Akademie věd České republiky
Ústav teorie informace a automatizace

Academy of Sciences of the Czech Republic
Institute of Information Theory and Automation

RESEARCH REPORT

Kárný M., Andrýsek J., Nedoma P., Böhm J., Guy T.V.

On Generalized Factors in Mixture Learning and Prediction

No. 2095 December 2003

Projects GA ČR 102/03/0049, 102/03/P010 and AV ČR S1075351, S1075102

ÚTIA AV ČR, P. O. Box 18, 182 08 Prague, Czech Republic
http://utia.cas.cz, Fax: (+420) 26605 2068, E-mail: utia@utia.cas.cz

This report constitutes a non-referred software description. Opinions and conclusions expressed in this report are those of the author(s) and do not necessarily represent the views of the Institute.

Acknowledgement: This project was supported by GA ČR 102/03/0049, 102/03/P010 and AV ČR S1075351, S1075102

Contents

1 Addressed problem

Normal ARX (autoregressive with external variables) factors form basic building blocks for constructing components creating finite (probabilistic) mixtures. Finite mixtures [1] serve for efficient description of non-linear, non-Gaussian systems. Their applicability can be substantially enhanced when using generalized ARX models in the sense introduced in [2]. This generalization deals essentially with normal ARX model defined on transformed data: both the regression vector and regressand are transformed by a known, unknown-parameter-free mapping. This simple idea allows to cover ordinary data scaling, work with log-normal versions of ARX factors, continuous convolution-based models [3, 4, 5] etc.

This paper tries to find out structure of necessary evaluations that support the use of the generalized ARX factors in all tasks treated by the software system Mixtools [6], i.e. mixture estimation, data prediction as well as design and computing of optimal actions.

2 Problem solution

The following notation is adopted throughout the text.

Symbol	**Meaning**
$\equiv$	equality by definition
x^*	a set of x-values
$\mathring{x}$	the number of elements in a vector x or in a finite x^*
$f(\cdot\|\cdot)$	probability density functions (pdf)
$d(t)$	sequence $(d_1, \ldots, d_t)$
t	discrete-time, always the last subscript after ;
$'$	transposition, the default vector orientation is row one
${}^{\lfloor a}B$	a non-numerical index a of a variable B
Ψ	raw data vector containing measured data d_t and their delayed values
Ψ	the richest raw data vector: all mixture components select their data vectors Ψ from it
$\tilde{\Psi}$	source data vector, i.e. filtered but non-permuted raw data vector Ψ
${}^{\lfloor\pi}\Psi$	filtered data vector obtained by permuting $\tilde{\Psi}$
$\|J\|$	absolute value of the determinant of the matrix J

The pdfs are distinguished by the identifiers in their arguments. No formal distinction is made between random variable, its realization and an argument of a pdf. The correct meaning follows from the context. We use the *chain rule* [2] for pdfs

$$f(a, b|c) = f(a|b, c)f(b|c) \tag{1}$$

and the formula for evaluation of pdfs of transformed variables. Let $\alpha = T(\beta, \gamma)$, $\mathring{\alpha} = \mathring{\beta}$, such that for almost all (β, γ) the Jacobian

$$J(\beta, \gamma) \equiv \left| \frac{\partial T(\beta, \gamma)}{\partial \beta} \right| \tag{2}$$

is non-zero. Then,

$$f(\beta|\gamma) = \frac{f_\alpha(T(\beta,\gamma)|\gamma)}{J(\beta,\gamma)} \tag{3}$$

where $f_\alpha(\alpha|\gamma)$ is the original pdf of α conditioned on γ.

2.1 Factorized form of normal components

Let us consider the *source data vector* $\tilde{\Psi}$ containing filtered both regressands and regression vector entering a normal component. Its entries are assumed to have a fixed meaning. Let us split its entries on those that are modelled, distinguished by the left upper subscript ${}^{\lfloor m}$, and non-modelled ones, marked by ${}^{\lfloor n}$. Let the mapping π permute indices $1,\ldots,\mathring{\tilde{\Psi}}$ of $\tilde{\Psi}$ so that its modelled entries are placed to the beginning of the *(permuted) filtered data vector* ${}^{\lfloor\pi}\Psi$, i.e.

$${}^{\lfloor\pi}\Psi \;\equiv\; \left[{}^{\lfloor m\pi}\Psi,\, {}^{\lfloor n\pi}\Psi\right], \quad {}^{\lfloor\pi}\Psi_j \equiv \tilde{\Psi}_{\pi(j)},\; j=1,\ldots,\mathring{\Psi} \equiv \mathring{\tilde{\Psi}}. \tag{4}$$

