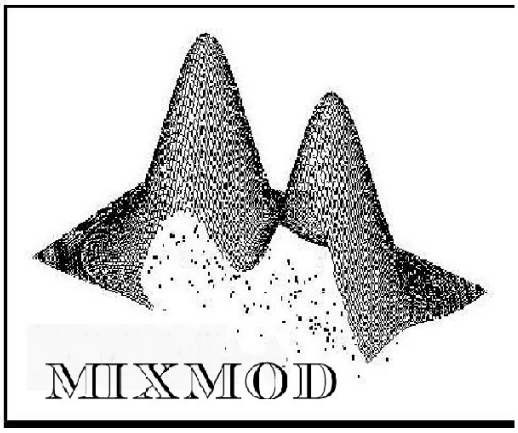
MIXMOD User's Guide

MIXture MODelling Software



High Performance Model-Based Cluster and Discriminant Analysis

`http://www-math.univ-fcomte.fr/mixmod/index.php`

Christophe Biernacki[1], Gilles Celeux[2], Jean-François Si Abdallah[2], Gérard Govaert[3], Florent Langrognet[4]

[1] *UMR 8524, Cnrs & Université de Lille 1*
[2] *INRIA Futurs*
[3] *UMR 6599, Cnrs & Université de Technologie de Compiègne*
[4] *UMR 6623, Cnrs & Université de Franche-Comté*

Contents

Chapter 1

Introduction

MIXMOD software is a tool for fitting a mixture model (see equation 1.1) of multivariate gaussian or multinomial components to a given data set with either a clustering, a density estimation or a discriminant analysis point of view.

This documentation is intended to help you to install and start MIXMOD. The use of the package is illustrated by many examples that can be run with MIXMOD.

How this user guide is organized
The *Introduction* chapter provides an overview of the software, areas of application and available user interfaces. The chapter *Getting started* provides basic information on systems requirements, installation and how to run MIXMOD. The *Using* MIXMOD chapter describes the Scilab and Matlab interfaces, toolkit functions and the command line expert use. Each section contains some examples which explain software features.

1.1 General purpose

The general purpose of the package is to discover, or explain, group structures in multivariate data sets with unknown (cluster analysis or clustering) or known class (discriminant analysis or classification). It is an exploratory data analysis tool for solving clustering and classification problems. But it can also be regarded as a semi-parametric tool to estimate densities with Gaussian mixture distributions and multinomial distributions.

Mathematically, mixture probability density function (pdf) f is a weighted sum of K components densities

$$f(\mathbf{x}_i|\theta) = \sum_{k=1}^{K} p_k h(\mathbf{x}_i|\lambda_k) \tag{1.1}$$

where $h(.|\lambda_k)$ denotes a d-dimensional distribution parametrized by λ_k. The parameters are the mixing proportions p_k and the component of the distribution λ_k.

In the Gaussian case, h is the density of a Gaussian distribution with mean μ_k and variance matrix Σ_k, and thus $\lambda_k = (\mu_k, \Sigma_k)$.

In the qualitative case, h is a multinomial distribution and $\lambda_k = (a_k, \epsilon_k)$ is the parameter of the distribution (see statistical documentation).

1.2 What is MIXMOD ?

The name MIXMOD stands for MIXture MODelling. Estimation of the mixture parameters is performed either through maximum likelihood via the EM (*Expectation Maximization*, Dempster et

al. 1977), the SEM (*Stochastic EM*, Celeux and Diebolt 1985) algorithm or through classification maximum likelihood via the CEM algorithm (*Clustering EM* , Celeux and Govaert 1992). These three algorithms can be chained to obtain original fitting strategies (e.g. CEM then EM with results of CEM) to use advantages of each of them in the estimation process. As mixture problems usually have multiple relative maxima, the program will produce different results, depending on the initial estimates supplied by the user. If the user does not input his own initial estimates, some initial estimates procedures are proposed (random centers for instance).

It is possible to constrain some input parameters. For example, dispersions can be equal between classes, etc.

In the Gaussian case, twenty-two models are implemented. Among them, fourteen models, based on the eigenvalue decomposition, are most generally used. They depend on constraints on the variance matrix such as same variance matrix between clusters, spherical variance matrix ... and they are suitable for data sets in any dimension.

The eight remaining gaussian models have to be used when the dimension of the data set is high and only in a discriminant analysis situation.

In the qualitative case, five multinomial models are available. They are based on a reparametrization of the multinomial probabilities.

In both cases, the models and the number of clusters can be chosen by different criteria BIC (Bayesian Information Criterion), ICL (Integrated Completed Likelihood, a classification version of BIC), NEC (Entropy Criterion), Cross-Validation (CV) or Double Cross-Validation (DCV).

MIXMOD is an object-oriented package built around C++ language (it consists of a collection of C++ classes). It preserves the full capability, performance, accuracy and low memory requirements of C, but takes advantage of the C++ object-oriented programming environment. This C++ version was developed jointly by Inria Futurs (Projects IS2 and SELECT), Laboratory of Mathematics of Besançon (UMR 6623 CNRS - Université de Franche-Comté), Heudiasyc Laboratory (UMR 6599 CNRS - Université de Technologies de Compiègne) and Paul Painlevé Laboratory (UMR 8524 CNRS - Université de Lille 1).
MIXMOD is interfaced with Scilab and Matlab; it is the easiest way to discover MIXMOD.

1.3 Example

Modelling data with a mixture distribution is suitable when it comes from a heterogeneous population. To illustrate this, consider the following example. The acidity data set concerns the distribution of enzymatic activity in the blood, for an enzyme involved in the metabolism of carcinogenic substances, among a group of 245 unrelated individuals. The interest here is to identify subgroups of slow or fast metabolism as a marker of genetic polymorphism in the general population. The purpose is to discover a group structure in the data with unknown class (cluster analysis). The number of classes can be chosen automatically by MIXMOD. In this example MIXMOD selects a two-component normal mixture.
The estimated parameters are given in Table 1.1 and Figure 1.1 represents the histogram and the two-component mixture model obtained by MIXMOD.

Component k	p_k	μ_k	σ_k^2
1	0.4055	6.2446	0.2739
2	0.5945	4.3277	0.1370

Table 1.1: A Two-component normal mixture solution for the acidity data set.

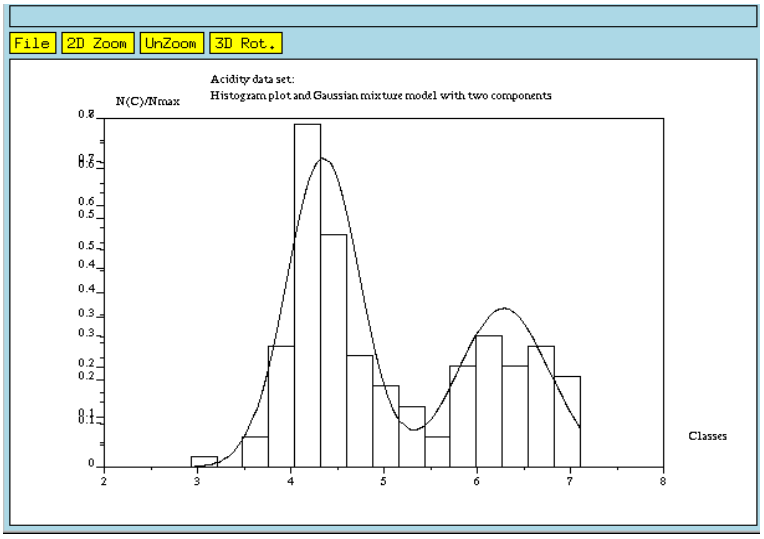


Figure 1.1: Plot of fitted two-component normal mixture density for acidity data set.

1.4 Areas of application

The package is intended to scientists and engineers who need to solve clustering, density estimation and classification problems. In academic environments, it can be regarded as an instructional tool for introductory and advanced courses in discriminant and cluster analysis. For research scientists, an expert mode provides full capabilities for simulations studies. In an industrial context, MIXMOD is a powerful tool for research in statistical pattern recognition.

1.5 User interface

Written in C++, MIXMOD is easily interfaced with widely mathematical software Scilab and Matlab. The MIXMOD Graphical User Interface (GUI) is an important key element for interactive statistical analysis, the user is guided step by step.

1.6 Documentation

The documentation around MIXMOD is composed of a *User's guide*, a *Statistical Manual*, a *Software Documentation* and a *Quick start Manual*. They are available on MIXMOD web site and in *<mixmodDir>/DOC* directory and are distributed as Postscript and pdf (Acrobat Reader) files.

1.7 Conditions of use

MIXMOD is publicly available under the GPL license. You can redistribute it and/or modify it under the terms of the GPL license (www.gnu.org/copyleft/gpl.htlm).

Please understand that there may still be bugs and errors. Use is at your own risk. We take no responsibility for any errors or omissions in this package or for any misfortune that may befall you or others as a result of its use.

Please report bugs at mixmod@univ-fcomte.fr

1.8 Other softwares

Several codes that are available characterize multivariate data sets as mixtures of Gaussian populations. They include (non exhaustive list)

- EMMIX by G. McLachlan
  `http://www.maths.uq.edu.au/$\sim$gjm/emmix/emmix.html`
- MCLUST by C. Fraley and A. Raftery
  `http://www.stat.washington.edu/fraley/mclust/soft.shtml`
- AutoClass by P. Cheeseman
  `http://ic-www.arc.nasa.gov/ic/projects/bayes-group/autoclass/`
- Snob by D. Dowe.
  `http://www.csse.monash.edu.au/$\sim$dld/Snob.html`
- NORMIX by John H. Wolfe.
  `http://www.alumni.caltech.edu/$\sim$wolfe/normix.htm`

Chapter 2

Getting started

2.1 System requirements

MIXMOD can be used on many systems (unix, linux and windows).

2.2 Download and Installation instructions

Platform	Version	Procedure
Linux	binary	1) download mixmod_2_1_1_linux_bin.tgz 2) put it where you want to install MIXMOD 3) extract mixmod_2_1_1_linux_bin.tgz
	source	1) download mixmod_2_1_1_linux_src.tgz 2) put it where you want to install MIXMOD 3) extract mixmod_2_1_1_linux_src.tgz 4) launch successively in < *mixmodDir* > *cmake .*, *make*, *make install*
Windows	mixmod_2_1_1.exe	1) download mixmod_2_1_1.exe 2) launch MIXMOD setup and choose < *mixmodDir* >

2.3 Using MIXMOD

Scilab

Run Scilab and execute the *initMixmod.sci* file in directory MIXMOD/ by typing the following command:

```
exec('initMixmod.sci');
```

You will be asked to enter MIXMOD directory location, this will load *MIXMOD* functions each time you run Scilab.
This procedure has to be done only once. Each time you install a new version of MIXMOD, you will have to launch *initMixmod.sci*. To know which version you are working with, you can type 'mixmodDir' in Scilab consol.

Matlab

MIXMOD runs with Matlab 7 or above. Before using *MIXMOD*, you have to set *mixmod path* in Matlab by selecting *Set Path* in *File* menu of Matlab window, clic on *Add Folder* then select *MIXMOD* and *MIXMOD/UTIL/MATLAB*. Either you have administrator rights and thus you can save the new path or you will have to set *mixmod path* each time you want to run *mixmod*.

Chapter 3

Using MIXMOD

In this chapter, examples with prompt `->` are written in Scilab environment and examples with prompt `>>` are written in Matlab environment. Examples with prompt `->(>>)` are available in both environments.
In files examples, note that all sequences written between `[ ]` are comments for users.

3.1 Data representation

3.1.1 Individuals representation

MIXMOD is concerned with data sets where experimental unit individuals are described with several variables. Individuals are represented in a standard way a row vector. Then data sets are represented by a matrix with n rows and d columns, rows representing individuals, and columns representing variables.
Let's consider the *esteve.dat* individual file, in the directory MIXMOD/DATA, which describes 77 individuals and 7 variables in a **quantitative situation**

```
0.8277   0.3626   0.4243   0.0269   0.0523   0.0000   2.7021
0.7713   0.4399   0.4586   0.0355   0.0000   0.0015   4.0874
0.6407   0.4841   0.5302   0.0000   0.2723   0.0000   2.7531
0.9384   0.2396   0.2436   0.0008   0.0516   0.0000   2.6013

...
```

Let's consider the *b_conf.dat* individual file, in the directory MIXMOD/DATA, which describes 30 individuals and 8 variables in a **qualitative situation**

```
1        1        5        1        1        1        1        2
1        5        1        1        1        4        2        2
2        2        5        1        1        3        1        1
1        5        2        1        1        3        2        1

...
```

Notice that the possible values of qualitative data depend on the number of modalities of the variable. Only integers are authorized and values such as *yes/no* are not allowed.

3.1.2 Weight representation

Weights are stored in a vector of real or integer numbers, with n (number of individuals) rows. Weight is an option of MIXMOD and for example it can represent the repetition of individuals. Let's consider the weight file *geyser.wgt* in the directory MIXMOD/DATA/. It gives the information of all individuals of *geyser.dat* repeated 2 times.

