ystart'`. This is a `1 x p` numerical array containing $y_{1-p}, \ldots, y_0$, where `p` is the order of the AR part.

Then structural `timeseries` has to be added as a third argument when calling the function `simulate`:

```
timeseries.ystart= ....
timeseries=simulate(marmix,N,timeseries);
```

If the function `simulate` is called with two arguments or with three arguments, but without a field named `ystart` added to the third argument, then $y_{1-p}, \ldots, y_0$ are set to the long range mean $\zeta_{S_0}/(1-\delta_{S_0,1}-\cdots-\delta_{S_0,p})$ in state $S_0$.

## 9.5 Markov Switching Dynamic Regression Models

The Markov switching dynamic regression model allows to combine the Markov switching regression model and the Markov switching autoregressive model, see Frühwirth-Schnatter (2006, Section 12.3). In its most general form the model reads:

$$Y_t = \delta_{S_t,1} Y_{t-1} + \cdots + \delta_{S_t,p} Y_{t-p} + \mathbf{x}_i^f \boldsymbol{\alpha} + \mathbf{x}_i^r \boldsymbol{\beta}_{S_t} + \zeta_{S_t} + \varepsilon_t.$$

### 9.5.1 Defining the Markov Switching Dynamic Regression Model

The model is defined in the following way:

- The field `dist` defines the parametric distribution family of the regression model. The current version of the package is able to handle the following distribution families:

  - `'Normal'`: normal distribution $\mathcal{N}\left(\mu_{k,t}, \sigma_k^2\right)$.

  The package will check just the first six characters, therefore the types may be abbreviated.
- The field `K` contains the number $K$ of regimes.
- The field `d` defines the dimension of the regression parameter, including the intercept (total number of columns in $\mathbf{x}_i^f$ and $\mathbf{x}_i^r$ plus 1).
- The field `indexdf` is a `fd x 1` integer array defining which elements in the regressor matrix `data.X` correspond to the fixed effects.
- `ar` specifies the order of the autoregressive part, if the AR coefficients are switching.
- `arf` specifies the order of the autoregressive part, if the AR coefficients are state-independent.

No fixed parameters are present, if the field `indexdf` is missing.

### Assigning Parameter Values

For a model with switching AR coefficients, parameter values are assigned in the following way:

- `par` specifies the model parameters. This is a structural array with following fields:
  - `beta` contains the switching intercept, the switching regression coefficients and the switching AR coefficients. This is a `(d-fd+ar) x K` numerical array.
  - `indexar` is a `ar x 1` array of indices defining which elements of `beta` correspond to the AR coefficients. If this field is missing, then it is assumed that the first `(d-fd)` element of `beta`, i.e. `beta(1:d-fd,:)` corresponds to the switching regression coefficients, whereas the remaining elements, i.e. `beta(end-ar:end,:)` corresponds to the switching AR coefficients.
  - `alpha` is a `fd x 1` numerical array containing the state independent regression coefficients.
  - For models based on the normal distributions the switching variance is stored in the field `sigma` being a `1 x K` numerical array.
- `indicmod.xi` contains the transition matrix $\boldsymbol{\xi}$.

If the autoregressive coefficients are state independent, then the parameters are assigned in the following way:

- `par` specifies the model parameters. This is a structural array with following fields:
  - `beta` contains the switching intercept and the switching regression coefficients. For a hidden Markov chain with $K$ states this is a `(d-fd) x K` numerical array.
  - `alpha` is a `(fd+arf) x 1` numerical array containing the state independent regression coefficients and the state independent AR coefficients.

  - The field `indexdf` is a `fd x 1` integer array defining which elements in the regressor matrix `data.X` correspond to the fixed effects.
  - `indexar` being a `ar x 1` array of indices defines which elements of `alpha` correspond to the AR coefficients. If this field is missing, then it is assumed that the first `fd` elements of `alpha`, i.e. `alpha(1:fd,1)` correspond to the fixed regression coefficients, whereas the remaining elements, i.e. `alpha(end-arf+1:end,1)` correspond to the fixed AR coefficients.
  - For models based on the normal distributions the switching variance is stored in the field `sigma` being a `1 x K` numerical array.
- `indicmod.xi` contains the transition matrix $\boldsymbol{\xi}$.

### 9.5.2 Getting Started Quickly

Several demos are available, that demonstrate how to fit a Markov switching dynamic regression model to simulated data:

- `demo_msar_reg`: fits a standard dynamic regression model and Markov switching dynamic regression models to data that are simulated from a Markov switching dynamic regression model and selects and evaluates the model with the largest marginal likelihood (takes about 11 CPU minutes).
- `demo_msar_reg_mixeffects`: fits a standard dynamic regression model and mixed-effects Markov switching dynamic regression models to data that are simulated from a mixed-effects Markov switching dynamic regression model and selects and evaluates the model with the largest marginal likelihood (takes about 11 CPU minutes).