The part ${}^{\lfloor m\pi}\Psi$ contains data whose dependence on ${}^{\lfloor n\pi}\Psi$ is modelled by the *normal component*, i.e. the normal multivariate pdf written in the factorized version

$$f\left({}^{\lfloor m\pi}\Psi|\,{}^{\lfloor n\pi}\Psi,\Theta\right) = \prod_{i=1}^{{}^{\lfloor m\pi}\mathring{\Psi}} \underbrace{\mathcal{N}_{{}^{\lfloor\pi}\Psi_i}\left(\theta_i\,{}^{\lfloor\pi}\psi_i', r_i\right)}_{\text{normal factor}}, \tag{5}$$

where *regression vector* ${}^{\lfloor\pi}\psi_i$ is a sub-selection of $\left[{}^{\lfloor\pi}\Psi_{i+1},\ldots,{}^{\lfloor\pi}\Psi_{{}^{\lfloor m\pi}\mathring{\Psi}},\,{}^{\lfloor n\pi}\Psi\right]$, θ_i is the corresponding vector of *regression coefficients* and r_i *noise variance*. The unknown parameters of the component are $\Theta = \{\theta_i, r_i\}_{i=1}^{{}^{\lfloor m\pi}\mathring{\Psi}}$.

Obviously, a permutation of the *modelled part has to appear in all considered components*, i.e. ${}^{\lfloor m\pi}\mathring{\Psi}$ has to be common to them and ${}^{\lfloor m\pi}\Psi$ in a component has to be permutation (possibly trivial one) of ${}^{\lfloor m\pi}\Psi$ in another component. This makes a clear restriction on allowed permutations π and on the source data vector $\tilde{\Psi}$. On the other hand, the non-modelled part ${}^{\lfloor n\pi}\Psi$ can be component specific.

For a given source data vector $\tilde{\Psi}$, the *structure of the component* is described by the permutation π and by the dimension ${}^{\lfloor m\pi}\mathring{\Psi}$.

The i-th factor models i-th entry ${}^{\lfloor\pi}\Psi_i$ of ${}^{\lfloor\pi}\Psi$. Its structure is determined by the mapping $s_i(\cdot)$ selecting entries in ${}^{\lfloor\pi}\Psi_j$, $j \in \left\{i+1,\ldots,\mathring{\Psi}\right\}$ used in the regression vector ${}^{\lfloor\pi}\psi_i$, i.e.

$${}^{\lfloor\pi}\psi_{ik} = {}^{\lfloor\pi}\Psi_{s_i(k)} = \tilde{\Psi}_{\pi(s_i(k))},\; k=1,\ldots,\mathring{s}_i, s_i(k) \in \left\{i+1,\ldots,\mathring{\Psi}\right\}. \tag{6}$$

Thus, the *structure of the factor* is described by the pair i, s_i.

2.2 Filters

The source data vector $\tilde{\Psi}$ is assumed to result from a sort of filtering, i.e. from a transformation T of *raw data vector* Ψ. Obviously, its permuted version ${}^{\lfloor\pi}\Psi$ is the result of

transformation ${}^{\lfloor\pi}T$ with entries ${}^{\lfloor\pi}T_i(\Psi) = T_{\pi(i)}(\Psi)$. Let's split the transformation ${}^{\lfloor\pi}T$ into the parts ${}^{\lfloor m\pi}T(\Psi)$, ${}^{\lfloor n\pi}T(\Psi)$ according to the following schema.

$$ {}^{\lfloor\pi}\Psi \equiv \left[{}^{\lfloor m\pi}\Psi,\ {}^{\lfloor n\pi}\Psi \right] \equiv {}^{\lfloor\pi}T(\Psi) \equiv \left[{}^{\lfloor m\pi}T(\Psi),\ {}^{\lfloor n\pi}T(\Psi) \right]. \tag{7} $$

The *raw data vector* Ψ forming argument of this transformation is selected from the *richest raw data vector* $\bar{\Psi}_t$ made of the raw data $d(t)$ complemented by the state vector in phase form, i.e.

$$ \bar{\Psi}_t = [d_t, d_{t-1}, \ldots, d_{t-\partial}, o], \tag{8} $$

where $t \in t^* \equiv \{1, \ldots, \mathring{t}\}$ denotes discrete time and d_t is the *data record* measured at time instance labelled by t and o is value of offset.