3.1.3 Parameter representation

This representation is used in the case of USER initialization type.

- Gaussian

  Let's consider the parameter file *parameter.init* with 2 clusters, in the directory MIXMOD/DATA/. It gives the values of the unknown mixture model parameters in the quantitative case

```
0.367647                [Initial proportion of component 1]
2.094330   54.750000    [Initial mean of component 1]
17.280889   0.000000    [Initial variance matrix of component 1]
0.000000   17.280889


0.632353                [Initial proportion of component 2]
4.297930   80.284884    [Initial mean of component 2]
15.830206   0.000000    [Initial variance matrix of component 2]
0.000000   15.830206
```

- Multinomial

  Let's consider the parameter file *parameter_qualitatif.init* with 2 clusters in the directory MIXMOD/DATA/. It gives the values of the unknown mixture model parameters in the qualitative case (with modalities [2;3;4;5])

```
0.2                     [Initial proportion of component 1]
1  2  3  4              [Initial mean of component 1]
0.1     0.1             [Initial dispersion array of component 1]
0.1     0.2     0.1
0.1     0.1     0.3     0.1
0.1     0.1     0.1     0.4   0.1


0.8                     [Initial proportion of component 2]
2  3  4  5              [Initial mean of component 2]
0.5     0.5             [Initial dispersion array of component 2]
0.25    0.25    0.5
0.1667  0.1667  0.1667  0.5
0.125   0.125   0.125   0.125   0.5
```

- Gaussian High Dimensional

  Let's consider the parameter file *parameter_HD.init* with 2 clusters in the directory MIXMOD/DATA. It gives the values of the unknown mixture parameters in the gaussian high dimensional (HD) case

```
0.75                                            [Initial proportion of component 1]
14.842      11.718     32.014    36.81   13.35  [Initial mean of component 1]
3                                               [Initial subDimension of component 1]
144.744604  0.214614   0.101925                 [Initial parameter Akj of component 1]
0.063887                                        [Initial parameter Bk of component 1]
-0.262423    0.355093   0.004478                [Initial parameter Bk oforientation array of component 1]
-0.170051   -0.888586  -0.207320
-0.601104    0.057918   0.532327
-0.687179   -0.071419  -0.105660
-0.262061    0.275444  -0.813918


0.25                                            [Initial proportion of component 2]
13.27       12.138     28.102    32.624  11.816 [Initial mean of component 2]
3                                               [Initial subDimension of component 2]
99.333875   0.155693   0.138530                 [Initial parameter Akj of component 2]
0.049261                                        [Initial parameter Bk of component 2]
-0.259855    0.127732  -0.473221                [Initial parameter Bk oforientation array of component 2]
-0.239538    0.555917   0.750006
-0.587639   -0.078075  -0.049268
-0.675526    0.144512  -0.205855
-0.271003   -0.804774   0.410791
```

3.1.4 Partition representation

A partition gives a classification of the individuals they are affected to a mixture component.
It is a matrix (of 0 and 1) with n rows and k columns, each row corresponding to an individual and each column indicating the group membership (0 if the individual does not belong to the group associated to this column and 1 otherwise).
Some individuals can have no group assignment they are represented by a row of 0.
Let's consider the full partition file *geyser3clusters.part* in the directory MIXMOD/DATA/. This file gives a full classification of *geyser.dat* for 3 mixture components.

This representation is used in the case of USER_PARTITION initialization type and/or with known partition.

```
0   1   0    [individual 1 in component 2]
0   0   1    [individual 2 in component 3]
0   1   0    [individual 3 in component 2]
0   0   1    [individual 4 in component 3]
1   0   0    [individual 5 in component 1]

...

0   1   0    [individual 268 in component 2]
0   0   1    [individual 269 in component 3]
1   0   0    [individual 270 in component 1]
0   0   1    [individual 271 in component 3]
1   0   0    [individual 272 in component 1]
```

3.2 MIXMOD inputs

This section is aimed to define all the inputs available in MIXMOD independently of how MIXMOD is used (scilab, matlab, or in command line).

3.2.1 MIXMOD required inputs

MIXMOD function needs two required inputs

1. *data* a matrix $[n, d]$ of individuals (n number of samples, d number of variables),
2. *nbCluster* a vector of integers representing the number of clusters (several clusters can be in competition),

- qualitative case

  tabModality a vector representing the modalities on each variable.

Optional inputs can be added to the required ones to precise the execution of MIXMOD.

3.2.2 MIXMOD optional inputs

Criterion This option permits to select the criterion giving the best configuration of an execution (model, number of cluster and strategy)

- BIC Bayesian Information Criterion (cf. Section 4.3.1 of Statistical Documentation) ;
- ICL Integrated Completed Likelihood (cf. Section 4.3.2 of Statistical Documentation) ;
- NEC Entropy Criterion (cf. Section 4.3.3 of Statistical Documentation) ;
- CV Cross-Validation (cf. Section 4.3.4 of Statistical Documentation) ;
- DCV Double Cross-Validation (cf. Section 4.3.5 of Statistical Documentation).

Default value is 'BIC'.

Model Specifying a model different of the default one is possible when informations on the data are known (for example there is the same number of individuals in each class).
Default value is 'Binary_pk_Ekjh' for qualitative data (see Table 3.2) or 'Gaussian_pk_Lk_C' for quantitative data (see Table 3.1 and Table 3.3).
With gaussian HD models, you have to give subDimensionFree and/or subDimensionEqual parameters.

Table 3.1: The 28 Gaussian Models (Covariance Structure).

Model	Categories
Gaussian_p_L_I	Spherical
Gaussian_p_L_k_I	
Gaussian_p_L_B	Diagonal
Gaussian_p_L_k_B	
Gaussian_p_L_B_k	
Gaussian_p_L_k_B_k	
Gaussian_p_L_C	General
Gaussian_p_L_k_C	
Gaussian_p_L_D_A_k_D	
Gaussian_p_L_k_D_A_k_D	
Gaussian_p_L_D_k_A_D_k	
Gaussian_p_L_k_D_k_A_D_k	
Gaussian_p_L_C_k	
Gaussian_p_L_k_C_k	
Gaussian_p_k_L_I	Spherical
Gaussian_p_k_L_k_I	
Gaussian_p_k_L_B	Diagonal
Gaussian_p_k_L_k_B	
Gaussian_p_k_L_B_k	
Gaussian_p_k_L_k_B_k	
Gaussian_p_k_L_C	General
Gaussian_p_k_L_k_C	
Gaussian_p_k_L_D_A_k_D	
Gaussian_p_k_L_k_D_A_k_D	
Gaussian_p_k_L_D_k_A_D_k	
Gaussian_p_k_L_k_D_k_A_D_k	
Gaussian_p_k_L_C_k	
Gaussian_p_k_L_k_C_k	

Table 3.2: The 10 Binary Models.

Model
Binary_p_E
Binary_p_E_j
Binary_p_E_k
Binary_p_E_{kj}
Binary_p_E_{kjh}
Binary_pk_E
Binary_pk_E_j
Binary_pk_E_k
Binary_pk_E_{kj}
Binary_pk_E_{kjh}

Table 3.3: The 16 High Dimensional (HD) Models.

Model
Gaussian_HD_p_A_{kj}_B_k_Q_k_D_k
Gaussian_HD_p_A_k_B_k_Q_k_D_k
Gaussian_HD_p_A_{kj}_B_k_Q_k_D
Gaussian_HD_p_A_{kj}_B_Q_k_D
Gaussian_HD_p_A_k_B_k_Q_k_D
Gaussian_HD_p_A_k_B_Q_k_D
Gaussian_HD_p_A_j_B_k_Q_k_D
Gaussian_HD_p_A_j_B_Q_k_D
Gaussian_HD_p_k_A_{kj}_B_k_Q_k_D_k
Gaussian_HD_p_k_A_k_B_k_Q_k_D_k
Gaussian_HD_p_k_A_{kj}_B_k_Q_k_D
Gaussian_HD_p_k_A_{kj}_B_Q_k_D
Gaussian_HD_p_k_A_k_B_k_Q_k_D
Gaussian_HD_p_k_A_k_B_Q_k_D
Gaussian_HD_p_k_A_j_B_k_Q_k_D
Gaussian_HD_p_k_A_j_B_Q_k_D

Weight This option is to be used when weight is associated to the data.
Default value is 1.

Partition This option is to be used when a partition of the data is already known.
Default value is no knownPartition.

Strategy This option permits to define different ways to execute the algorithms available in MIXMOD (cf. section 3 of Statistical Documentation) .

1. **initialization** defining the initialization of the strategy. There are different ways to initialize an algorithm
   - RANDOM Initialization from a random position is a standard way to initialize an algorithm. This random initial position is obtained by choosing at random centers in the data set. This simple strategy is repeated 5 times (the user can choose the number of times) from different random positions and the position that maximises the likelihood is selected.
   - USER_PARTITION This option initializes the strategy from a specified classification (full or partial) of the observations. This option provides the possibility to use MIXMOD for Discriminant Analysis and in this case, partition must be full.
   - USER This option starts the strategy with specified initial values of the unknown mixture model parameters, i.e. the mixing proportions and the parameters of the distribution.
   - SMALL_EM A maximum of 50 iterations of the EM algorithm according to the process n_i numbers of iterations of EM are done (with random initialization) until the SMALL_EM stop criterion value has been reached (cf. Statistical Documentation Equation (14)). This action is repeated until the sum of n_i reaches 50 iterations (or if in one action 50 iterations are reached before the stop criterion value).
     It appears that repeating runs of EM is generally profitable since using a single run of EM can often lead to suboptimal solutions.
   - CEM_INIT 10 repetitions of 50 iterations of the CEM algorithm are done. One advantage of initializing an algorithm with CEM lies in the fact that CEM converges generally

in a small number of iterations. Thus, without consuming a large amount of CPU times, several runs of CEM are performed. Then EM is run with the best solution among the 10 repetitions.

– SEM_MAX A run of 500 iterations of SEM. The idea is that an SEM sequence is expected to enter rapidly in the neighbourhood of the global maximum of the likelihood function.

Default value is RANDOM.

2. **algorithm** defining the algorithms used in the strategy, the stopping rule and when to stop.

   a. Algorithms
      * EM Expectation Maximisation (cf. Section 3.1 of Statistical Documentation),
      * CEM Classification EM (cf. Section 3.3 of Statistical Documentation),
      * SEM Stochastic EM (cf. Section 3.2 of Statistical Documentation),
      * MAP Maximum a Posteriori,
      * M Only M step.

   b. Stopping rules for the algorithm
      * NBITERATION Sets the maximum number of iterations,
      * EPSILON Sets relative increase of the log-likelihood criterion.
      * NBITERATION_EPSILON Sets the maximum number of iterations and the epsilon value.

   Default values are 200 NBITERATION of EM with an EPSILON value of 10-4.

Criterion	Initialization	Algorithms	Stopping rules	Models Quantitatif/ Qualitatif
BIC	**RANDOM**	**EM**	**NBITERATION_EPSILON**	**'Gaussian_pk_Lk_Ck'/ 'Binary_pk_E**
CV	USER	CEM	EPSILON	
ICL	USER_PARTITION	SEM	NBITERATION	
NEC	SMALL_EM	MAP		
DCV	CEM_INIT	M		
	SEM_MAX			

Table 3.4: MIXMOD options bold strings represent default values. .

Warning 1 with HD models, only init USER_PARTITION + M and init USER + MAP (the two steps of Discriminant Analysis) are available.
Warning 2 with CEM algorithm, EPSILON = 0 is allowed.
Warning 3 with SEM algorithm, only NBITERATION is allowed.

3.3 MIXMOD Graphical User Interfaces (GUI)

This section describes the procedure to use MIXMOD graphical user's interfaces. This procedure is roughly speaking the same for Scilab and Matlab environments. The two functions *mixmodGraph.sci* (Scilab) and *mixmodGraph.m* (Matlab) are available, in the directory MIXMOD/, to launch MIXMOD graphical user's interface in Scilab (or Matlab) environment.

Remark With Scilab, you have to execute *initMixmod.sci* and with Matlab, you have to add MIXMOD and MIXMOD/UTIL/MATLAB directories to Matlab path before using *mixmodGraph.sci* or *mixmodGraph.m*.

```
-> (>>) mixmodGraph;
```

Once MIXMOD is launched, the *Main Menu* window appears. It contains the four following choices

- *Discriminant Analysis*, to start discriminant analysis,
- *Cluster Analysis* (or density estimation), to start cluster analysis,
- *Demos*, to start demonstration session,
- *Settings* (only for Scilab), to change the display mode,
- *Help*, provides further information on MIXMOD software.

3.3.1 Demo menu

We propose to begin by a demonstration session. A first contact with MIXMOD can be made by clicking on *Demos* in *Main Menu* and, selecting a data set example, for instance Birds.