Bayesian estimation using MCMC and prior choices are discussed in Section 9.7.

### 9.5.3 Simulating from the Markov Switching Dynamic Regression Model

To simulate a time series $y_1, \ldots, y_N$ of length $N = T$ from a Markov switching dynamic regression model, call the function `simulate` in the following way:

```
data=simulate(mixreg,N,data),
```

where the design matrix has to be stored in `data.X` by row. If the function is called without a design, a random design is simulated, where all covariates are drawn from uniform distribution and the last column of the design matrix corresponds to the intercept, see also Subsection 8.5.1. The field `ystart` has to be added, if starting values of $y_{1-p}, \ldots, y_0$ are available, see also Subsection 9.4.3. If no starting values are provided, these values are set to zero.

## 9.6 State Estimation for Known Parameters

Statistical inference on the states of the hidden Markov chain $\mathbf{S}$ for fixed state parameters and a known transition matrix is discussed in Frühwirth-Schnatter (2006, Section 11.2). To perform filtering of the states as in Algorithm 11.1 and smoothing of the states as in Algorithm 11.2, call the function

```
class=dataclass(timeseries,marmix);
```

where `timeseries` is a structure array containing the data, see Section 9.1, and `marmix` is a structure array defining a finite Markov mixture model, see Subsection 9.2.1. For state estimation, `mix` has to be a fully specified model. The function `dataclass` which is an extension of the corresponding function discussed for finite mixtures in Section 4.1 produces a structural array `class` with following fields:

- The field `t0`, if the model contains an autoregressive part and classification does not start with $t = t_0 = 1$, but with $t_0 > 1$.
- `prob` are the filtered state probabilities $\Pr(S_t = k|\mathbf{y}^t, \boldsymbol{\vartheta})$, $t = t_0, \ldots, T$, being equal to a `(data.N-t0+1) x K` numerical array, where the rows sum to 1.
- `mixlik` is the logarithm of the Markov mixture likelihood function $\log p(\mathbf{y}|\boldsymbol{\vartheta})$,

$$p(\mathbf{y}|\boldsymbol{\vartheta}) = \prod_{t=t_0}^{T} p(y_t|\mathbf{y}^{t-1}, \boldsymbol{\vartheta}).$$

  evaluated at $\boldsymbol{\vartheta}$ equals to the value of `marmix.par` and `marmix.indicmod.xi`.
- `probsmooth` are the smoothed state probabilities $\Pr(S_t = k|\mathbf{y}, \boldsymbol{\vartheta})$, $t = t_0, \ldots, T$, being equal to a `(data.N-t0+1) x K` numerical array, where the rows sum to 1.
- `entropy` is the entropy of the filtered state probabilities, defined by (Frühwirth-Schnatter, 2006, Subsection 2.2.2, pp. 28)

$$\mathrm{EN}(\boldsymbol{\vartheta}|\mathbf{y}) = -\sum_{t=t_0}^{T}\sum_{k=1}^{K} \Pr(S_t = k|\mathbf{y}^t, \boldsymbol{\vartheta}) \log \Pr(S_t = k|\mathbf{y}_t, \boldsymbol{\vartheta}). \quad (9.5)$$

### Computation of the Markov Mixture Likelihood

The function `dataclass` is also called in the package, if the primary aim is likelihood evaluation rather than classification, because the Markov mixture likelihood is a byproduct of computing the filtered state probabilities, see (Frühwirth-Schnatter, 2006, Subsection 11.4.1). In this case, the computation of the smoothed state probabilities is superfluous and may be suppressed by calling `dataclass` with a third input argument being equal to 0:

```
class=dataclass(timeseries,marmix,0);
```

This will speed up the computation of the likelihood considerably.

### Sampling Posterior Paths of the Hidden Markov Chain

In Section 9.7 a sampled path $\mathbf{S}^{(m)} = (S_{t_0-1}^{(m)}, S_1^{(m)}, \ldots, S_T^{(m)})$ of the hidden Markov chain is needed. This is obtained by calling `dataclass` with a second output argument:

```
[class, S]=dataclass(timeseries,marmix,0)
```

The output argument `S` is a `1 x (data.N-t0+2)` array containing the simulated states with `S(1)` being equal to the starting value $S_{t_0-1}^{(m)}$ and `S(t+1)` being equal to $S_{t+t_0}^{(m)}$. Furthermore, the following field is added to the structure array `class`:

- `postS` which is equal to the posterior density $p(\mathbf{S}^{(m)}|\mathbf{y}, \boldsymbol{\vartheta})$ of the simulated Markov chain.