The *structure of the richest raw data vector* $\bar{\Psi}$ is determined by the list of data channels entering the data record d_t, by the order $\partial \geq 0$ and by indicator $o \neq 0$ of the presence of unknown offset $o\times$unknown_constant.

The mapping S determining Ψ from $\bar{\Psi}$ has to be time-invariant so that for any $t \in t^*$

$$ \Psi_i = \bar{\Psi}_{S(i)},\ S(i) \in \{1, \ldots, (\mathring{d}+1)\partial + 1\},\ i = 1, \ldots, \mathring{\Psi}. \tag{9} $$

The mapping S may choose any entry from $\bar{\Psi}$ at most once. It should not introduce delay in processing so that at least some items from the newest data record d_t available at time t have to be selected by S.

We want to deal with causal (in informational sense) models. This implies that the *the non-modelled part of the transformed data vector* ${}^{\lfloor n\pi}T(\Psi)$ *must not depend on* d_t. We also want to model modelled part ${}^{\lfloor m}\Psi$ of the raw data vector. ${}^{\lfloor m}\Psi$ has to include d_t but it may be wider. The transformation ${}^{\lfloor m\pi}T$ must be regular, i.e. it has to have almost everywhere non-zero Jacobian. The formulas (3) (5) imply that

$$ f({}^{\lfloor m}\Psi |\, {}^{\lfloor n\pi}\Psi,\ {}^{\lfloor n}\Psi, \Theta) = \left| \underbrace{\frac{\partial\, {}^{\lfloor m\pi}T\,(\Psi)}{\partial\, {}^{\lfloor m}\Psi}}_{\text{Jacobi matrix related to } {}^{\lfloor m}\Psi} \right|^{-1} \prod_{i=1}^{{}^{\lfloor m\pi}\mathring{\Psi}} \mathcal{N}_{{}^{\lfloor m\pi}T_i(\Psi)} \left(\theta_i\, {}^{\lfloor\pi}\psi_i', r_i \right). \tag{10} $$

This formula implies that the advantageous *factorized version of the component can be preserved only when the Jacobi matrix related to the modelled part is upper triangular one with i-th diagonal entry dependent on* ${}^{\lfloor m}\Psi_i$*, on un-modelled* ${}^{\lfloor n}\Psi$ *and* ${}^{\lfloor\pi}\psi$.

Whenever we require preservation of this property even for different permutations $\pi(\cdot)$, the Jacobi matrix related to the modelled part has to be diagonal one with $\partial\, {}^{\lfloor m\pi}T_i(\Psi)/\partial\, {}^{\lfloor m}\Psi_i$ depending on ${}^{\lfloor m}\Psi_i$ and ${}^{\lfloor n}\Psi$.

There is much more freedom in filtering of the non-modelled part. There, various channels can be combined and dimensions need not be preserved: both ${}^{\lfloor n\pi}\mathring{\Psi} \geq {}^{\lfloor n}\mathring{\Psi}$ and ${}^{\lfloor n\pi}\mathring{\Psi} \leq {}^{\lfloor n}\mathring{\Psi}$ may hold.

2.3 Parameter estimation on filtered data

The possibility to use the standard estimation of the ARX model is the main practical advantage of the generalized ARX model. It can be run within the class of Gauss-inverse-

Wishart distributions with pdfs

$$GiW_{\Theta_i}(V,\nu) \equiv GiW_{\theta_i,r_i}(V_i,\nu_i) \equiv \frac{r_i^{-0.5\left(\nu_i+\,\lfloor^{\pi}\mathring{\psi}_i+2\right)}}{\mathcal{I}(V_i,\nu_i)} \exp\left\{-\frac{1}{2r_i}\mathrm{tr}\left(V_i\left[-1,\theta_i\right]'\left[-1,\theta_i\right]\right)\right\} \tag{11}$$

and statistics updated according to the formulas

$$\lfloor^{new}V_i = V_i + \left[\,\lfloor^{\pi}\Psi_i,\ \lfloor^{\pi}\psi_i\right]'\left[\,\lfloor^{\pi}\Psi_i,\ \lfloor^{\pi}\psi_i\right],\ \ \lfloor^{new}\nu_i = \nu_i+1,\ i=1,\ldots,\ \lfloor^{m\pi}\mathring{\Psi}. \tag{12}$$