3.3.1.1 Example with Birds data set

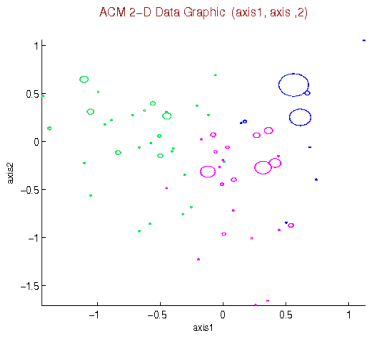


Figure 3.1: Birds demonstration: multinomial distribution with three components (general model $[Binary\_p_k\_E_{kjh}]$ see Table 3.2).

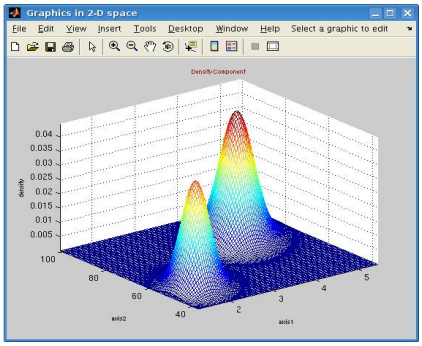


Figure 3.2: Old Faithful Geyser demonstration: Gaussian mixture model with two components (general model $[p_k\lambda_k C]$ see Table 3.1).

3.3.2 Cluster analysis

3.3.2.1 Getting started with the GUI

By clicking on *Cluster Analysis* in *Main Menu*, three windows appear successively to define required inputs for the analysis

- a window to enter the problem dimension (number of variables), and the array of modalities in qualitative case,
- an explorer window to select a data file (.dat file or others),
- a window to enter the number of clusters. One or several values can be entered. In this last case, MIXMOD returns the number of components giving the best results according to some chosen options.

Then cluster analysis can be run immediately by clicking on *Start Cluster Analysis* in the *Cluster Analysis Menu* or options can be chosen (see Section 3.2.1).

After running the program, a selection of outputs to be displayed is proposed in the *Output Menu*

- *View Diagnostics* numerical outputs visualisation mode for parameters, model(s), criterion(s), classification, probabilities and likelihood ;
- *Graphics* graphical outputs visualisation in Figure 3.2, the screen displays the Old Faithful Geyser results (two components have been used with the Ellipsoidal Gaussian model $[p_k\lambda_k C]$ see Table 3.1).

  If several criteria have been chosen in cluster analysis options, MIXMOD proposes to choose the criterion for which the output shall be displayed.
- *Save output variable* to save the output. It can be loaded in Scilab or Matlab and used with mixmodView function (see Section 3.3.2).
  Warning the name of the variable to draw is necessary **output**.

Example consider the *output* variable and *output.mat* saved in directory MIXMOD/:

```
-> load('output'); // create the output variable
-> mixmodView(output);
```

```
>> load('output.mat'); % create the output variable
>> mixmodView(output);
```

3.3.2.2 Cluster analysis options

By clicking on *Options* in *Cluster Analysis Menu*, the *Cluster Analysis Options Menu* appears. This menu, allows to select cluster analysis options among the following ones (see section 3.2.1 for more details)

- *Model*
  - *Gaussian Model* (Gaussian_pk_Lk_C by default),
  - *Qualitative Model* (Binary_pk_Ekjh by default),

    Remark Gaussian HD models are not available in cluster analysis.
- *Criteria* (BIC by default),
- *Strategy* (RANDOM initialization and 100 iterations of EM by default),
- *Weight for data* (1 by default),
- *Partition* (none by default, if given, it won't change during the execution).

3.3.3 Discriminant analysis

3.3.3.1 Getting started with the GUI

The user supplies classified observations (training observations) and will classify optional observations (remaining observations). In practice, the optional individuals must be given in another separate file with the same number of columns.

Now suppose that a set of K groups is available and for each observation, the group from which it arises is known. This group information is used to design a classification rule that assigns any vector observation to one of the groups.

Discriminant analysis can begin by clicking on *Discriminant Analysis* in *Main Menu*. Then required inputs for the analysis on training data must be defined

- the problem dimension (number of variables),
- the number of clusters of the training observations,
- the array of modalities in qualitative case
- selecting a training observation file (.dat file),
- selecting a *full* partition file of training observations (.part file).

Discriminant analysis can be run immediately by clicking on *Start Discriminant Analysis* in the *Discriminant Analysis Menu* or options can be chosen (see Section 3.3.3.2).

First, MIXMOD runs an M step to obtain a classification rule from the training observations. Moreover, if several models have been chosen in discriminant analysis options, MIXMOD returns the best model. After this step, the *Information Training Data Menu* appears to select the information on the training observations to be displayed

- *Reclassifying information by MAP rule*
  - *Reclassifying array of samples* displays an array $[K, K]$. Each value (i, j) represents the percentage of observations in group i classified in the group j after the M step.
  - *List of samples misclassified* displays the list of observations misclassified by MAP.
- *CV or DCV information (cf. sections 4.3.4 and 4.3.5 of Statistical documentation)*
  - CV Reclassifying array of samples by Cross Validation rule and list of samples misclassified.
  - DCV Double Cross Validation rate.
- *Model Parameter* displays the parameters of the best model. For gaussian HD models proportions, means, subDimension, parameters Akj, parameters Bk and orientation. For the others parameters, means and dispersion.

By clicking on *CONTINUE DISCRIMINANT ANALYSIS*, the user can select a remaining observations file to continue the analysis.

Then MIXMOD runs the second step of discriminant analysis (MAP step with classification rule computed in the first step). The aim of this step is to assign remaining observations to one of the groups. At the end of the analysis, an output can be displayed to see numerical or graphical results.

3.3.3.2 Discriminant analysis options

By clicking on *Options* in *Discriminant Analysis Menu*, the *Discriminant Analysis Options Menu* appears. This menu allows to select discriminant analysis options among the following choices

- *Gaussian Model-Based* (Gaussian_pk_Lk_C by default) or *Model* (Binary_pk_Ekjh by default), Remark Gaussian HD models are available, you have to give values for subDimensionFree and/or subDimensionEqual parameters. The parameter subDimensionEqual can be an integer or an array of integers, the parameter subDimensionFree can be a vector or an array of vectors.

- *Criteria* (CV by default),
- *Weight for data.*

Warning CV criterion gives pertinent results only if weights are integers.

3.4 MIXMOD toolkit functions

3.4.1 MIXMOD function

The functions *mixmod.sci* and *mixmod.m* (Scilab and Matlab) are available in the directory MIXMOD/.
Remark With Scilab, you have to execute *initMixmod.sci* and with Matlab, you have to add MIXMOD and MIXMOD/UTIL/MATLAB directories to Matlab path to load MIXMOD toolkit functions.

The calling sequence is

- quantitative case

```
-> (>>) [output [,outTab, optionalOutput]] = mixmod(data, nbCluster [,'criterion',criterion,
        'model',model,'weight', weight,'strategy',strategy,'partition',partition])
```

- qualitative case

```
 -> (>>) [output [,outTab, optionalOutput]] = mixmod(data, nbCluster, 'tabModality',tabModality
         [,'criterion',criterion,'model',model,'weight',weight,'strategy',strategy,'partition',partition])
```

In the following examples, we will either consider quantitative or qualitative data sets. The quantitative data set will be named *geyser* and the qualitative one *b_toby*.

The principal differences between qualitative and quantitative treatement is the presence of the variable 'tabModality' in the qualitative case.

3.4.1.1 Begining with MIXMOD

The simplest way (without options) to execute MIXMOD function is

```
-> geyser = read('DATA/geyser.dat',272,2);  |  >> geyser = load('DATA/geyser.dat');

-> b_toby = read('DATA/b_toby.dat',216,4);  |  >> b_toby = load('DATA/b_toby.dat');
```

```
-> (>>) nbCluster1 = 2;
-> (>>) tabModality = [2 ; 2 ; 2 ; 2];
-> (>>) out1 = mixmod(b_toby, nbCluster1,'tabModality',tabModality);
-> (>>) nbCluster2 = [2; 3];
-> (>>) out2 = mixmod(geyser, nbCluster2);
```

3.4.1.2 Executing MIXMOD with options

- criterion a vector of strings (scilab) or a cell array of strings (matlab)

```
-> criterion = 'BIC';                                  |  >> criterion = 'BIC';
-> out = mixmod(geyser, 2, 'criterion',criterion);     |  >> out = mixmod(geyser, 2, 'criterion',criterion);
-> criteria = ['BIC'; 'ICL'; 'NEC'];                   |  >> criteria = {'BIC'; 'ICL'; 'NEC'};
-> out = mixmod(geyser, 2 ,'criterion',criteria);      |  >> out = mixmod(geyser, 2, 'criterion',criteria);
```

- models a structure containing three fields name, subDimensionFree, subDimensionEqual defining the caracteristics of a model.

```
-> model = tlist(['model','name','subDimensionFree',
          'subDimensionEqual'],'',[],[]);
-> model.name = 'Binary_pk_Ekj';
-> out = mixmod(b_toby,2,'tabModality',tabModality,
              'model',list(model));

-> model1 = tlist(['model','name','subDimensionFree',
          'subDimensionEqual'],'Binary_pk_Ek',[],[]);
-> model2 = tlist(['model','name','subDimensionFree',
          'subDimensionEqual'],'Binary_pk_Ej',[],[]);
-> model3 = tlist(['model','name','subDimensionFree',
          'subDimensionEqual'],'Binary_pk_E',[],[]);
-> models = list(model1,model2,model3);
-> out = mixmod(b_toby,2,'tabModality',tabModality,
              'model',models);
```

```
>> model = struct('name','','subDimensionFree',
            [],'subDimensionEqual',[]);
>> model.name = 'Binary_pk_Ekj';
>> out = mixmod(b_toby, 2, 'tabModality',tabModality,
                'model',{model});
>> model1 = struct('name','Binary_pk_Ek','subDimensionFree',
             [],'subDimensionEqual',[]);
>> model2 = struct('name','Binary_pk_Ej','subDimensionFree',
            [],'subDimensionEqual',[]);
>> model3 = struct('name','Binary_pk_E','subDimensionFree',
             [],'subDimensionEqual',[]);
>> models = {model1,model2,model3};
>> out = mixmod(b_toby, 2, 'tabModality',tabModality,
                'model',models);
```

- weight a vector of real or integer values (scilab and matlab)

```
-> weight = read('DATA/b_toby.wgt',216,1);
-> out    = mixmod(b_toby, 2, 'tabModality',tabModality,
                 'weight',weight);
```

```
>> weight = load('DATA/b_toby.wgt');
>> out    = mixmod(b_toby, 2, 'tabModality',tabModality,
                    'weight',weight);
```

- partition a list of partition matrices (scilab) or a cell array of partition matrices (matlab)

```
-> part2 = read('DATA/geyser.part',272,2);
-> out = mixmod(geyser, 2, 'partition',list(part2));

-> part3 = read('DATA/geyser3clusters.part',272,3);
-> out = mixmod(geyser, [2;3], 'partition',
        list(part2, part3));
```

```
    >> part2 = load('DATA/geyser.part');
    >> out = mixmod(geyser, 2, 'partition',{part2});

    >> part3 = load('DATA/geyser3clusters.part');
    >> out = mixmod(geyser, [2;3], 'partition',
           {part2; part3});
```

- strategies a structure or a list of structures containing two fields, the first one defining the way of initializing the strategy (*initialization*) and the second the algorithms that can be used (*algorithm*).

  – strategies initialization type a structure (scilab and matlab)

```
-> init1  = tlist(['initialization','name','param',
     'partition'],'SMALL_EM',list(),list());
```

```
>>  init1 = struct('name','SMALL_EM','param',
       [],'partition',{{}});
```

```
-> init2 = tlist(['initialization','name','param',
     'partition'],'USER',list(),list());
-> init2.param = out.modelOutput(1).param;
```

```
>> init2 = struct('name','USER','param',
     [],'partition',{{}});
>> init2.param = out.modelOutput.param;
```

```
-> init3  = tlist(['initialization','name','param',
    'partition'],'USER_PARTITION',list(),list());
-> p2 = read('DATA/geyser.part',272,2);
-> p3 = read('DATA/geyser3clusters.part',272,3);
-> init3.partition = list(p2, p3);
```

```
>> init3 = struct('name','USER_PARTITION','param',
     [],'partition',{{}});
>> p2 = load('DATA/geyser.part');
>> p3 = load('DATA/geyser3clusters.part');
>> init3.partition = {p2, p3};
```

  – strategies algorithms type a structure (scilab and matlab)

```
-> algo1 = tlist(['algorithm','name','stopRule',
    'stopRuleValue'],'EM','EPSILON',0.0001);
-> algo2 = tlist(['algorithm','name','stopRule',
     'stopRuleValue'],'CEM',
     'NBITERATION_EPSILON',[100; 0.0001]);
```

```
>> algo1 = struct('name','EM','stopRule',{'EPSILON'},
    'stopRuleValue',[0.0001]);
>> algo2 = struct('name','CEM','stopRule',
   'NBITERATION_EPSILON',
   'stopRuleValue',[100 ; 0.0001]);
```

  – strategy example

```
-> init1 = tlist(['initialization','name','param',
    'partition'],'SMALL_EM',list(),list());
-> algo1 = tlist(['algorithm','name','stopRule',
    'stopRuleValue'],'EM','EPSILON',0.0001);
-> algo2 = tlist(['algorithm','name','stopRule',
    'stopRuleValue'],'CEM',
    'NBITERATION_EPSILON',[100; 0.0001]);
-> algos = list(algo1,algo2);
-> strategy  = tlist(['strategyType','initialization',
    'algorithm'],init1,algos);
-> out = mixmod(b_toby, 2, 'tabModality',tabModality,
              'strategy',strategy);
```

```
>> init1 = struct('name','SMALL_EM','param',
    [],'partition',{{}});
>> algo1 = struct('name','EM','stopRule',
    {'EPSILON'},'stopRuleValue',[0.0001]);
>> algo2 = struct('name','CEM','stopRule',
    'NBITERATION_EPSILON','stopRuleValue',
     [100 ; 0.0001]);
>> algos  = [algo1 ; algo2];
>> strategy = struct('initialization',init1,'algorithm',algos);
>> out = mixmod(b_toby, 2,'tabModality',tabModality,
               'strategy',strategy);
```

3.4.1.3 Output

MIXMOD functions return a required output structure *output* and two optional outputs *outTab* and *optionalOutput*.