Again, sampling is speeded up considerably by suppressing the computation of the marginal smoothed state probabilities $\Pr(S_t = k|\mathbf{y}, \boldsymbol{\vartheta})$.

## 9.7 Bayesian Parameter Estimation with Known Number of States

### 9.7.1 Choosing the Prior for the Parameters of a Markov Mixture Model

Frühwirth-Schnatter (2006, Subsection 11.5.1) discusses in details how to choose the prior for a Markov mixture model. The prior is the same as defined in Section 4.2.1 for finite mixture models, however, the prior on $\boldsymbol{\eta}$ has to be substituted by a prior on $\boldsymbol{\xi}$. It is assumed that the rows of $\boldsymbol{\xi}$ are independent a priori, each following a Dirichlet distribution:

$$\boldsymbol{\xi}_{k\cdot} \sim \mathcal{D}\left(e_{k1}, \ldots, e_{kK}\right), \qquad k = 1, \ldots, K. \tag{9.6}$$

To obtain a prior that is invariant to relabeling, Frühwirth-Schnatter (2001) suggested choosing $e_{kk} = e^P$ and $e_{kk'} = e^T$, if $k \neq k'$. By choosing $e^P > e^T$, the Markov switching model is bounded away from a finite mixture model.

The prior is defined through a structural array exactly as in Subsection 4.2.1, however, the field `weight` is substituted by the field

- `indicmod.xi`. This is a `K x K` numerical array containing the hyper parameters $e_{k1}, \ldots, e_{k,K}$ in the `k`th row.

### Automatic Prior Choices

The toolbox allows an automatic selection of slightly data dependent rather noninformative prior by calling the function `priordefine`

```
prior=priordefine(timeseries,marmix);
```

where `timeseries` is a structure array containing the data and `marmix` is a structure array defining the Markov mixture distribution which need not be fully specified. Only the fields `dist`, `K` and `indicmod`, with the latter being equal to `'Markov'`, are necessary.

The selected prior is a hierarchical independence prior, where

$$\begin{aligned} &\mathbf{b}_{0,k} = \mathbf{b}_0, \qquad \mathbf{B}_{0,k} = \mathrm{Diag}\left(B_{0,1}, \ldots, B_{0,d}\right), \qquad \mathbf{A}_0 = \mathrm{Diag}\left(A_{0,1}, \ldots, A_{0,r}\right), \\ &c_{0,k} = \nu_c, \qquad g_0 = 0.5, \qquad \mathbf{G}_0 = g_0 \phi (\nu_c - 1) s_y^2. \end{aligned} \tag{9.7}$$

$s_y^2$ is the sample variance of the dependent variable, $\nu_c = 2.5$, and $\phi = 0.5$. $b_{0,j} = \overline{y}$ with $\overline{y}$ being the sample mean of the dependent variable if $\beta_{k,j}$ is a switching intercept, and $b_{0,j} = 0$, otherwise. $a_j = \overline{y}$ if $\alpha_j$ is a constant intercept, and $a_j = 0$, otherwise. $B_{0,j} = 0.25$ if $\beta_{k,j}$ is a switching AR coefficient, and $B_{0,j} = 10$, otherwise. $A_{0,j} = 0.25$ if $\alpha_j$ is a state-independent AR coefficient, and $A_{0,j} = 10$, otherwise.

### 9.7.2 Parameter Estimation for Known States

Parameter estimation for known states is discussed in Frühwirth-Schnatter (2006, Section 11.3). Estimation of the state specific parameters is essentially the same as in Section 4.4. The only new step is complete-data Bayesian estimation of the transition matrix, see Frühwirth-Schnatter (2006, Subsection 11.3.3).