The normalization integral $\mathcal{I}(V_i,\nu_i)$ is the most effectively expressed when we deal with $L'DL$ decomposition of the extended information matrix $V_i = L_i'D_iL_i$. Here, L_i is lower triangular matrix with unit diagonal, D_i diagonal matrix with non-negative entries. It holds

$$\mathcal{I}(V_i,\nu_i) \equiv \mathcal{I}(L_i,D_i,\nu_i) = \Gamma(0.5\nu_i)\,\lfloor^{d}D_i^{-0.5\nu_i}\left|\,\lfloor^{\psi}D_i\right|^{-0.5} 2^{0.5\nu_i}(2\pi)^{0.5\,\lfloor^{\pi}\mathring{\psi}_i}, \tag{13}$$

where $\lfloor^{d}D_i$ is the first diagonal entry of D_i and $\lfloor^{\psi}D_i$ is gained from D_i by cancelling the first column and row. Note that $\left|\,\lfloor^{\psi}D_i\right|$ is simply product of diagonal entries of D_i with the first one omitted.

2.4 Prediction on filtered data

Predictive pdf of individual filtered data $\lfloor^{\pi}\Psi_i$ can be expressed in terms of the normalization integral (13) as follows

$$f\left(\lfloor^{\pi}\Psi_i|\,\lfloor^{\pi}\psi_i\right) = \frac{\mathcal{I}\left(V_i+\left[\,\lfloor^{\pi}\Psi_i,\ \lfloor^{\pi}\psi_i\right]'\left[\,\lfloor^{\pi}\Psi_i,\ \lfloor^{\pi}\psi_i\right],\nu_i+1\right)}{\mathcal{I}(V_i,\nu_i)},\ i=1,\ldots,\ \lfloor^{m\pi}\mathring{\Psi}. \tag{14}$$

Let us introduce *mixed regression vector* $\tilde{\psi}$ consisting of the modelled raw data Ψ and non-modelled filtered data $\lfloor^{n\pi}\Psi$

$$\tilde{\psi}_i \equiv \left[\Psi_{i+1},\ldots,\Psi_{\,\lfloor^{m}\mathring{\Psi}},\ \lfloor^{n\pi}\Psi\right],\ i=1,\ldots,\ \lfloor^{m}\mathring{\Psi}. \tag{15}$$

With the upper triangular Jacobi matrix, the prediction of the original normalized data reads, $i = \lfloor^{m\pi}\mathring{\Psi},\ldots,1$,

$$f(\Psi_i|\tilde{\psi}_i) = \left[\frac{\partial\,\lfloor^{\pi}T_i(\Psi_i,\tilde{\psi}_i)}{\partial\Psi_i}\right]^{-1}\frac{\mathcal{I}\left(V_i+\left[\,\lfloor^{\pi}T(\Psi_i,\tilde{\psi}_i),\tilde{\psi}_i\right]'\left[\,\lfloor^{\pi}T(\Psi_i,\tilde{\psi}_i),\tilde{\psi}_i\right],nu_i+1\right)}{\mathcal{I}(V_i,\nu_i)}. \tag{16}$$

The prediction order stresses that we have to start from the last predicted entry of the modelled raw data vector $\lfloor^{m}\Psi$.

Often, the predictions are evaluated for measured values. Then, the value of $\lfloor^{\pi}\Psi_i \equiv \lfloor^{\pi}T_i(\Psi_i,\tilde{\psi}_i)$ enters the prediction formula and thus the presence of the Jacobian factor $\left[\frac{\partial\,\lfloor^{\pi}T_i(\Psi_i,\tilde{\psi}_i)}{\partial\Psi_i}\right]$ computed at this $\lfloor^{\pi}\Psi_i$ is the only difference encountered between predictions with ARX and generalized ARX models.

It has immediate practical consequence: *the current structure estimation can be used without a change if the diagonal Jacobian depending only on the corresponding data-vector entry is considered.* Otherwise, attempts to cancel some entries from $\tilde{\psi}_i$ influence the values of ${}^{\lfloor\pi}T_i(\Psi_i, \tilde{\psi}_i)$ and the accumulated statistics correlating in V_i this entry of the data vector Ψ with entries of $\tilde{\psi}_i$ have to be recomputed. This is the decisive argument for *the use of transformations with diagonal Jacobians depending on the corresponding entry of Ψ only.*

