Akademie věd České republiky
Ústav teorie informace a automatizace

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

RESEARCH REPORT

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

Mixtures with Generalized Factors
User's Guide

No. 2155 March 2005

Project GA ČR 102/03/0049, AV ČR S1075351

ÚTIA AV ČR, P. O. Box 18, 182 08 Prague, Czech Republic
Telex: 122018 atom c, Fax: (+420)(2)6884 903
E-mail: utia@utia.cas.cz

Contents

Chapter 1

Addressed problem

Normal ARX (autoregressive with external variables) factors form basic building blocks for constructing components creating finite (probabilistic) mixtures. Finite mixtures [?] 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 [1]. 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 [2, 3, 4] 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 [5], i.e. mixture estimation, data prediction as well as design and computing of optimal actions.

1.1 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^*
$f(\cdot\vert\cdot)$	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
$\tilde{\Psi}$, ${}^{\lfloor \pi}\Psi$, $\bar{\Psi}$, Ψ	source, filtered, richest raw and raw data vectors, respectively

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 need the *chain rule* [1] for pdfs

$$f(a,b|c) = f(a|b,c)f(b|c) \tag{1.1}$$

as well as 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

$$J(\beta,\gamma) \equiv \left|\frac{\partial T(\beta,\gamma)}{\partial \beta}\right| \tag{1.2}$$

is non-zero. Then,

$$f(\beta|\gamma) = J^{-1}(\beta,\gamma) f_\alpha(T(\beta,\gamma)|\gamma), \tag{1.3}$$

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

1.1.1 Factorized form of normal components

Let us consider the data vector $\tilde{\Psi}$ 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 data vector ${}^{\lfloor \pi}\Psi$, i.e.

$${}^{\lfloor \pi}\Psi \quad\equiv\quad \left[{}^{\lfloor m\pi}\Psi,\ {}^{\lfloor n\pi}\Psi\right], \quad {}^{\lfloor \pi}\Psi_j \equiv \tilde{\Psi}_{\pi(j)},\ j = 1,\ldots,\mathring{\Psi} \equiv \mathring{\tilde{\Psi}}. \tag{1.4}$$

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

$$f\left({}^{\lfloor m\pi}\Psi \middle| {}^{\lfloor n\pi}\Psi, \Theta\right) = \prod_{i=1}^{{}^{\lfloor m\pi}\mathring{\Psi}} \underbrace{\mathcal{N}_{{}^{\lfloor \pi}\Psi_i}\left(\theta_i\, {}^{\lfloor \pi}\psi_i', r_i\right)}_{\text{normal factor}}, \tag{1.5}$$

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 πs and on the source data vector $\tilde{\Psi}$. On the other hand, the non-modelled part ${}^{\lfloor n}\Psi$ 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.

$${}^{\lfloor \pi}\psi_{ik} = {}^{\lfloor \pi}\Psi_{s_i(k)} = \tilde{\Psi}_{\pi(s_i(k))},\ k = 1,\ldots,\mathring{s}_i, s_i(k) \in \left\{i+1,\ldots,\mathring{\Psi}\right\}. \tag{1.6}$$

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

1.1.2 Filters

Let us consider the transformation T of a data vector Ψ and its permuted version ${}^{\lfloor \pi}T$ with entries ${}^{\lfloor \pi}T_i(\Psi) = T_{\pi(i)}(\Psi)$. The values of the permuted transformation define

$$
{}^{\lfloor \pi}\Psi \equiv \left[{}^{\lfloor m\pi}\Psi,\ {}^{\lfloor n\pi}\Psi \right] \equiv {}^{\lfloor \pi}T(\Psi) \equiv \left[{}^{\lfloor m\pi}T(\Psi),\ {}^{\lfloor n\pi}T(\Psi) \right]. \tag{1.7}
$$

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.

$$
\bar{\Psi}_t = [d_t, d_{t-1}, \ldots, d_{t-\partial}, 1], \tag{1.8}
$$

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.