### 9.7.3 Parameter Estimation Through Data Augmentation and MCMC

Bayesian estimation of finite Markov mixtures using data augmentation and MCMC is discussed in great detail in Frühwirth-Schnatter (2006, Section 11.5). MCMC sampling is performed as described in Algorithm 11.3 in Frühwirth-Schnatter (2006, Subsection 11.5.3). Posterior paths of the hidden Markov chain are sampled using Algorithm 11.5 in Frühwirth-Schnatter (2006, Subsection 11.5.3). The method used for sampling the unknown transition matrix depends on the starting distribution of $S_0$. For stationary Markov chains where the ergodic distribution is used as starting distribution the Metropolis–Hastings algorithm is applied, whereas for all starting distributions that are independent of $\boldsymbol{\xi}$ Gibbs sampling is used, see Frühwirth-Schnatter (2006, Subsection 11.5.5). To run data augmentation and MCMC, call the function

```
mcmcout=mixturemcmc(timeseries,marmix,prior,mcmc);
```

where `timeseries` is a structure array containing the data, `marmix` is a structure array defining the Markov mixture distribution, which should be fitted, and `prior` is a structure array defining the prior distribution. `mcmc` is a

structural array controlling MCMC, see Subsection 4.3.1. Unless stated otherwise (see `mcmc.ranperm`), each sampling step is concluded by a random permutation step, see Algorithm 11.4 in Frühwirth-Schnatter (2006, Subsection 11.5.4). The structure of the MCMC output is explained in full detail at the end of this subsection.

It may take some time to execute MCMC sampling. MCMC estimation of a Markov mixture model is more time consuming than for a finite mixture model, because the hidden indicators are correlated and need to be sampled recursively. Each minute, the function `mixturemcmc` reports the expected remaining execution time.

One may call the function `mcmcstart` explained in Subsection 4.3 before starting MCMC to make use of default starting values. The remainder of this subsection explains, how this starting values are selected.

### Default Starting Values

If the logical field `startpar` has been set to `true` in the definition of `mcmc`, then sampling is started with drawing the allocations $\mathbf{S}$ conditional on $\boldsymbol{\vartheta}^{(0)}$. In this case the Markov switching model `marmix` needs to be fully specified.

Otherwise, MCMC estimation starts with sampling the parameters and the indicators stored in `data.S` will be selected as starting value for the classification $\mathbf{S}^{(0)}$. In this case, the Markov mixture model `marmix` need not be fully specified. In the model structure, the fields `dist` and `inidicmod` have to be specified in any case. The latter variable has to be equal to `'Markov'` to run MCMC estimation for a Markov mixture, rather than a finite mixture model. The field `ar` has to be added for a Markov switching autoregressive model with switching autoregressive coefficients, the field `arf` has to be added for a Markov switching autoregressive model with state independent autoregressive coefficients.

If sampling of $\boldsymbol{\theta}_k$ involves more than one block, starting values for some parameters are needed, which need to be stored in `par` before calling `mixturemcmc`. If a Markov mixture of normal distributions is estimated under an independence prior, a starting value for `par.mu` is needed. For Markov switching regression models or for MSAR models based on the normal distribution sampling of the regression and/or autoregressive coefficients and the error variances involves two blocks under an independence prior, where the first block samples the regression and/or autoregressive coefficients conditional on the error variances. Thus starting values for the error variances are needed which have to be stored in `mixmar.par.sigma` before calling the function `mixturemcmc`. Finally, if the initial distribution of $S_0$ is equal to the ergodic distribution, then a starting value for $\boldsymbol{\xi}$ is needed to run the Metropolis-Hastings algorithm for sampling $\boldsymbol{\xi}$.

For both ways of starting MCMC, you may call the function

```
[data,mixmar]=mcmcstart(data,mixmar,mcmc);
```

before starting MCMC to make use of default starting values. Automatic classification is based on the quantiles of the empirical marginal distribution of the time series.

Under a hierarchical prior the prior parameter, e.g. `prior.par.b` for a Markov mixture of Poisson distributions, has to be set to an appropriate starting value, for instance, the mean of the prior put on the random parameter. Under an automatic prior definition using the function `priordefine`, see Subsection 9.7.1 for more details, this value will automatically be chosen as starting value for MCMC estimation.

### MCMC Output

`mcmcout` is a structure array containing the MCMC draws and consists of the following fields:

- `M` contains the number of MCMC draws
- `par` contains the MCMC draws $(\boldsymbol{\theta}_1^{(m)}, \ldots, \boldsymbol{\theta}_K^{(m)})$ for each parameter in `marmix.par`. In general, the field `par` has the same structure as the corresponding field in definition of the Markov switching model.
- `indicmod.xi` contains the MCMC draws $\boldsymbol{\xi}^{(m)}$ for the transition which are stored in a `M x K x K` numerical array.
- `acc.xi` report acceptance rates, if a Metropolis Hastings algorithm has been used to sample the transition matrices under a starting distribution being equal to the ergodic distribution.
- `perm` is a logical variable, which is `true` if the MCMC draws are based on random permutation sampling. Otherwise `perm` is false.
- `hyper` is added under a hierarchical prior and contains the MCMC draws for the random hyperparameter.
- `log` stores the logarithm of various function evaluated at the MCMC draws. The field `log` is a structure array containing the following fields, each of them being a `M x 1` numerical array:
  - `mixlik` stores the log of the Markov mixture likelihood, $\log p(\mathbf{y}|\boldsymbol{\vartheta}^{(m)})$, for each MCMC draw $\boldsymbol{\vartheta}^{(m)}$.
  - `mixprior` stores the log of the prior, $\log p(\boldsymbol{\vartheta}^{(m)})$, for each MCMC draw $\boldsymbol{\vartheta}^{(m)}$.
  - `cdpost` stores the log of the (non-normalized) complete data posterior, $\log p(\boldsymbol{\vartheta}^{(m)}, \mathbf{S}^{(m)}|\mathbf{y})$, which is equal to

    $$p(\boldsymbol{\vartheta}^{(m)}, \mathbf{S}^{(m)}|\mathbf{y}) = p(\boldsymbol{\vartheta}^{(m)}|\mathbf{y})p(\mathbf{S}^{(m)}|\boldsymbol{\vartheta}^{(m)}, \mathbf{y}) \propto p(\mathbf{y}|\boldsymbol{\vartheta}^{(m)})p(\boldsymbol{\vartheta}^{(m)})p(\mathbf{S}^{(m)}|\boldsymbol{\vartheta}^{(m)}, \mathbf{y})$$

    for each MCMC draw $\boldsymbol{\vartheta}^{(m)}$ and $\mathbf{S}^{(m)}$.
  - The field `t0`, if the model contains an autoregressive part and computation of the likelihood does not start with $t = t_0 = 1$, but with the $t_0 > 1$.
- `entropy` is a `Mx1` numerical array storing the entropy $\mathrm{EN}(\boldsymbol{\vartheta}^{(m)}|\mathbf{y})$, see (9.5), for each MCMC draw.

- `S` is added, if paths of the hidden Markov chain are stored (see `mcmc.storeS` above). The field contains the last `L=mcmc.storeS` MCMC draws of $\mathbf{S}^{(m)}$ (without the state at 0), stored as a `L x N` numerical array, where `N` are the number of observations.
- `ST` contains all MCMC draws of the last state $S_T^{(m)}$, which are stored as a `M x 1` numerical array. These draws are stored independently of `mcmc.storeS`, because they are needed for forecasting purposes.
- `post` is added, if posterior moments are stored (see `mcmc.storepost` above). This is a structure array with the fields `par` and `indicmod.xi`:
  - `indicmod.xi` is a `M x K x K` numerical array containing the moments $e_1(\mathbf{S}), \ldots, e_K(\mathbf{S})$ of the the posterior Dirichlet distribution $\mathcal{D}(e_1(\mathbf{S}), \ldots, e_K(\mathbf{S}))$ used for simulating the transition matrix $\boldsymbol{\xi}^{(m)}$.
  - `par` contains certain moments of the complete data posterior distributions used for simulating the model parameters. These are used for computing marginal likelihoods in Subsection 9.8.6.

The following field is added if the ergodic distribution is chosen as initial distribution:

- `acc.xi` contains the acceptance rate when sampling $\boldsymbol{\xi}$ by the help of the Metropolis-Hastings algorithm, see Frühwirth-Schnatter (2006, p.341).

If $K$ is equal to 1, then a single member from the distribution family `dist` is fitted and redundant fields like `indicmod.xi`, `S`, `post.indicmod.xi`, and `perm` are not added to `mcmcout`.

## 9.8 Bayesian Inference Using the Posterior Draws

Frühwirth-Schnatter (2006, Subsection 11.5.8) discusses in detail, how posterior draws could be used for Bayesian inference for Markov mixture models.

### 9.8.1 Plotting MCMC

The function `mcmcplot` explained in Subsection 4.5.1 could be used to plot and monitor the MCMC output. To visualize the posterior density $p(\boldsymbol{\vartheta}|\mathbf{y})$ of a Markov switching model, draws from the posterior density $p(\boldsymbol{\vartheta}|\mathbf{y})$ are used as a sampling representation of the posterior distribution, which is then visualized as in Subsection 4.5.1 by calling the function `mcmcsamrep`. This function produces point process representation of the posterior draws. For Markov mixtures with univariate component parameter $\theta$, $\theta_k^{(m)}$ is plotted against the draws the persistence probability $\xi_{kk}^{(m)}$. For Markov mixtures with bivariate component parameter $\boldsymbol{\theta} = (\theta_1, \theta_2)$, $\theta_{1,k}^{(m)}$ is plotted against $\theta_{2,k}^{(m)}$. For Markov mixtures with multivariate components parameters $\boldsymbol{\theta}$, special point process representations are generated for different models.