2.5 Action design and generating with filtered data

The designed action form a part of d_t and thus, with the adopted "diagonal" Ts, they are mapped in one-to-one fashion on the filtered modelled data ${}^{\lfloor m\pi}\Psi$. Thus, the *design can be performed fully in terms of filtered data* ${}^{\lfloor\pi}\Psi$ if the design target is expressed in them, too. In the adopted fully probabilistic design, the optimal strategy is normal and written in a factorized version mimic to (5). Thus, *the transformation onto the raw (original) actions is made exactly as it is done with prediction.*

One additional aspect arises. The implemented design procedures rely on the shifted phase form of processed data. They can be used without a change whenever this property is preserved for filtered data. The following important filters meet this property:

- scaling,
- static, non-linear, invertible transformations applied to all delayed variables,
- entry-wise applied linear filters,
- dynamic, non-linear transformations applied to all delayed variables and invertible for the newest value,
- spline based filters.

3 Software implementation

Software representation of the above approach to *generalized ARX factors* is discussed here. The reader is supposed to be familiar with principles of Mixtools processing.

3.1 Constructors

Each component is a structure. It has a field *comstr* referred to as *component structure* (S in (9)). It describes the richest raw data vector $\bar{\Psi}$. The component structure has two rows. Each column has the meaning of a channel and a time delay. The pair $[0; o]$ introduces *offset* of the value o, see (8).

The factor structure *str* (s_i in(6)) and modelled channel *ychn* are described relatively to *comstr*.

The relevant constructors are summarized.

```
Fac  = facarx(ychn, str)                % ARX factor
Fac  = facarxls(ychn, str)              % ARX LS factor
Com  = comarx(comstr, Facs)             % ARX or ARX LS component
Com  = matarxls(ychns, str, comstr)     % matrix ARX LS component
Mix  = mixconst(Coms, dfcs)             % mixture of any type
Flt  = fltconst(type, filter arguments % filter
```

The matrix ARX component is not discussed here. It has only an auxiliary meaning for data input and interpretation of results. The arguments *Facs* is a cell vector of individual factors, the argument *Coms* a cell vector of individual components.

As an example, we have the component structure:

```
comstr = [2 1 1 1 2 2; 0 0 1 2 1 2]      % component structure
comstr =
     2     1     1     1     2     2
     0     0     1     2     1     2
```

Two dynamic factors are build. The first one is a model for the *comstr* structure item [1;0]. It means, that its relative position in *comstr* is 2. We speak about the *modelled channel* 2 in this sense.

```
ychn = [1; 0];                          % modelled channel
str  = [2 1 1 2 2; 0 1 2 1 2];          % factor structure
rstr = str2str(comstr, str)             % relative factor structure
rstr =
     1     3     4     5     6
ychn = str2str(comstr, ychn)            % modelled channel
ychn =
     2
Fac1 = facarx(ychn, str)                % ARX factor


Fac1 =
    ychn: 2                             -> modelled item of comstr
     str: [1 3 4 5 6]                   -> factor structure
     dfm: 1                             -> degrees of freedom
    type: 1                             -> coded factor type
      LD: [6x6 double]                  -> L'Lt of information matrix
```

The second factor is build similarly.

```
ychn = 1;                               % modelled channel
str  = [3 4 5 6];                       % factor structure
Fac2 = facarxls(ychn, str);             % build matrix ARX LS factor
```

An ARX component is build by the function *comarx*.

```
Com  = comarx(comstr, {Fac1 Fac2})      % ARX component
Com =
        Facs: {[1x1 struct]  [1x1 struct]}   -> component factors
      comstr: [2x6 double]                   -> component structure
        Flts: {[0]  [0]  [0]  [0]  [0]  [0]}   -> array of filters
        type: 11                             -> coded component type
     rawdata: [0 0 0 0 0 0]                  -> workspace for Ψ
    datavect: [0 0 0 0 0 0]                  -> workspace for Ψ̃
```

A mixture is build by *mixconst*:

```
dfcs = [22 11];                          % degrees of freedom
Mix  = mixconst({Com Com}, dfcs)         % ARX mixture
Mix =
    Coms: {[1x1 struct]  [1x1 struct]}   -> mixture components
    dfcs: [22 11]                        -> degrees of freedom
    mmod: 2                              -> number of modelled channels
    type: 21                             -> mixture type (ARX)
```

3.2 Filters

Filters are realized as structures. A filter contains a coded filter type and all internal states needed for the data transformation made by it. The filters are build by the function $fltconst$ and the data transformation is done by the function $filters$. *Composed filters* are cell vectors of individual filters - structures.