Output structure A structure with two fields

- condExe containing the conditions of execution of MIXMOD.

  A structure with fifteen or sixteen fields

  - data the matrix $[n, d]$ of individuals (n number of samples, d number of variables) (see Section 3.1.1),
  - weight the vector of real or integer values defining the weight of the data,
  - dataType an integer defining the type of the data set (1 for gaussian data type and 2 for binary data type),
  - nbSamples an integer representing the number of samples n,
  - weightTotal a number representing the total weight,
  - pbDimension an integer representing the problem dimension d (number of variables),
  - nbCluster a vector of integers representing the number of clusters,
  - tabModality a vector of integers representing the modality of each variable in qualitative case,
  - criterion a vector of strings (scilab) or a cell array of strings (matlab) defining the type of criterion(ia),
  - modelType a list of model structures (scilab) or a cell array of model structures (matlab) defining the type of models,
  - strategy a list of strategy structures (scilab) or a vector of strategy structures (matlab) defining the type of strategy(ies),
  - mixmodError an integer representing the code error of mixmod execution (0 if mixmod runs successfully),
  - tabEstimError a vector representing the code error of each configuration (combination of strategy, number of clusters and model) (0 if no error),
  - tabCriterError a vector representing the code error in the criterion value calculation for each combination of strategy, number of clusters and model,
  - tabSelecError a vector representing the code error of selections of the best configuration.

  Examples of access to fields

```
-> (>>) output.condExe
-> (>>) output.condExe.nbSamples
-> (>>) output.condExe.pbDimension
-> (>>) output.condExe.nbCluster
-> (>>) output.condExe.criterion
-> (>>) output.condExe.modelType                  // (%) the first model (if several models)
-> (>>) output.condExe.strategy(1).initialization.name     // (%) the first strategy (if several strategies)
```

- modelOutput a list of outputs (scilab) or a vector of outputs (matlab) (for each criterion) for the best configuration (among all models, strategies and number of clusters).

  A structure with eight fields

  - type string giving the best model type,
  - nbCluster integer giving the best number of clusters,
  - criterion structure giving the name and the value of the criterion,
  - strategy the best strategy structure,

- param structure giving the parameters of the best model, (proportions, means and dispersion or proportions, means, subDimension, parameters_Akj, parameters_Bk, orientation)
- proba (only if standard output) structure giving the labels, the partition and the posterior probabilities of the best model,
- likelihood (only if standard output) structure giving the loglikelihood, the complete loglikelihood, the number of free parameters and the entropy for the best model.
- error the code error of the model.

Note in the quantitative case, the dispersion represents the variance matrix.

Examples of access to fields

```
-> (>>) output.modelOutput(1).type       // % for the 1st criterion
-> (>>) output.modelOutput(1).criterion  // % for the 1st criterion
-> (>>) output.modelOutput(1).param
```

```
-> (>>) output.modelOutput(1).criterion.name
-> (>>) output.modelOutput(1).criterion.value
-> (>>) output.modelOutput(1).nbCluster
-> (>>) output.modelOutput(1).param.proportion
-> (>>) output.modelOutput(1).param.mean
-> (>>) dis = output.modelOutput(1).param.dispersion
```

```
-> dis(1) // (%) variance of first mixture component
-> dis(2)// (%) variance of second mixture component
```

```
>> dis{1} // (%) variance of first mixture component
>> dis{2} // (%) variance of second mixture component
```

```
-> (>>) output.modelOutput(1).proba.label
-> (>>) output.modelOutput(1).proba.partition
-> (>>) output.modelOutput(1).proba.postProba
```

```
-> (>>) output.modelOutput(1).likelihood.logLikelihood
-> (>>) output.modelOutput(1).likelihood.completeLL
-> (>>) output.modelOutput(1).likelihood.nbFreeParam
-> (>>) output.modelOutput(1).likelihood.entropy
```

3.4.1.4 Optional output

Two optional outputs are available

- *outTab* a four dimension matrix which contains values of all criteria for all estimations (chosen by the user) all combinations of strategies, numbers of different cluster members and models. The first dimension represents strategies, the second the number of clusters, the third represents models and the last, criteria.

Example of using

```
-> nbCluster = [2; 3; 4];
-> criteria  = ['BIC'; 'ICL'];
-> model1 = tlist(['model','name','subDimensionFree',
            'subDimensionEqual'],'Gaussian_p_L_I',[],[]);
-> model2 = tlist(['model','name','subDimensionFree',
            'subDimensionEqual'],'Gaussian_p_Lk_I',[],[]);
-> model3 = tlist(['model','name','subDimensionFree',
            'subDimensionEqual'],'Gaussian_pk_L_I',[],[]);
-> model4 = tlist(['model','name','subDimensionFree',
           'subDimensionEqual'],'Gaussian_pk_Lk_I',[],[]);
-> models    = list(model1,model2,model3,model4);
```

```
>> nbCluster = [2; 3; 4];
>> criteria  = {'BIC'; 'ICL'};
>> model1 = struct('name','Gaussian_p_L_I','subDimensionFree',
               [],'subDimensionEqual',[]);
>> model2 = struct('name','Gaussian_p_Lk_I','subDimensionFree',
               [],'subDimensionEqual',[]);
>> model3 = struct('name','Gaussian_pk_L_I','subDimensionFree',
               [],'subDimensionEqual',[]);
>> model4 = struct('name','Gaussian_pk_Lk_I','subDimensionFree'
               [],'subDimensionEqual',[]);
>> models    = {model1,model2,model3,model4};
```

```
-> (>>) [out, outTab, optionalOutput] = mixmod(geyser, nbCluster, 'criterion',criteria, 'model',models);
```

```
-> (>>) value = outTab(1, 2, 1, 1);
```

This command returns the first criterion (BIC) value for the first strategy (default strategy), the second number of clusters (3) and the first model (Gaussian_p_L_I).

```
-> (>>) value = outTab(1, 1, 4, 2);
```

This command returns the second criterion (ICL) value for the first strategy (default strategy), the first number of clusters (2) and the fourth model (Gaussian_pk_Lk_I).

```
-> (>>) value = outTab(1,,:,:);
```

This command returns all criterion (BIC, ICL) values for the first strategy (default strategy), all number of clusters (2, 3, 4) and all models (24 values).

- *optionalOutput*: it is the same output as *output* but for all configurations (or estimations) (an estimation or configuration is a combination of a number of cluster, a model type and a strategy) except for the following fields:
  - criterion: The field criterion of *modelOutput* structure regroups informations about each criterion.
  - proba: empty.
  - likelihood: empty.

  Note: Number of estimation = number of number of clusters * number of models * number of strategies.

```
-> nbCluster = [2; 3; 4];
-> criteria  = ['BIC'; 'ICL'];
-> model1 = tlist(['model','name','subDimensionFree',
         'subDimensionEqual'],'Gaussian_p_L_I',[],[]);
-> model2 = tlist(['model','name','subDimensionFree',
         'subDimensionEqual'],'Gaussian_p_Lk_I',[],[]);
-> model3 = tlist(['model','name','subDimensionFree',
         'subDimensionEqual'],'Gaussian_pk_L_I',[],[]);
-> model4 = tlist(['model','name','subDimensionFree',
         'subDimensionEqual'],'Gaussian_pk_Lk_I',[],[]);
-> models    = list(model1,model2,model3,model4);
```

```
>> nbCluster = [2; 3; 4];
>> criteria  = {'BIC'; 'ICL'};
>> model1 = struct('name','Gaussian_p_L_I','subDimensionFree',
              [],'subDimensionEqual',[]);
>> model2 = struct('name','Gaussian_p_Lk_I','subDimensionFree',
              [],'subDimensionEqual',[]);
>> model3 = struct('name','Gaussian_pk_L_I','subDimensionFree',
              [],'subDimensionEqual',[]);
>> model4 = struct('name','Gaussian_pk_Lk_I','subDimensionFree',
             [],'subDimensionEqual',[]);
>> models    = {model1,model2,model3,model4};
```

```
-> (>>) [out, outTab, optionalOutput] = mixmod(geyser, nbCluster,'criterion',criteria,'model',models);
-> (>>) optionalOutput.condExe
-> (>>) optionalOutput.modelOutput(2).type
-> (>>) optionalOutput.modelOutput(12).param
```

In this case, there are twelve estimations for each criterion.

3.4.1.5 Examples

Examples are written if MIXMOD/ directory is the current directory.

- Scilab quantitative examples

```
data       = read('DATA/geyser.dat',272,2);
criterion  = ['BIC' ; 'ICL'];
model1 = tlist(['model','name','subDimensionFree','subDimensionEqual'],'Gaussian_p_L_I',[],[]);
model2 = tlist(['model','name','subDimensionFree','subDimensionEqual'],'Gaussian_p_Lk_I',[],[]);
model3 = tlist(['model','name','subDimensionFree','subDimensionEqual'],'Gaussian_pk_L_I',[],[]);
model4 = tlist(['model','name','subDimensionFree','subDimensionEqual'],'Gaussian_pk_Lk_I',[],[]);
model    = list(model1,model2,model3,model4);

weight     = read('DATA/geyser.wgt',272,1);

init  = tlist(['initialization','name','param','partition'],'SMALL_EM',list(),list());
```

```
algo1     = tlist(['algorithm','name','stopRule','stopRuleValue'],'EM','EPSILON',0.0001);
algo2     = tlist(['algorithm','name','stopRule','stopRuleValue'],'CEM','NBITERATION_EPSILON',[100 0.0001]);
algo   = list(algo1,algo2);
strategy  = tlist(['strategyType','initialization','algorithm'],init,algo);
output = mixmod(data,2);
output = mixmod(data,2,'weight',weight);
output = mixmod(data,[2;3],'criterion',criterion,'model',model,'strategy',strategy);
[output, outTab] = mixmod(data,[2;3],'model',model);
[output, outTab, optionalOutput] = mixmod(data,2,'criterion',criterion,'strategy',strategy,'weight',weight);

output = mixmod(data,2);
geyserPart2Clusters = output.modelOutput(1).proba.partition;
strategy.initialization.name = 'USER_PARTITION';
strategy.initialization.partition = list(geyserPart2Clusters);
output = mixmod(data,2,'strategy',strategy);

output = mixmod(data,2);
geyserPart2Clusters = output.modelOutput(1).proba.partition;
geyserPart3Clusters=zeros(272,3);
geyserPart3Clusters(1:100,1)=1;
geyserPart3Clusters(101:200,2)=1;
geyserPart3Clusters(201:272,3)=1;
strategy.initialization.partition = list(geyserPart2Clusters, geyserPart3Clusters);
output = mixmod(data,[2;3],'strategy',strategy);

output = mixmod(data,2);
strategy.initialization.name = 'USER';
strategy.initialization.param = list(output.modelOutput(1).param);
output = mixmod(data,2,'criterion',criterion,'model',model,'strategy',strategy);


// Using USER initialization


[criterion,strategy] = mixmodInputStrategy();

u=file('open','DATA/geyser.init','old');

nbCluster = 2;
pbDimension = 2;

proportion = [];
meand = list();
vard = list();
scatter = [];
for k=1:nbCluster
   proportion = [proportion read(u,1,1)];
   meand(k) = [];
   meand(k) = [read(u,1,pbDimension)];
   vard(k) = [];
   vard(k) = read(u,pbDimension,pbDimension);
end;

file('close',u);

tabParam = list(tlist(['parameterType','proportion','mean','dispersion'],proportion,meand,vard));
strategy.initialization.param = tabParam;

output = mixmod(data,3,'strategy',strategy);
```