These scatter plots are closely related to the point process representation of the underlying marginal mixture distribution discussed in Subsection 9.2.4. The MCMC draws will scatter around the points corresponding to the true point process representation, with the spread of the clouds representing the uncertainty of estimating the points. The number of simulations clusters visible in these MCMC draws are helpful for Markov mixtures with an unknown number of states. If the Markov mixture distribution is not overfitting, then $K$ simulations clusters should be present in these figures. If the Markov mixture distribution is overfitting, then fewer simulations clusters are present, and a Markov mixture with less states should be fitted to the time series.

### 9.8.2 Estimating the State Specific Parameters and the Transition Matrix

To perform parameter estimation after MCMC sampling, call the function `mcmcestimate` introduced in Subsection 4.5.2 with the structure array, say `mcmcout`, containing the MCMC output as input argument. If `mcmcestimate` is called with two output arguments,

```
[est,mcmcout]=mcmcestimate(mcmcout);
```

then the estimators and the identified MCMC output will be added to the MCMC output `mcmcout`. The structural array `est` has the same structure as discussed in Subsection 4.5.2. For each estimation method, the estimator of the transition matrix $\boldsymbol{\xi}$ is stored in the field `indicmod.xi`, e.g.

- `est.pm.indicmod.xi` – (approximate) posterior mode estimator,
- `est.ident.indicmod.xi` – ergodic average after identification.

For each estimation method, the estimators of the parameters are stored in the field `par`, which has the same structure as for the estimated model.

### 9.8.3 Bayesian Time Series Segmentation and State Probabilities

It is possible to cluster the time series observations into the different states, see Frühwirth-Schnatter (2006, Subsection 11.5.8). To carry out time series segmentation call the function `mcmcclust` introduced in Subsection 4.5.3:

```
clust=mcmcclust(timeseries,mcmcout),
```

If initial observations $y_1, \ldots, y_{t_0-1}$ have been used to the define the design matrix of the MSAR model, then clustering starts with $t_0$. In this case, the field

- `t0` containing the index of the first classification

is added to the structural array `clust`.

To visualize time series segmentation and the estimated state probabilities $\Pr(S_t = k|\mathbf{y})$, call the function `mcmcclustplot` in the following way

```
[nfig=]mcmcclustplot(timeseries,clust[,nfig]);
```

where `clust` is the output from calling the function `mcmcclust`. Plotting starts with the input figure number `nfig`, or with figure one, if the input argument `nfig` is missing. The output argument `nfig` reports the number of the last figure and may be omitted.

This function produces a plot of the estimated state probabilities $\Pr(S_t = k|\mathbf{y}), t \geq t_0$ and a plot showing time series segmentation for the different estimators of $\mathbf{S}$ stored in `clust`.

### 9.8.4 Diagnosing Markov Mixture Models

As discussed in Frühwirth-Schnatter (2006, Subsection 11.6.1), diagnosing the goodness-of-fit for Markov switching models may be based on studying the posterior distribution of certain moments implied by the Markov mixture and studying the predictive posterior distribution of certain statistics.

The function `mcmcdiag` introduced in Subsection 5.2 produces various diagnostic plot for the comparison of more than one model. The function may be called simultaneously for more than one MCMC output, in order to compare the different models:

```
[nfig=]mcmcdiag(data,mcmcout1,...,mcmcoutK[,nfig]);
```

where `data` are the data, and `mcmcout1,...,mcmcoutK` is an arbitrary number of structure arrays containing the MCMC output of a certain model. Plotting starts with the input figure number `nfig`, or with figure one, if the input argument `nfig` is missing. The output argument `nfig` reports the number of the last figure and may be omitted.

A particularly useful statistic for assessing goodness-of-fit for a Markov switching model is the predictive posterior distribution of the implied autocorrelation function $\rho_{Y_t}(h|\boldsymbol{\vartheta})$ in comparison to the observed autocorrelation function as well as the autocorrelation function of the squared process. Several figures compare the posterior distribution of moments of the marginal distribution for the different models.

The remaining plots are standard diagnostic predictive checks, based on the sample moments included in the function `datamoments`, see Subsection 3.2.2 for more details. This includes also the empirical autocorrelation function of the time series and the squared time series. These predictive checks are based on drawing 200 predictive samples of size $T$.