The filters on modelled channels must ensure one-to-one data transformation between raw data and modelled filtered data. The Jacobian of the transformation is needed in learning, backwards transformation by prediction.

There is no limitation on the filters on non-modelled part of data vector.

The filter constructor has the form:

```
Flt = fltconst(filter_type, filter_arguments)
```

The data filtering on modelled channels is done in three modes. The arguments are:

$yraw$	raw data item
$yflt$	filtered data item
dy	logarithm of diagonal Jacobian-matrix entry of the transformation $\Psi -> \tilde{\Psi}$
$mode$	coded mode of operation

Individual modes of filtering are:

```
[yflt, dy]  = filters(Flt, yraw, 1)     % used in trial step of estimation
% filter state is not updated
[yflt, Flt] = filters(Flt, yraw, 2)     % update filter states
[yraw, Flt] = filters(Flt, yflt, 3)     % inverse used in prediction
```

An example folows. Data scaling with filters is discussed. We have a data sample. Mean and standard deviation of $DATA$ is used for building of filters for each channel.

```
m = mean(DATA')
m =
   -0.1723   -0.1559
s = std (DATA')
s =
    0.6925    0.8284
```

The corresponding filter for scalling data of the 1st channel is build. +

```
Flt = fltconst(1, -m(1), 1/s(1))        % filter for the 1st channel
Flt =
     add: 0.1723
     mul: 1.4441
    type: 1
```

The filter scales the raw data item by adding $Flt.add$ and multiplying the result by $Flt.mul$. The Jacobian of the transformation is $Flt.mul$. This value is returned in logarithm.

The filters are set either "manually" or or using auxiliary functions.

The scaling and other filters are supported by the function $fltpre$. The preprocessing list is used as input.

```
pre  = {'scale', []};                   % preprocessing list
Mix0 = fltpre(Mix0, pre);               % set filters for all channels
```

Here, the function $fltpre$ sets filters for each *comstr* element with the exception of offset.

3.3 Learning with ARX mixtures

Learning with an ARX component is done in several steps:

- the component field *rawdata* (i.e. Ψ) is filled from data sample $DATA$ using *comstr* (S in (9));
- corresponding filters recorded in $Flts$ are called, the transformed data elements are copied to *datavect*. If the filter contain a numerical value, no filtering occurs. In this case, the numerical value is only a placeholder;
- the factors $Facs$ are updated. The factors gets values from *datavect*.

Example follows, the results are displayed in Fig. 1. The data sample is generated.

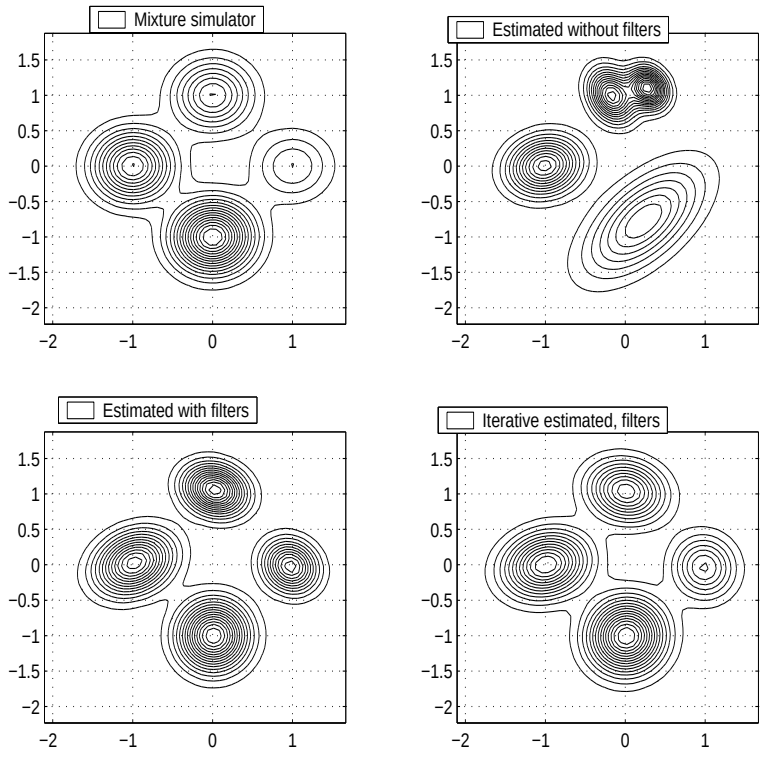

Figure 1: Case study: mixture estimation

```
ndat = 1000;                          % size of data sample
cove = [0.1 0.01; 0.01 0.1];          % component noise variance
ncom = 4;                             % number of components
Sim = statsim(ndat, ncom, cove);      % data sample
...  plot
```