Example for discriminant analysis (in directory MIXMOD/):

```
//Step 1:
dataTraining = read('DATA/geyser.dat',272,2);
partition    = read('DATA/geyser.part',272,2);
[criterion1,strategy1] = mixmodInputStrategy('DAstep1');
//Press Enter in Scilab

strategy1.initialization.partition = list(partition);
model = tlist(['model','name','subDimensionFree','subDimensionEqual'],'Gaussian_p_L_I',[],[]);
output = mixmod(dataTraining,2,'criterion',criterion1,'model',list(model),'strategy',strategy1,
        'partition',list(partition));
//Step 2:
[criterion2,strategy2] = mixmodInputStrategy('DAstep2');
//Press Enter in Scilab

strategy2.initialization.param = list(output.modelOutput(1).param);
dataRemaining = read('DATA/geyser.discriminant.dat',5,2);
output2 = mixmod(dataRemaining,2,'criterion',criterion2,'model',list(model),'strategy',strategy2);
```

Example for discriminant analysis with HD models

```
dataTraining = read('DATA/crabes.dat',200,5);
partition    = read('DATA/crabes.part',200,4);
model1 = tlist(['model','name','subDimensionFree','subDimensionEqual'],'Gaussian_HD_pk_AkjBkQkD'
          ,[],[2;3]);
model2 = tlist(['model','name','subDimensionFree','subDimensionEqual'],'Gaussian_HD_pk_AkjBkQkDk'
         ,[1,2,2,1;2,1,1,2],[]);
model = list(model1,model2);
[criterion1,strategy1] = mixmodInputStrategy('DAstep1');
//Press Enter in Scilab

strategy1.initialization.partition = list(partition);
output = mixmod(dataTraining,4,'criterion',criterion1,'model',model,'strategy',strategy1,'partition'
         ,list(partition));

//Step 2:
bestModel = output.modelOutput(1).type;
[criterion2,strategy2] = mixmodInputStrategy('DAstep2');
//Press Enter in Scilab

strategy2.initialization.param = list(output.modelOutput(1).param);
dataRemaining = read('DATA/crabes_10indiv.dat',10,5);
output2 = mixmod(dataRemaining,4,'criterion',criterion2,'model',list(bestModel),'strategy',strategy2);
```

- Scilab qualitative examples

```
data = read('DATA/birds.dat',153,6);
partition    = read('DATA/birds.part',153,3);
tabModality = [2 ; 4 ; 5 ; 2 ; 5 ; 3];
criterion = ['BIC','ICL', 'NEC'];

model1 = tlist(['model','name','subDimensionFree','subDimensionEqual'],'Binary_pk_Ekjh',[],[]);
model2 = tlist(['model','name','subDimensionFree','subDimensionEqual'],'Binary_pk_Ekj',[],[]);
model3 = tlist(['model','name','subDimensionFree','subDimensionEqual'],'Binary_pk_Ek',[],[]);
model4 = tlist(['model','name','subDimensionFree','subDimensionEqual'],'Binary_pk_Ej',[],[]);
model5 = tlist(['model','name','subDimensionFree','subDimensionEqual'],'Binary_pk_E',[],[]);
model    = list(model1,model2,model3,model4,model5);

weight    = read('DATA/birds.lab',153,1);

init1 = tlist(['initialization','name','param','partition'],'CEM_INIT',list(),list());
algo1 = tlist(['algorithm','name','stopRule','stopRuleValue'], 'EM', 'NBITERATION_EPSILON',[300,0.0001]);
algo2 = tlist(['algorithm','name','stopRule','stopRuleValue'], 'CEM', 'EPSILON',0.001);
strategy = tlist(['strategyType','initialization','algorithm'],init1,list(algo1, algo2));

output = mixmod(data,[2;3],'tabModality',tabModality);
output = mixmod(data,[2;3],'tabModality',tabModality,'criterion',criterion);
output = mixmod(data,2,'tabModality',tabModality,'strategy',strategy);
output = mixmod(data,2,'tabModality',tabModality,'weight',weight,'strategy',strategy);
output = mixmod(data,3,'tabModality',tabModality,'partition',list(partition));
[output,outTab] = mixmod(data,2,'tabModality',tabModality,'criterion',criterion,'model',model);

strategy.initialization.name = 'USER';
strategy.initialization.param = list(output.modelOutput(1).param);
out = mixmod(data,2,'tabModality',tabModality,'strategy',strategy);

mixmodView(out);
mixmodView(out, 'mca1D');
mixmodView(out, 'mca2D', 'pointClass');
mixmodView(out, 'mca3D', 'class');

mixmodView(out, [1],'point');
mixmodView(out, [1 2],'class');
```

Example for discriminant analysis (in directory MIXMOD/):

```
//Step 1:
dataTraining = read('DATA/b_toby.dat',216,4);
partition    = read('DATA/b_toby.part',216,2);
```

```
model = tlist(['model','name','subDimensionFree','subDimensionEqual'],'Binary_pk_Ekjh',[],[]);
tabModality = [2;2;2;2];
[criterion1,strategy1] = mixmodInputStrategy('DAstep1');
//Press Enter in Scilab

strategy1.initialization.partition = list(partition);
output = mixmod(dataTraining,2,'tabModality',tabModality,'criterion',criterion1,'model',list(model),'strategy'
,strategy1,'partition',list(partition));

//Step 2:
[criterion2,strategy2] = mixmodInputStrategy('DAstep2');
//Press Enter in Scilab

strategy2.initialization.param = list(output.modelOutput(1).param);
dataRemaining = read('DATA/b_toby.discriminant.dat',5,4);
output2 = mixmod(dataRemaining,2,'tabModality',tabModality,'criterion',criterion2,'model',list(model),
'strategy',strategy2);
```

- Matlab quantitative examples

```
data = load('DATA/geyser.dat');
criterion  = {'BIC' ; 'ICL'};
model1 = struct('name','Gaussian_p_L_I','subDimensionFree',[],'subDimensionEqual',[]);
model2 = struct('name','Gaussian_p_Lk_I','subDimensionFree',[],'subDimensionEqual',[]);
model3 = struct('name','Gaussian_pk_L_I','subDimensionFree',[],'subDimensionEqual',[]);
model4 = struct('name','Gaussian_pk_Lk_I','subDimensionFree',[],'subDimensionEqual',[]);
model      = {model1,model2,model3,model4};
weight     = load('DATA/geyser.wgt');


init = struct('name','SMALL_EM','param',[],'partition',{{}});
algo1    = struct('name','EM','stopRule',{'EPSILON'},'stopRuleValue',[0.0001]);
algo2    = struct('name','CEM','stopRule','NBITERATION_EPSILON','stopRuleValue',[100 ; 0.0001]);
tabAlgo  = [algo1 ; algo2];

strategy = struct('initialization',init,'algorithm',tabAlgo);

output = mixmod(data,[2]);
output = mixmod(data,2,'weight',weight);
output = mixmod(data,[2;3],'criterion',criterion,'model',model,'strategy',[strategy]);
[output, outTab] = mixmod(data,[2;3],'model',model);
[output, outTab, optionalOutput] = mixmod(data,2,'criterion',criterion,'model',model,'strategy',[strategy],'weight'
,weight);

output = mixmod(data,2);
partition2Clusters = output.modelOutput(1).proba.partition;
strategy.initialization.name = 'USER_PARTITION';
strategy.initialization.partition = {partition2Clusters};
output = mixmod(data,2,'strategy',strategy);

output = mixmod(data,2);
partition2Clusters = output.modelOutput(1).proba.partition;
partition3Clsuters = zeros(272,3);
partition3Clsuters(1:100,1)=1;
partition3Clsuters(101:200,2)=1;
partition3Clsuters(201:272,3)=1;
strategy.initialization.partition = {partition2Clusters;  partition3Clsuters};
output = mixmod(data,[2;3],'strategy',strategy);

strategy.initialization.name = 'USER';
strategy.initialization.param = [output.modelOutput(1).param];
output = mixmod(data,2,'criterion',criterion,'model',model,'strategy',strategy);

% Using USER initialization
[criteion,strategy] = mixmodInputStrategy();
u  = fopen('DATA/geyser.init');

parameter = [];
meand      = {};
vard   = {};
nbCluster = 2;
pbDimension = 2;
   for k=1:nbCluster
      parameter = [parameter, fscanf(u,'%g',1)];
```

```
      meand{k} = fscanf(u,'%g',[1 pbDimension]);
      vard{k} = [];
      vard{k} = fscanf(u,'%g',[pbDimension pbDimension]);
   end;
fclose(u);
tabParam = [struct('proportion',{parameter},'mean',{meand},'dispersion',{vard})];
strategy.initialization.param = tabParam;
output = mixmod(data,2,'strategy',strategy);
```

Example for discriminant analysis (in directory MIXMOD/):

```
% Step 1:
dataTraining = load('DATA/geyser.dat');
partition    = load('DATA/geyser.part');
model = struct('name','Gaussian_p_L_I','subDimensionFree',[],'subDimensionEqual',[]);
[criterion1,strategy1] = mixmodInputStrategy('DAstep1');
strategy1.initialization.partition = {partition};
output = mixmod(dataTraining,2,'criterion',criterion1,'model',{model},'strategy',strategy1,'partition',
{partition});

% Step 2:
model = struct('name','Gaussian_p_L_I','subDimensionFree',[],'subDimensionEqual',[]);
[criterion2,strategy2] = mixmodInputStrategy('DAstep2');
strategy2.initialization.param = [output.modelOutput.param];
dataRemaining = load('DATA/geyser.discriminant.dat');
output2 = mixmod(dataRemaining,2,'criterion',criterion2,'model',{model},'strategy',strategy2);
```

Example for discriminant analysis with HD models

```
dataTraining = load('DATA/crabes.dat');
partition    = load('DATA/crabes.part');
model1 = struct('name','Gaussian_HD_pk_AkjBkQkD','subDimensionFree',[],'subDimensionEqual',[2;3]);
model2 = struct('name','Gaussian_HD_pk_AkjBkQkDk','subDimensionFree',[1,2,2,1;2,1,1,2],'subDimensionEqual',[]);
model = {model1,model2};
[criterion1,strategy1] = mixmodInputStrategy('DAstep1');

strategy1.initialization.partition = {partition};
output = mixmod(dataTraining,4,'criterion',criterion1,'model',model,'strategy',strategy1,'partition',{partition});

%Step 2:
bestModel = output.modelOutput(1).type;
[criterion2,strategy2] = mixmodInputStrategy('DAstep2');

strategy2.initialization.param = [output.modelOutput(1).param];
dataRemaining = load('DATA/crabes_10indiv.dat');
output2 = mixmod(dataRemaining,4,'criterion',criterion2,'model',{bestModel},'strategy',strategy2);
```

- Matlab qualitative examples

```
data = load('DATA/birds.dat');
partition = load('DATA/birds.part');
tabModality = [2 ; 4 ; 5 ; 2 ; 5 ; 3];
criterion = {'BIC','ICL','NEC'};
model1 = struct('name','Binary_pk_Ekjh','subDimensionFree',[],'subDimensionEqual',[]);
model2 = struct('name','Binary_pk_Ekj','subDimensionFree',[],'subDimensionEqual',[]);
model3 = struct('name','Binary_pk_Ej','subDimensionFree',[],'subDimensionEqual',[]);
model4 = struct('name','Binary_pk_E','subDimensionFree',[],'subDimensionEqual',[]);

model = {model1,model2,model3,model4};
weight = load('DATA/birds.lab');

init = struct('name','CEM_INIT','param',[],'partition',{{}});
algo1 = struct('name','EM','stopRule','NBITERATION_EPSILON','stopRuleValue',[300,0.0001]);
algo2 = struct('name','CEM','stopRule','EPSILON','stopRuleValue',0.001);

strategy = struct('initialization',init,'algorithm',[algo1;algo2]);
```

```
output = mixmod(data,[2:3],'tabModality',tabModality);
output = mixmod(data,[2:3],'tabModality',tabModality,'criterion',criterion);
output = mixmod(data,2,'tabModality',tabModality,'strategy',strategy);
output = mixmod(data,3,'tabModality',tabModality,'weight',weight,'strategy',strategy);
output = mixmod(data,2,'tabModality',tabModality,'partition',{partition});
[output,outTab] = mixmod(data,2,'tabModality',tabModality,'criterion',criterion,'model',model);


strategy.initialization.name = 'USER';
strategy.initialization.param = [output.modelOutput(1).param];
out = mixmod(data,2,'tabModality',tabModality,'strategy',strategy);

mixmodView(out);
mixmodView(out, 'mca1D');
mixmodView(out, 'mca2D', 'pointClass');
mixmodView(out, 'mca3D', 'class');

mixmodView(out, [1],'point');
mixmodView(out, [1 2],'class');
```

Example for discriminant analysis (in directory MIXMOD/):

```
//Step 1:
dataTraining = load('DATA/b_toby.dat');
tabModality = [2;2;2;2];
partition    = load('DATA/b_toby.part');
model = struct('name','Binary_pk_Ekjh','subDimensionFree',[],'subDimensionEqual',[]);
[criterion1,strategy1] = mixmodInputStrategy('DAstep1');
strategy1.initialization.partition = {partition};
output = mixmod(dataTraining,2,'tabModality',tabModality,'criterion',criterion1,'model',{model},
'strategy',strategy1,'partition',{partition});
//Step 2:
[criterion2,strategy2] = mixmodInputStrategy('DAstep2');
strategy2.initialization.param = [output.modelOutput(1).param];
dataRemaining = load('DATA/b_toby.discriminant.dat');
output2 = mixmod(dataRemaining,2,'tabModality',tabModality,'criterion',criterion2,'model',{model},'strategy'
,strategy2);
```

3.4.2 MIXMODVIEW functions

The *mixmodView.m* and *mixmodView.sci* functions are available in the directory MIXMOD/. They allow to display graphics (density, iso-density,...), in 1-D, 2-D and 3-D spaces, of the output coming from an execution of the MIXMOD function.