### 9.8.5 Model Choice Criteria

Common model choice criteria are AIC, BIC, and different classification-based information criteria (Frühwirth-Schnatter, 2006, Section 4.4.2,7.1.4) which are minimized for the optimal model among a set of potential models. To compute these criteria from the MCMC output, call the function `mcmcic` introduced in Subsection 5.4.

### 9.8.6 Marginal Likelihoods for Markov Switching Models

To compute the marginal likelihood of a Markov switching model (Frühwirth-Schnatter, 2006, Subsection 11.6.3), call the function

```
marlik = mcmcbf(data,mcmcout)
```

introduced in Subsection 5.3.

#### Marginal Likelihoods for Selecting the AR Order

MCMC estimation and time series segmentation is carried out conditional on the minimum number of observations needed to define the model, thus the number of dependent observations is different for models differing in the AR order. When comparing models of different AR orders through marginal likelihoods one has to make sure that the number of dependent observations is the same. If the maximum AR order of all models to be compared is equal to $p_{\max}$, then all marginal likelihoods have to be computed as $p(y_{p_{\max}+1}, \ldots, y_T | y_1, \ldots, y_{p_{\max}})$. Thus the first data point has to be set to $p_{\max} + 1$. This is achieved by adding the field

- `t0`, taking the value $p_{\max} + 1$

to the structure array specifying the data, say `timeseries`, before calling `mcmcbf`.

## 9.9 Prediction of Time Series Based on Markov Switching Models

Bayesian forecasting of future observations $\mathbf{y}_f = (y_{T+1}, \ldots, y_{T+H})$ of a time series $\mathbf{y} = (y_1, \ldots, y_T)$ is based on the predictive density $p(y_{T+1}, \ldots, y_{T+H}|\mathbf{y})$. Algorithm 12.1 in Frühwirth-Schnatter (2006, Section 12.4) shows how to sample $M$ future sequences $\mathbf{y}_f^{(m)} = (y_{T+1}^{(m)}, \ldots, y_{T+H}^{(m)}), m = 1, \ldots, M$ of length $H$ from the posterior predictive distribution $p(y_{T+1}, \ldots, y_{T+H}|\mathbf{y})$. This algorithm is based on sampling $\mathbf{y}_f^{(m)} = (y_{T+1}^{(m)}, \ldots, y_{T+H}^{(m)})$ from the conditional density $p(y_{T+1}, \ldots, y_{T+H}|\boldsymbol{\vartheta}^{(m)}, S_T^{(m)}, \mathbf{y})$ for each MCMC draw $\boldsymbol{\vartheta}^{(m)}$ and $S_T^{(m)}$ in the structure array `mcmcout`. To this aim, the function `simulate` is called for each MCMC draw, with the "starting value" of the hidden Markov chain being fixed at $S_T^{(m)}$, see Subsection 9.2.3.

### 9.9.1 Prediction of a Basic Markov Mixture

If the conditional density $p(y_t|\boldsymbol{\vartheta}, S, y^{t-1})$ is independent of the past values $y^{t-1}$, like for the basic Markov mixture model considered in Section 9.2,

then $p(y_{T+1},\ldots,y_{T+H}|\boldsymbol{\vartheta}^{(m)},S_T^{(m)})$ is independent of observed time series and simulation based forecasting may be implemented by call the function `mcmcpredsam`, introduced in Subsection 4.5.4 for finite mixture models in the same way:

```
pred=mcmcpredsam(mcmcout,H);
```

where `mcmcout` contains the MCMC draws. For a univariate time series `pred` is a `M x H` numerical array, containing the future sequences, i.e. `pred(m,:)` contains the $m$th future sequences $\mathbf{y}_f^{(m)}=(y_{T+1}^{(m)},\ldots,y_{T+H}^{(m)})$.

### 9.9.2 Prediction of an MSAR Model

If the conditional density $p(y_t|\boldsymbol{\vartheta},S,y^{t-1})$ depends of the past values $y^{t-1}$, like for the MSAR model considered in Section 9.4, then the predictive density $p(y_{T+1},\ldots,y_{T+H}|\boldsymbol{\vartheta}^{(m)},S_T^{(m)},y_{T-p+1},\ldots,y_T)$ depends on the observed time series which has to be added as a calling argument:

```
pred=mcmcpredsam(mcmcout,H,timeseries);
```

To this aim, the function `simulate` is called for each MCMC draw, with the "starting value" for the past observations, which are stored in the field `ystart` before calling `simulate`, being fixed at $y_{T-p+1},\ldots,y_T$, see Subsection 9.4.3.