The initial mixture is build and estimated without filters and forgetting.

```
Mix0 = genmixe(ncom);                 % build initial mixture
frg  = 1;                             % without forgetting
Mix  = mixestim(Mix0, frg, ndat);     % mixture estimation
...  plot
```

The scaling is specified, relevant filters set and the intial mixture is estimated and displayed.

```
pre  = {'scale', []};                 % preprocessing requirement
Mix0 = fltpre(Mix0, pre);             % set filters
Mix  = mixestim(Mix0, frg, ndat);     % mixture estimation
...  plot
```

The estimation is done in 10 iterations.

```
niter = 10;
Mix = mixest(Mix0, frg, ndat, niter);  % iterative quasi-Bayes estimation
...  plot
```

3.4 Mixture projection and prediction

There are two basic operations related to prediction with normal mixture:

- *mixture projection*
  means marginalization and conditioning. The result of these operations is referred to as *mixture projector*;
- *mixture prediction*
  arises from the mixture projection by substitution of a specific regression vector into it. The result is a static mixture referred to as *mixture predictor*.

3.4.1 Mixture projection

The projection converts mixture estimator into mixture *projector*. It provides description of Student pdf mostly approximated by normal pdf. The projection is conditional pdf on a set of modelled channels referred to as *predicted channels* and conditioned by values of another set of modelled channels referred to as *channels in condition*. The predictor can be re-built for a new selection of these channels.

The mixture projection is done by the function *mix*2*pro*:

`pMix = mix2pro(Mix, pchns, cchns)` *build mixture projector*

The argument together with defaults are:

argument	*meaning*	*defaults*
Mix	mixture estimator or projector	must be specified
pchns	predicted channels	all modelled channels
cchns	channels in condition	no channels in condition

3.4.2 Prediction with a mixture

Prediction with a mixture is derived from a mixture projector by substitution of a regression vector into it.

The regression vector can be specified for several *comstr* items only or can be missing. In the case, the data needed are extracted from the signal database. Often, only values of zero-delayed structure items are specified and the history items are loaded from database (DATA).

The regression vector is specified as a raw regression vector. It consists of 3 rows. The first one contains values, the second and third ones contains the corresponding structure items.

The third row can be omitted - in the case, row of zeros is internally substituted. Moreover, the second row can be omitted if there is only one channel in condition. The order of the regression vector is not important. Note that the specification or regression vector is in "raw data" style.

The mixture prediction is done by:

`lhs = profix(pMix, psi0)`	*prediction*

where

argument	*meaning*
lhs	output argument(s), see below
$pMix$	mixture projector
$psi0$	the data vector

The output arguments can have the form:

$pMix$
$[Eths, coves, alphas]$
$[pMix, weights]$
$[Eths, coves, alphas, weights]$

where

argument	*meaning*
$Eths$	vector or cell vector of means of individual components
$coves$	vector or cell vector of noise covariances
$alphas$	weights of individual components corresponding to normalized dfcs modified due to conditioning
$weights$	data-dependent approximate component weights

The prediction can be done for a number of processing steps ahead.

`lhs = profixn(Mix, psi0, nsteps)`	*prediction nsteps ahead*

Join mixture projection and prediction done by one function can be made:

`lhs = mixpro(Mix, pchns, cchns, psi0)`	*mixture prediction*

3.4.3 Prediction example

The mixture projection is documented by a simulated example. A static ARX LS mixture is built and data cluster displayed in Fig. 2, subplot 1.