The calling sequence is

```
-> (>>) mixmodView(output, [axis, marker, density, isoDensity])
```

All the options are not always available, depending on the problem dimension and the data type (see section 3.4.2.3).

3.4.2.1 Required Input

There is only one required input

1. *output* output structure resulting from an execution of the *mixmod* function (Scilab or Matlab) on the data set.

Example

```
-> geyser  = read('DATA/geyser.dat',272,2);
-> outGeys = mixmod(geyser,2);
-> mixmodView(outGeys);

-> b_toby  = read('DATA/b_toby.dat',216,4);
-> outB_toby = mixmod(b_toby,2,'tabModality',
                [2 ; 2 ; 2 ; 2]);
-> mixmodView(outB_toby);
```

```
>> geyser  = load('DATA/geyser.dat');
>> outGeys = mixmod(geyser,2);
>> mixmodView(outGeys);

>> b_toby  = load('DATA/b_toby.dat');
>> outB_toby = mixmod(b_toby,2,'tabModality',
                [2 ; 2 ; 2 ; 2]);
>> mixmodView(outB_toby);
```

3.4.2.2 Optional Inputs

There are five optional inputs

1. *axis* an array or a string defining axis in scilab or matlab.
   - Each array has dimension between 1 and 3
     Possible values are
     - 1-D space `[1], [2], [3],...` .
     - 2-D space `[1 2], [2 3], [1 3],...` .
     - 3-D space `[1 2 3], [1 2 4], [1 3 4], [2 3 4],...` .
   - string values
     - 1-D space
       * `'pca1D'` for the 1rst axis of Principal Component Analysis (quantitative data).
       * `'fda1D'` for the 1rst axis of Factoriel Discriminant Analysis (quantitative data).
       * `'mca1D'` for the 1rst axis of Multiple Correspondance Analysis (qualitative data).
     - 2-D space
       * `'pca2D'` for the 1rst plane of Principal Component Analysis (quantitative data).
       * `'fda2D'` for the 1rst plane of Factoriel Discriminant Analysis (quantitative data).
       * `'mca2D'` for the 1rst plane of Multiple Correspondance Analysis (qualitative data) each circle is centered on an individual and the diameter depends of the weight associated to the individual.
     - 3-D space
       * `'pca3D'` for the 1rst 3-D axes of Principal Component Analysis (quantitative data).
       * `'fda3D'` for the 1rst 3-D axes of Factoriel Discriminant Analysis (quantitative data).
       * `'mca3D'` for the 1rst 3-D axes of Multiple Correspondance Analysis (qualiitative data) each individual is represented.
2. *marker* a string defining the way samples points are displayed.
   Possible values are
   - `'point'` samples diplayed by black points,
   - `'class'` samples displayed by colored symbol for each cluster,
   - `'pointClass'` samples displayed by black points and by colored symbols,
   - `'number'` samples displayed by their number in black string,
   - `'classNumber'`: samples displayed by their number in colored string (one color by cluster).
3. *density* a string to define the type of density to be displayed.
   Possible values are
   - `'densityComponent'` only the density of each component (cluster) is displayed,
   - `'densityMixture'` only the density mixture is displayed,
   - `'bothDensity'` both density types are displayed.
4. *isoDensity* a string to define the type of iso-density to be displayed (not available in qualitative case).
   Possible value are
   - `'isoDensityComponent'` display points of same density for each cluster,
   - `'isoDensityMixture'` display points of same density for mixture,
   - `'bothIsoDensity'` display both iso-densities on different graphics.

3.4.2.3 Summary of mixmodView inputs

- Quantitative data

  This table describes allowed inputs according to the problem dimension (the number of variables) for quantitative data.
  Bold strings define default values.
  Italic strings define values which are only available in Matlab environment.

1D	**2D**	**3D**
[i] or pca1D or fda1D	[i j] or pca2D or fda2D	[i j k] or pca3D or fda3D
point class **pointClass**	point class **pointClass** number classNumber	point class **pointClass** *number* *classNumber*
densityMixture densityComponent **bothDensity**	densityMixture densityComponent **bothDensity**	
	isoDensityMixture isoDensityComponent **bothIsoDensity**	**isoDensityComponent**

- Qualitative data

  This table describes allowed inputs according to the problem dimension (the number of variables) for qualitative data.
  Bold strings define default values.

1D	**2D**	**3D**
[i] or mca1D	[i j] or mca2D	mca3D
point class **pointClass**	point class **pointClass**	point class **pointClass**
densityComponent	**densityComponent**	**densityComponent**

Example [i] = [1], [2] ; [i j] = [1 2], [2 3] ; [i j k] = [1 2 3] ; [2 3 1].

3.4.2.4 Examples

Let's consider *iris* and *b_toby* data sets in directory MIXMOD/DATA/. The following examples are executed in the directory MIXMOD/

```
-> iris    = read('DATA/iris.dat',150,4);
-> b_toby  = read('DATA/b_toby.dat',216,4);
-> outIris = mixmod(iris,3);
-> outB_toby = mixmod(b_toby,2,'tabModality'
               ,[2 ; 2 ; 2 ; 2]);
```

```
>> iris    = load('DATA/iris.dat');
>> b_toby  = load('DATA/b_toby.dat');
>> outIris = mixmod(iris,3);
>> outB_toby = mixmod(b_toby,2,'tabModality'
               ,[2 ; 2 ; 2 ; 2]);
```

- Example 1

```
-> (>>) mixmodView(outIris);
```

This command displays graphics with default options for gaussian data (with pbDimension=4)
- 'pointClass' in 'pca2D',
- 'bothDensity' in 'pca2D',
- 'bothIsoDensityComponent' in 'pca2D'.

```
-> (>>) mixmodView(outB_toby);
```

This command displays graphics with default options for qualitative data (with pbDimension=4)
- 'pointClass' in 'mca3D',
- 'densityComponent'.

- Example 2

```
-> (>>) mixmodView(outIris,[1],'bothDensity');
```

This command displays density mixture and density components for axes [1].

```
-> (>>) mixmodView(outB_toby,[2],'class');
```

This command displays histograms of the individuals on the second axis for each class.

- Example 3

```
-> (>>) mixmodView(outIris, 'pca1D','bothDensity');
```

This command displays density components and density mixtures in 'pca1D'.

```
-> (>>)  mixmodView(outB_toby, 'mca2D');
```

This command displays one graphic representing the indivuduals in the first plane of MCA.

- Example 4

```
-> (>>) mixmodView(outIris, [1 2],'point', 'densityComponent','isoDensityComponent');
```

This command displays density components and iso-density components on axis [1 2]. Samples are displayed by black points.

```
-> (>>)  mixmodView(outB_toby,[1 2],'class');
```

This command displays histograms of the individuals on [1 2] of the entire data set and for each class.

- Example 5:

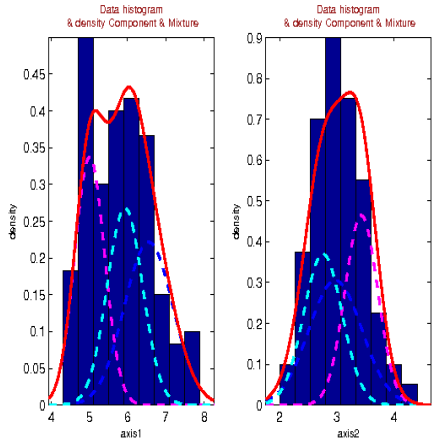


Figure 3.3: Density Mixture and Component

```
-> (>>)   mixmodView(outIris, [2], 'densityMixture');
```

displays the following window

```
-> (>>)  mixmodView(outB_toby, [2],'pointClass');
```

displays the following window:

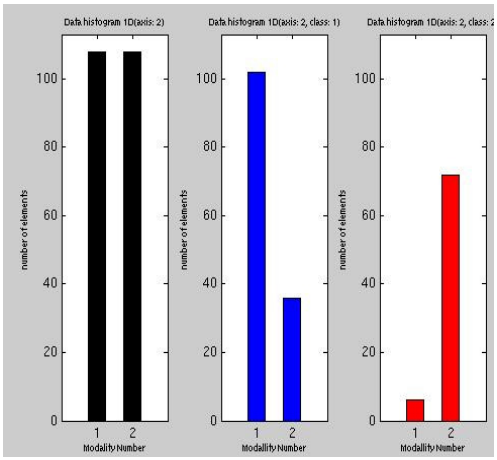


Figure 3.4: Individuals on the first and the second axis for the entire data set and for each class.

- Example 6:

```
-> (>>)  mixmodView(outIris, [1 2], 'densityMixture','isoDensityMixture','point');
```

displays the following window:

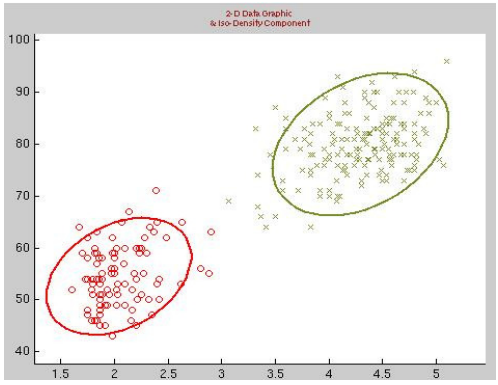


Figure 3.5: Iso-density Component

```
-> (>>)  mixmodView(outB_toby, 'densityComponent');
```

displays the following window:

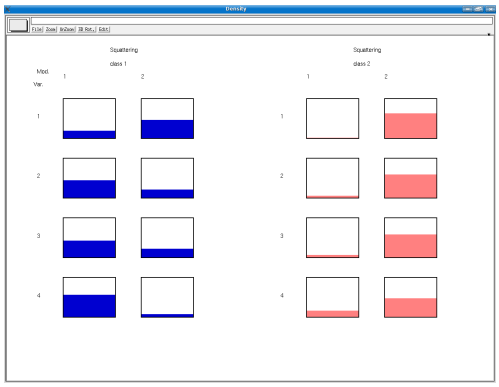


Figure 3.6: Density component.

3.4.3 Other functions

3.4.3.1 printMixmod functions

The functions *printMixmod.m* and *printMixmod.sci* are available in the directory MIXMOD/. They allow to display the main information of the output coming from an execution of mixmod function (Scilab or Matlab).
The calling sequence is

```
-> (>>) printMixmod(output)
```

output is a required input.

Example

```
-> geyser = read('DATA\geyser.dat',272,2);
-> output = mixmod(geyser,2);
-> printMixmod(output);
```

```
>> geyser   = load('DATA/geyser.dat');
>> output = mixmod(geyser,2);
>> printMixmod(output);
```

```
Example of printMixmod output
 ------------------------------------------------------------
 *  Number of samples 272
 *  Problem dimension 2
 ------------------------------------------------------------
 *           Criterion type BIC
 *          Criterion value 2322.97
 *        Number of clusters 2
 *                Model type Gaussian_pk_Lk_C
 *               Proportions list by cluster
                             . cluster1   0.35712
                             . cluster2   0.64288
 *                     Means list by mean vector of clusters
                             . cluster1   2.03968      54.5168
                             . cluster2    4.2922      79.9962
 *                 Variances list by matrix variance of clusters
                             . cluster1   0.098359      0.56174
                                           0.56174      27.6803
                             . cluster2   0.145347       0.8301
                                            0.8301       40.904
 *           Log-likelihood -1136.26
 *  Completed log-likelihood -1136.47
 * Number of free parameters 9
 *                   Entropy 0.66452
 ------------------------------------------------------------
```

3.4.3.2 MIXMODINPUT functions

The functions are available in the directory MIXMOD/.

- ***mixmodInputCriterion.m or mixmodInputCriterion.sci***

  This function is meant to help you change *criterion* variable. The calling sequence is

  ```
  -> (>>) criterion = mixmodInputCriterion([str]);
  ```

  str is an optional input and available values are

  – 'allClusteringCriteria' returns all criteria types available for Cluster Analysis (BIC,ICL,NEC),

  – 'allDiscriminantCriteria' returns all criteria types available for Discriminant Analysis (CV, DCV).