### 9.9.3 Prediction of Dynamic Regression Models

In the Markov switching regression model considered in Section 9.3, as well as in the Markov switching dynamic regression model considered in Section 9.5, time-varying regressors are present. In this case the predictive density $p(y_{T+1},\ldots,y_{T+H}|\boldsymbol{\vartheta}^{(m)},S_T^{(m)},y_{T-p+1},\ldots,y_T)$ depends on the future values of this regressor. These values have to added to the field `X`, before calling simulate:

```
future = ... (array of size d x H)
timeseries.X=[timeseries.X future];
pred=mcmcpredsam(mcmcout,H,timeseries).
```

## References


Aitkin, M. (1996). A general maximum likelihood analysis of overdispersion in generalized linear models. *Statistics and Computing 6*, 251–262.

Bensmail, H., G. Celeux, A. E. Raftery, and C. P. Robert (1997). Inference in model-based cluster analysis. *Statistics and Computing 7*, 1–10.

Chib, S. (1995). Marginal likelihood from the Gibbs output. *Journal of the American Statistical Association 90*, 1313–1321.

Chib, S. (1996). Calculating posterior distributions and modal estimates in Markov mixture models. *Journal of Econometrics 75*, 79–97.

Escobar, M. D. and M. West (1998). Computing nonparametric hierarchical models. In D. Dey, P. Müller, and D. Sinha (Eds.), *Practical Nonparametric and Semiparametric Bayesian Statistics*, Number 133 in Lecture Notes in Statistics, pp. 1–22. Berlin: Springer.

Fernández, C. and M. F. J. Steel (1999). Multivariate student-$t$ regression models: Pitfalls and inference. *Biometrika 86*, 153–167.

Fonseca, T. C. O., M. A. R. Ferreira, and H. S. Migon (2008). Objective Bayesian analysis for the Student-$t$ regression model. *Biometrika 95*, 325–333.

Frühwirth-Schnatter, S. (2001). Markov chain Monte Carlo estimation of classical and dynamic switching and mixture models. *Journal of the American Statistical Association 96*, 194–209.

Frühwirth-Schnatter, S. (2004). Estimating marginal likelihoods for mixture and Markov switching models using bridge sampling techniques. *The Econometrics Journal 7*, 143–167.

Frühwirth-Schnatter, S. (2006). *Finite Mixture and Markov Switching Models*. Springer Series in Statistics. New York/Berlin/Heidelberg: Springer.

Frühwirth-Schnatter, S., R. Frühwirth, L. Held, and H. Rue (2009). Improved auxiliary mixture sampling for hierarchical models of non-Gaussian data. *Statistics and Computing 19*, forthcoming.

Geweke, J. (1993). Bayesian treatment of the independent Student-$t$ linear model. *Journal of Applied Econometrics 8(Supplement)*, 19–40.

Hamilton, J. D. (1989). A new approach to the economic analysis of nonstationary time series and the business cycle. *Econometrica 57*, 357–384.

Hurn, M., A. Justel, and C. P. Robert (2003). Estimating mixtures of regressions. *Journal of Computational and Graphical Statistics 12*, 55–79.

Leroux, B. G. and M. L. Puterman (1992). Maximum-penalized-likelihood estimation for independent and Markov-dependent mixture models. *Biometrics 48*, 545–558.

Lin, T. I., J. C. Lee, and W. J. Hsieh (2007). Robust mixture modeling using the skew $t$-distribution. *Statistics and Computing 17*, 81–92.

McCulloch, R. E. and R. S. Tsay (1994). Statistical analysis of economic time series via Markov switching models. *Journal of Time Series Analysis 15*, 523–539.

Pauler, D. K., M. D. Escobar, J. A. Sweeney, and J. Greenhouse (1996). Mixture models for eye-tracking data: A case study. *Statistics in Medicine 15*, 1365–1376.

Richardson, S. and P. J. Green (1997). On Bayesian analysis of mixtures with an unknown number of components. *Journal of the Royal Statistical Society, Ser. B 59*, 731–792.

Stephens, M. (1997). *Bayesian Methods for Mixtures of Normal Distributions*. Ph. D. thesis, University of Oxford. CHECK.

Titterington, D. M., A. F. M. Smith, and U. E. Makov (1985). *Statistical Analysis of Finite Mixture Distributions*. Wiley Series in Probability and Statistics. New York: Wiley.

Viallefont, V., S. Richardson, and P. J. Green (2002). Bayesian analysis of Poisson mixtures. *Journal of Nonparametric Statistics 14*, 181–202.

Wagner, H. (2007). Bayesian analysis of mixtures of exponentials. *Journal of Applied Mathematics, Statistics and Informatics 3*, 165–183.