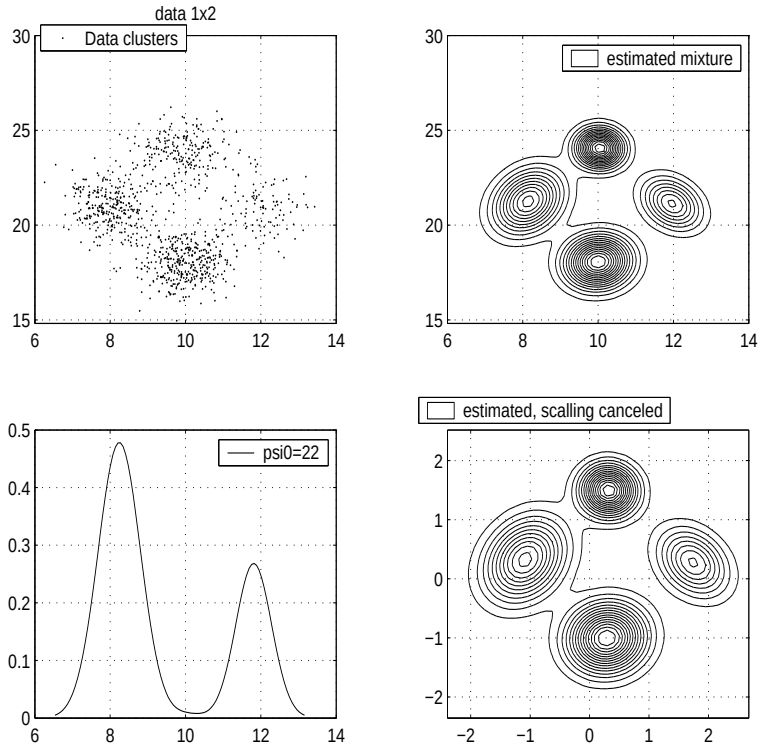

Figure 2: Mixture prediction

slide3.m

```
cove = ltdl([0.1 0.01; 0.01 0.1]);      % common component noise covari-
ance
ncom = 4;                               % number of components
Mix  = statsim(0, ncom, cove);          % build ARX LS mixture
...  plot ...
```

Then, the mixture projector is build:

```
pchns = 1;                              % predicted channel
cchns = 2;                              % channel in condition
pMix  = mix2pro(Mix, pchns, cchns);     % build mixture predictor
pMix.states
ans =
      predicted: 1
    incondition: 2
```

The result is p-ARX LS mixture. The mixture states record the projection done.

The mixture prediction is built and displayed in Fig. 2 subplot 2. The prediction is static p-matrix ARX LS mixture. The degrees of freedom of components are transformed due to conditioning. This form cannot be rebuild for another conditioning and marginalization. The data vector can be supplied in the forms shown below.

```
psi0  = 0.5;                            % zero-delayed data vector
psi0  = [0.5 2]';                       % other form of psi0
psi0  = [0.5 2 0]';                     % the complete psi0 specification
pMix1 = profix(pMix, psi0);             % get prediction
... plot ...
```

The filters are disconnected and the mixture is displayed in Fig. 2 subplot 4. Note that internally the mixture is held in scalled data but, any projection displays it in the user data levels.

```
for i=1:4
    Mix1.Coms{i}.Flts{1} = 0;
    Mix1.Coms{i}.Flts{2} = 0;
end
... plot mixture ...
```

4 Unsolved problem

The presented implementation of generalized factors does not solve systematically how to handle admissible non-diagonal relationships among non-modelled channels. This case is important as it includes the data transformation that must be done during estimation of important factors. ARMAX models represent a prominent example of this type. This case will be solved later on.

References

[1] D.M. Titterington, A.F.M. Smith, and U.E. Makov, *Statistical Analysis of Finite Mixtures*, John Wiley & Sons, Chichester, New York, Brisbane, Toronto, Singapore, 1985, ISBN 0 471 90763 4.

[2] V. Peterka, "Bayesian system identification", in *Trends and Progress in System Identification*, P. Eykhoff, Ed., pp. 239–304. Pergamon Press, Oxford, 1981.

[3] T.V. Guy and M. Kárný, "Spline-based hybrid adaptive controller", in *Modelling, Identification and Control. Proceedings*, M. H. Hamza, Ed., Anaheim, February 1997, pp. 118–122, IASTED Acta Press.

[4] T. V. Guy, M. Kárný, and J. Böhm, "Linear adaptive controller based on smoothing noisy data algorithm", in *European Control Conference. ECC '99. (CD-ROM)*, Karlsruhe, August 1999, pp. –, VDI/VDE GMA.

[5] Guy T. and Karny M., "Possible way of improving the quality of modelling for adaptive control", in *Computer Aided Control System Design*, Gray J. O., Ed., Amsterdam, September 2001, pp. 179–185, Elsevier.

[6] P. Nedoma, M. Kárný, I. Nagy, and M. Valečková, "Mixtools. MATLAB Toolbox for Mixtures", Tech. Rep. 1995, ÚTIA AV ČR, Praha, 2000.