  The output is a vector of strings (scilab) or a cell array of strings (matlab) representing criterion type. If *str* is not given, *criterion* is set to default value 'BIC'.

  Example

  ```
  -> criterion=mixmodInputCriterion();
  ```

  ```
  >> criterion=mixmodInputCriterion();
  ```

  ```
  -> criterion=mixmodInputCriterion('allClusteringCriteria');
  ```

  ```
  >> criterion=mixmodInputCriterion('allClusteringCriteria');
  ```

- ***mixmodInputModel.m or mixmodInputModel.sci***

  This function is meant to help you change *model* variable. The calling sequence is

  ```
  -> (>>) model = mixmodInputModel([str1,str2]);
  ```

  str1 or str2 is an optional input and available values are

  – 'allGaussianModels' returns all gaussian model types.

- 'sphericalModels' returns all spherical model types.
- 'diagonalModels' returns all diagonal model types
- 'generalModels' returns all general model types.
- 'allBinaryModels' returns all qualitative model types.
- 'allGaussianHDModels' returns all gaussian HD model types.

The output is a list of structure (scilab) or a cell array of structure (matlab) representing model type. If *str* is not given, *model* is set to default value 'Gaussian_pk_Lk_C'. For HD models, you have to give subDimensionFree and subDimensionEqual.

Example

```
-> model = mixmodInputModel();

-> model = mixmodInputModel('allGaussianModels',
           'allGaussianHDModels');
```

```
>> model = mixmodInputModel();

>> model = mixmodInputModel('allGaussianModels',
           'allGaussianHDModels');
```

- ***mixmodInputStrategy.m or mixmodInputStrategy.sci***

This function is meant to help you change *criterion* and *strategy* variables. The calling sequence is

```
-> (>>) [criterion,strategy] = mixmodInputStrategy([str]);
```

str is an optional input and available values are

- 'DAstep1' returns strategy for first step of discriminant analysis.
- 'DAstep2' returns strategy for second step of discriminant analysis.
- 'DAallStep': returns strategies for the two steps of discriminant analysis.

The output is a vector of strings (*criterion*) and a list of strategy structure (*strategy*) with scilab or a cell array of strings (*criterion*) and a vector of strategy structure (*strategy*) with matlab. If *str* is not given, *criterion* is set to default value 'BIC' and *strategy* is set to RANDOM initialization and 100 iterations (and 0.0001 stop xml criterion value) of the EM algorithm.

Example

```
-> [criterion,strategy] = mixmodInputStrategy();

-> [criterion,strategy] = mixmodInputStrategy('DAstep1');
```

```
>> model = mixmodInputStrategy();

>> model = mixmodInputStrategy('DAstep1');
```

Example 4 using mixmodInput functions for discriminant analysis in two steps

Scilab

```
// Step 1:
   -> dataTraining = read('DATA/geyser.dat',272,2);
   -> partition    = read('DATA/geyser.part',272,2);
   -> model = tlist(['model','name','subDimensionFree','subDimensionEqual'],'Gaussian_p_L_I',[],[]);
   -> [criterion1,strategy1] = mixmodInputStrategy('DAstep1');
   //Press Enter in Scilab

   -> strategy1.initialization.partition = list(partition);
   -> output = mixmod(dataTraining,2,'criterion',criterion1,'model',list(model),'strategy',strategy1,
               'partition',list(partition));
// Step 2:
   -> [criterion2,strategy2] = mixmodInputStrategy('DAstep2');
   //Press Enter in Scilab

   -> strategy2.initialization.param = list(output.modelOutput(1).param);
   -> dataRemaining = read('DATA/geyser.discriminant.dat',5,2);
   -> output2 = mixmod(dataRemaining,2,'criterion',criterion2,'model',list(model),'strategy',strategy2);
```

Matlab

```
%% Step 1:
   >> dataTraining = load('DATA/geyser.dat');
   >> model = struct('name','Gaussian_p_L_I','subDimensionFree',[],'subDimensionEqual',[]);
   >> partition    = load('DATA/geyser.part');
   >> [criterion1,strategy1] = mixmodInputStrategy('DAstep1');
   >> strategy1.initialization.partition = {partition};
   >> output = mixmod(dataTraining,2,'criterion',criterion1,'model',{model},'strategy',strategy1,
              'partition',{partition});
%% Step 2:
   >> [criterion2,strategy2] = mixmodInputStrategy('DAstep2');
   >> strategy2.initialization.param = [output.modelOutput.param];
   >> dataRemaining = load('DATA/geyser.discriminant.dat');
   >> output2 = mixmod(dataRemaining,2,'criterion',criterion2,'model',{model},'strategy',strategy2);
```

Example 5 using mixmodInput functions for discriminant analysis in one step
Scilab

```
-> dataTraining = read('DATA/geyser.dat',272,2);
-> dataRemaining = read('DATA/geyser.discriminant.dat',5,2);
-> model = tlist(['model','name','subDimensionFree','subDimensionEqual'],'Gaussian_p_L_I',[],[]);
-> partition    = read('DATA/geyser.part',272,2);
-> [criterion,strategy] = mixmodInputStrategy('DAallStep');
//Press Enter in Scilab

-> strategy(1).initialization.partition = list(partition);
-> output1 = mixmod(dataTraining,2,'criterion',criterion,'model',list(model),'strategy',strategy(1),
             'partition',list(partition));

-> strategy(2).initialization.param = list((output1.modelOutput(1).param));
-> output2 = mixmod(dataRemaining,2,'criterion','BIC','model',list(model),'strategy',strategy(2));
```

Matlab

```
>> dataTraining = load('DATA/geyser.dat');
>> dataRemaining = load('DATA/geyser.discriminant.dat');
>> model = struct('name','Gaussian_p_L_I','subDimensionFree',[],'subDimensionEqual',[]);
>> partition    = load('DATA/geyser.part');
>> [criterion,strategy] = mixmodInputStrategy('DAallStep');
>> strategy(1).initialization.partition = {partition};
>> output = mixmod(dataTraining,2,'criterion',criterion,'model',{model},'strategy',strategy(1),'partition',{partition});

>> strategy(2).initialization.param = [output.modelOutput(1).param];
>> output2 = mixmod(dataRemaining,2,'criterion','BIC','model',{model},'strategy',strategy(2));
```

3.5 MIXMOD command line (expert use)

This section outlines the operations and the available options of the package for an on-line use. Instructions on the form of the input and output files are also given. MIXMOD is called by MIXMOD's executable file in the MIXMOD/BIN/ directory.
Results are stored in the output text files.

Synopsis

```
mixmod  input_filename (or mixmod.exe input_filename for windows)
```

Example

```
mixmod  ../EXAMPLES/geyser.xem
```

3.5.1 Input file

3.5.1.1 File structure for Cluster Analysis

MIXMOD needs an input data file with arguments that reflect the chosen preferences or needs.
An input data file consists in a succession of keywords followed by values that precise the caracteristics of the input data. They make this input data file easier to write because there is no order in the definition except for the strategies.

Two types of inputs are availables

- Required Inputs
  - *NbLines* Specifies the number of rows in the data set file.
  - *NbNbCluster* Specifies the length of the list of number of clusters.
  - *ListNbCluster* Specifies the list of number of clusters defined by NbNbCluster.
  - *DataFile* Specifies the path and name of the data file.
  - *PbDimension* Specifies the problem dimension an integer which gives the number of colums in the data file.
  - In qualitative case
    - *NbModality* Specifies the modalities on each variable.
- Optional Inputs
  - criterion,
  - models,
  - weight file,
  - strategy.

Let's consider the input files *geyserDefault.xem*, which is in the package directory MIXMOD/EXAMPLES/. It contains the required inputs and all the optional inputs are initialized with default values

```
NbLines
    272                     [Number of lines in the data file]
PbDimension
    2                       [Dimension sample]
NbNbCluster
    1                       [List of size 1]
ListNbCluster
    2                       [2 components specified]
DataFile
    ../DATA/geyser.dat      [Input data set (Old Faithful Geyser)]
```

To run this example under Linux/Unix, use the following command in MIXMOD/BIN/ directory.

```
mixmod  ../EXAMPLES/geyserDefault.xem
```

Example of an input file with strategy geyserMiniUSER.xem

```
NbLines
    272
PbDimension
    2
NbNbCluster
    1
ListNbCluster
    2
DataFile
    ../DATA/geyser.dat

NbStrategy
    1
InitType
    USER                        [Starting parameter specified by user]
InitFile
    ../DATA/geyser.init         [Input file for parameter initialisation for 2 clusters]
NbAlgorithm
    1
Algorithm
    EM
StopRule
    NBITERATION
StopRuleValue
    200
```

An input file with all options geyser.xem

```
NbLines
    272                     [Number of lines in the data file]
PbDimension
    2                       [Dimension sample]
NbCriterion
    1                       [One criterion]
ListCriterion
    BIC                     [BIC criterion]
NbNbCluster
    1                       [List of size 1]
ListNbCluster
    2                       [2 components specified]
NbModel
    1                       [One model]
ListModel
    Gaussian_pk_Lk_I        [Gaussian spherical model [pk_Lk_I]]
NbStrategy
    1                       [One strategy]
InitType
    RANDOM                  [Starting parameter by random centers]
NbAlgorithm
    1                       [One algorithm in the strategy]
Algorithm
    EM                      [EM algorithm]
StopRule
    NBITERATION             [Stopping rule for EM is number of iterations]
StopRuleValue
    200                     [200 iterations desired]
DataFile
    ../DATA/geyser.dat      [Input data set (Old Faithful Geyser)]
WeightFile
    ../DATA/geyser.wgt      [Weight file for data]
```

Many examples are available in directories MIXMOD/EXAMPLE and MIXMOD/TEST.

3.5.1.2 File structure for Discriminant Analysis

To execute Discriminant Analysis with MIXMOD, the user supplies classified observations (training observations) and tests observations to be classified. Suppose that we have 2 groups, and for each observation we know the group membership. An estimation of the parameters is obtained by maximizing the log-likelihood.

```
NbLines
     1756
PbDimension
     256
NbCriterion
     1
ListCriterion
     CV
NbNbCluster
     1
ListNbCluster
     3
NbModel
     2
ListModel
     Gaussian_HD_pk_AkjBkQkD
     subDimensionEqual
        10
     Gaussian_HD_pk_AkjBkQkDk
     subDimensionFree
        5
        6
        7
NbStrategy
     1                       [One strategy]
InitType
     USER_PARTITION          [Initialisation by user partition]
InitFile
     ../DATA/USPS_358.part     [Labels (input file) in directory MIXMOD/DATA/]
NbAlgorithm
     1                       [One algorithm]
Algorithm
     M                       [Maximization of log-Likelihood]
PartitionFile
```

```
     ../DATA/USPS_358.part
DataFile
     ../DATA/USPS_358.dat
```

Running this file (*HD_USPS_358_M.test* in directory MIXMOD/TEST) produces estimated parameters (i.e. CVparameter.txt file, see output files (3.5.1.3)).

Now we use the estimated parameters to classify the optional test sample by the MAP algorithm.

```
 NbLines
     17
PbDimension
     256
NbCriterion
     1
ListCriterion
     BIC
NbNbCluster
     1
ListNbCluster
     3
NbModel
     1
ListModel
     Gaussian_HD_pk_AkjBkQkD
     subDimensionEqual
        10
NbStrategy
     1
InitType
     USER
InitFile
     ../DATA/USPS_358.init
NbAlgorithm
     1
Algorithm
     MAP
DataFile
     ../DATA/USPS_358.dat
```

Running this file (*HD_USPS_358_MAP.test* in directory MIXMOD/TEST) produces estimated classes/partition of the remaining data set (i.e. BIClabel.txt, BICpartition.txt files).

3.5.1.3 Others options

All global parameters, maximum numbers of iterations, number of random centers initialisations, overflow, underflow,... are implemented and can be modified in file XEMUtil.h in MIXMOD/SRC/ directory.

3.5.2 Output files

The Output files contain the results of the fit of mixture(s) model(s) for each criterion type, they are created in the working directory.
This is an exhaustive list of the files

- complete.txt,
- numericComplete.txt,
- errorMixmod.txt,
- errorModel.txt,
- Xstandard.txt,
- XnumericStandard.txt,

- Xpartition.txt,
- Xlabel.txt,
- XposteriorProbabilities.txt,
- Xlikelihood.txt,
- XnumericLikelihood.txt,
- Xparameter.txt
- XError.txt,
- CVlabelClassification.txt,
- DCVinfo.txt,
- DCVnumericInfo.txt.

Here X represents all available criteria BIC, ICL, NEC, CV, DCV.

Example of complete output (result of *geyserMini2clusters.xem file*) in the directory MIXMOD/EXAMPLES

```
--------------------------------
     MIXMOD  Complete Output
--------------------------------
Number of samples 272.000000

        Strategy
        --------
        Initial start parameters method RANDOM
        Number of algorithms in the strategy 1
        Algorithm 1
          Type EM
          Stopping rule NBITERATION
          Number of iterations 100
          Set tolerance (xml criterion) 0.000100

                Number of Clusters 2
                ------------------

                        Model Type Gaussian Ellipsoidal Model pk_Lk_C
                        ----------

                        BIC Criterion Value 2322.971927
                        Component 1
                        ---------
                        Mixing proportion 0.642879
                        Mean 4.292206 79.996273
                        Covariance matrix
                                0.145346 0.830099
                                0.830099 40.903602

                        Component 2
                        ---------
                        Mixing proportion 0.357121
                        Mean 2.039681 54.516886
                        Covariance matrix
                                0.098361 0.561759
                                0.561759 27.680967


        Strategy
        --------
        Initial start parameters method RANDOM
        Number of algorithms in the strategy 1
        Algorithm 1
          Type EM
          Stopping rule NBITERATION
          Number of iterations 100
          Set tolerance (xml criterion) 0.000100

                Number of Clusters 3
```

```
------------------

 Model Type Gaussian Ellipsoidal Model pk_Lk_C
 ----------

BIC Criterion Value 2327.088167
 Component 1
 ---------
 Mixing proportion 0.642748
 Mean 4.292448 79.998690
 Covariance matrix
         0.158302 0.756685
         0.756685 36.185614

 Component 2
 ---------
 Mixing proportion 0.140194
 Mean 1.974277 49.482145
 Covariance matrix
         0.042067 0.201082
         0.201082 9.615981

 Component 3
 ---------
 Mixing proportion 0.217058
 Mean 2.082576 57.777045
 Covariance matrix
         0.097293 0.465063
         0.465063 22.239891
```

3.6 MIXMOD C++ library

MIXMOD can be used in another program as a library in the conditions of (*see www.gnu.org/copyleft/gpl.html)* GNU GPL license. We will quickly describe how to use Mixmod as a library in this section but the reference guide is the **Software Documentation**.

3.6.1 How to build a Mixmod project ?

To build a such project, the user must link his program to

- MIXMOD library (in *LIB/MIXMOD* directory)
- NEWMAT library (in *LIB/NEWMAT* directory)

The *MIXMOD* directory is a example of a such program *MIXMOD/SRC* directory contains *main.cpp* which have to be linked to MIXMOD and NEWMAT libraries. To build this project, the user can use'cmake' tools (but he is free to use another way !) by taping the following lines in *MIXMOD directory*

- *camke . (note the '.')*
- *make*
- *make install*

So,

- *libnewmat,*
- *libmixmod,*
- *mixmod executable*

will be created.
So, to build a software using the MIXMOD *library, the user has to replace main.cpp by another main which does not use input and output files. Such mains are available in the MIXMOD/EXAMPLES directory.*

3.6.2 How to create a *main.cpp* ?

This file must

- *create an XEMInput object which contains all input informations,*
- *create an XEMMain object using the XEMInput object,*
- *create an XEMOutput object which contains all output informations,*
- *call the run method of the XEMMain object with the XEMOutput object.*

3.6.3 Examples

3.6.3.1 First example minimal main.cpp

This example is the simplest way to create input, output and xmain objects.
See MIXMOD/EXAMPLES/main_mini.cpp.

```
#include "../LIB/MIXMOD/XEMMain.h"
#include "../LIB/MIXMOD/XEMGaussianData.h"

int main(int argc, char * argv[]){

XEMOutput * output = NULL;
try{
  int nbSample = 272;
  int pbDimension = 2;

  int nbNbCluster = 1;
  int tabNbCluster[1];
  tabNbCluster[0]=2;

  char dataFileName[100];
  strcpy(dataFileName,"../DATA/geyser.dat");
  XEMData * data = new XEMGaussianData(nbSample, pbDimension, dataFileName);

  XEMInput * input = new XEMInput(nbSample, pbDimension, nbNbCluster, tabNbCluster, data);
  input->finalize();

  //XEMMain
  XEMMain xmain(input);

  //Create a XEMOutput object
  //-------------------------
  output = new XEMOutput(input, xmain);

  // xmain run
  xmain.run(output);

  // output printing ...
  cout<<"model paramter "<<endl;
  cout<<"--------------"<<endl;
  XEMModelOutput * modelOutput = output->_tabBestModelOutput[0];
  XEMParameter * param = modelOutput->_param;
  param->edit();

  // delete
  delete data;
  delete input;
}

catch (XEMErrorType errorType){
  XEMError error(errorType);
  error.run();
}

delete output;

return 0;
}
```

So, the user can use the output object to edit informations or to use them in his own objects (see **Software Documentation** *to know more about the XEMOutput class).*

3.6.3.2 How to give specific information to an input object ?

Several methods of XEMInput class can be called to change default values (see **Software Documentation***).*

This other example explain how to specify model types in the input object.
See MIXMOD/EXAMPLES/main_model.cpp.

```
#include "../LIB/MIXMOD/XEMMain.h"
#include "../LIB/MIXMOD/XEMModelType.h"
#include "../LIB/MIXMOD/XEMGaussianData.h"

int main(int argc, char * argv[]){

  XEMOutput * output = NULL;

  try{
    int nbSample = 272;
    int pbDimension = 2;

    int nbNbCluster = 1;
    int tabNbCluster[1];
    tabNbCluster[0]=2;

    char dataFileName[100];
    strcpy(dataFileName,"../DATA/geyser.dat");
    XEMData * data = new XEMGaussianData(nbSample, pbDimension, dataFileName);

    XEMInput * input = new XEMInput(nbSample, pbDimension, nbNbCluster, tabNbCluster, data);

    XEMModelType * tabModelType[28];
    tabModelType[0] = new XEMModelType(Gaussian_p_L_I);
    tabModelType[1] = new XEMModelType(Gaussian_p_Lk_I);
    tabModelType[2] = new XEMModelType(Gaussian_p_L_B);
    tabModelType[3] = new XEMModelType(Gaussian_p_Lk_B);
    tabModelType[4] = new XEMModelType(Gaussian_p_L_Bk);
    tabModelType[5] = new XEMModelType(Gaussian_p_Lk_Bk);
    tabModelType[6] = new XEMModelType(Gaussian_p_L_C);
    tabModelType[7] = new XEMModelType(Gaussian_p_Lk_C);
    tabModelType[8] = new XEMModelType(Gaussian_p_L_D_Ak_D);
    tabModelType[9] = new XEMModelType(Gaussian_p_Lk_D_Ak_D);
    tabModelType[10] = new XEMModelType(Gaussian_p_L_Dk_A_Dk);
    tabModelType[11] = new XEMModelType(Gaussian_p_Lk_Dk_A_Dk);
    tabModelType[12] = new XEMModelType(Gaussian_p_L_Ck);
    tabModelType[13] = new XEMModelType(Gaussian_p_Lk_Ck);
    tabModelType[14] = new XEMModelType(Gaussian_pk_L_I);
    tabModelType[15] = new XEMModelType(Gaussian_pk_Lk_I);
    tabModelType[16] = new XEMModelType(Gaussian_pk_L_B);
    tabModelType[17] = new XEMModelType(Gaussian_pk_Lk_B);
    tabModelType[18] = new XEMModelType(Gaussian_pk_L_Bk);
    tabModelType[19] = new XEMModelType(Gaussian_pk_Lk_Bk);
    tabModelType[20] = new XEMModelType(Gaussian_pk_L_C);
    tabModelType[21] = new XEMModelType(Gaussian_pk_Lk_C);
    tabModelType[22] = new XEMModelType(Gaussian_pk_L_D_Ak_D);
    tabModelType[23] = new XEMModelType(Gaussian_pk_Lk_D_Ak_D);
    tabModelType[24] = new XEMModelType(Gaussian_pk_L_Dk_A_Dk);
    tabModelType[25] = new XEMModelType(Gaussian_pk_Lk_Dk_A_Dk);
    tabModelType[26] = new XEMModelType(Gaussian_pk_L_Ck);
    tabModelType[27] = new XEMModelType(Gaussian_pk_Lk_Ck);

    input->setModelType(28, tabModelType);
    input->finalize();

    //XEMMain
    XEMMain xmain(input);

    //Create a XEMOutput object
    //-------------------------
    output = new XEMOutput(input, xmain);

    // xmain run
    xmain.run(output);

    // output printing ...
    cout<<"best model paramter "<<endl;
    cout<<"-------------------"<<endl;
    XEMModelOutput * modelOutput = output->_tabBestModelOutput[0];
    XEMParameter * param = modelOutput->_param;
    param->edit();
```

```
      // delete
      delete data;
      delete input;
      for (int i=0; i<28; i++){
        delete tabModelType[i];
      }
    }

    catch (XEMErrorType errorType){
      XEMError error(errorType);
      error.run();
    }

  delete output;

  return 0;
}
```

3.6.3.3 Other examples

The MIXMOD/EXAMPLES directory contains some main examples

- *main_criterion.cpp to change default value of the criterion,*
- *main_strategy1.cpp to set the default strategy to the input object,*
- *main_strategy2.cpp to set a new strategy to the input object USER initialization and 200 iterations of SEM algorithm,*
- *main_discriminantAnalysis.cpp an example for discriminant analysis in qualitative case.*

3.7 Illustrative data sets

The data sets have been chosen to illustrate the features of the methods implemented in MIXMOD*. All of them are available in MIXMOD/DATA/ directory, which contains the following data sets:*

Old Faithful Geyser

The file geyser.dat contains 272 observations from the Old Faithful Geyser in the Yellowstone National Park. Each observation consists of two measurements: the duration (in minutes) of the eruption and the waiting time (in minutes) to the next eruption. Old Faithful erupts more frequently than any other big geyser, although it is not the largest nor the most regular geyser in the park. Its average interval between two eruptions is about 76 minutes, varying from 45 - 110 minutes. An eruption lasts from 1.1/2 to 5 minutes, expels 3,700 - 8,400 gallons (14,000 - 32,000 liters) of boiling water, and reaches heights of 106 - 184 feet (30 - 55m). It was named for its consistent performance by members of the Washburn Expedition in 1870. Old Faithful is still as spectacular and predictable as it was a century ago.

Diabetes in Pima Indians

The diagnostic, binary-valued variable investigated is whether the patient shows signs of diabetes according to the World Health Organization criterion (i.e. if the 2 hour post-load plasma glucose was at least 200 mg/dl at any survey examination or if found during routine medical care). The population of women who were at least 21 years old, lives near Phoenix, Arizona, USA.

It appears that this data set was incorrectly recorded. Many attributes have missing values and these have been encoded with the numerical value 0. This applies to something like 5 of the eight attributes. Besides, the classification variable is coded as 1=healthy and 2=diabetic (i.e. interchanged wrt. UCI dataset).

Table 3.5: Diabetes in Pima Indians

Column	Type	Description
1	metric	number of pregnancies
2	metric	plasma glucose concentration in an oral glucose tolerance test
3	metric	diastolic blood pressure (mm Hg)
4	metric	triceps skin fold thickness (mm)
5	metric	serum insulin (μ U/ml)
6	metric	body mass index (weight in kg/(height in m)^2)
7	metric	diabetes pedigree function
8	metric	age in years

Haemophilia

The Haemophilia data set haemophilia.dat, analyzed by Basford and McLachlan (1985), consists of two bivariate components, and its analysis illustrates the caution needed to be exercised in practice when fitting mixture models.

Enzyme

The data set enzyme.dat concerns the distribution of enzymatic activity in the blood, for an enzyme involved in the metabolism of carcinogenic substances, among of group of 245 unrelated individuals. The interest here is to identify subgroups of slow or fast metabolisers as a marker of genetic polymorphism in the general population. This data set has been analyzed by Bechtel et al. *(1993), who identified a mixture of 2 skewed distributions using maximum likelihood technics implemented in the program* Skumix *of Maclean* et al *(1976).*

Acidity

The data set acidity.dat concerns an acidity index measured in a sample of 155 lakes in the North-eastern United States and has been previously analyzed as a mixture of Gaussian distributions on the log scale by Crawford et al.*(1992, 1994).*

U.S. Companies

The data file uscomp.dat contains measurements for 79 U.S. companies out of the top 500.

Table 3.6: U.S. Companies

Column	Type	Description
1	text	company
2	metric	assets
3	metric	sales
4	metric	market value
5	metric	profits
6	metric	cash flow
7	metric	employees
8	text	branch

Swiss Bank Notes

The data file bank.dat contains measurements on 100 genuine and 100 forged old Swiss 1000 franc bills from Flury and Riedwyl (1988). The first 100 rows correspond to the genuine bills and the second 100 rows to the forged data.

Table 3.7: Swiss Bank Notes

Column	Type	Description
1	metric	length of the bill
2	metric	height of the bill, measured on the left
3	metric	height of the bill, measured on the right
4	metric	distance of inner frame to the lower border
5	metric	distance of inner frame to the upper border
6	metric	length of the diagonal