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 of the presence of unknown offset.

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

$$
\Psi_i = \bar{\Psi}_{S(i)},\ S(i) \in \{1, \ldots, (\mathring{d} + 1)\partial + 1\},\ i = 1, \ldots, \mathring{\Psi}. \tag{1.9}
$$

The mapping S may choose any entry from $\bar{\Psi}$ at most once. It should not introduce delay so that at least some items from the newest data record d_t available at time t have to be selected by S. For simplicity, let us assume that all entries of d_t are chosen and placed as leading entries of Ψ.

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 a leading part of the raw data, say ${}^{\lfloor m}\Psi$, i.e. $\Psi = \left[{}^{\lfloor m}\Psi,\ {}^{\lfloor n}\Psi \right]$. ${}^{\lfloor m}\Psi$ 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 (1.3) (1.5) imply that

$$
f({}^{\lfloor m}\Psi |\, {}^{\lfloor n\pi}\Psi,\ {}^{\lfloor n}\Psi, \Theta) = \left| \underbrace{\frac{\partial\, {}^{\lfloor m\pi}T\left({}^{\lfloor m}\Psi,\ {}^{\lfloor n}\Psi \right)}{\partial\, {}^{\lfloor m}\Psi}}_{\text{Jacobi matrix related to } {}^{\lfloor m}\Psi} \right|^{-1} \prod_{i=1}^{{}^{\lfloor m\pi}\mathring{\Psi}} \mathcal{N}_{{}^{\lfloor m\pi}T_i\left({}^{\lfloor m}\Psi,\, {}^{\lfloor n}\Psi \right)} \left(\theta_i\, {}^{\lfloor \pi}\psi_i', r_i \right). \tag{1.10}
$$

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* ${}^{\lfloor m}\Psi_i$ *and* ${}^{\lfloor n}\Psi$ *only.*

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 $\partial T_i(\Psi)/\partial \Psi_i,\ i = 1, \ldots, {}^{\lfloor m}\mathring{\Psi}$, depending on Ψ_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.

1.1.3 Parameter estimation on filtered data

The possibility to us 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

$$GiW_{\Theta_i}(V,\nu) \equiv GiW_{\theta_i,r_i}(V_i,\nu_i) \equiv \frac{r_i^{-0.5\left(\nu_i + {}^{\lfloor\pi}\mathring{\psi}_i + 2\right)}}{\mathcal{I}(V_i,\nu_i)} \exp\left\{-\frac{1}{2r_i}\mathrm{tr}\left(V_i\,[-1,\theta_i]'\,[-1,\theta_i]\right)\right\} \tag{1.11}$$

and statistics updated according to the formulas

$${}^{\lfloor new}V_i = V_i + \left[{}^{\lfloor\pi}\Psi_i,\ {}^{\lfloor\pi}\psi_i\right]'\left[{}^{\lfloor\pi}\Psi_i,\ {}^{\lfloor\pi}\psi_i\right],\ \ {}^{\lfloor new}\nu_i = \nu_i + 1,\ \ i = 1,\ldots,\ {}^{\lfloor m\pi}\mathring{\Psi}. \tag{1.12}$$

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

$$\mathcal{I}(V_i,\nu_i) \equiv \mathcal{I}(L_i,D_i,\nu_i) = \Gamma(0.5\nu_i)\ {}^{\lfloor d}D_i^{-0.5\nu_i}\left|{}^{\lfloor\psi}D_i\right|^{-0.5} 2^{0.5\nu_i}(2\pi)^{0.5\mathring{\psi}_i}, \tag{1.13}$$

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.

1.1.4 Prediction on filtered data

Predictive pdf of individual filtered data ${}^{\lfloor\pi}\Psi_i$ can be expressed in terms of the normalization integral (1.13) as follows

$$f\left({}^{\lfloor\pi}\Psi_i|\,{}^{\lfloor\pi}\psi_i\right) = \frac{\mathcal{I}\left(V_i + \left[{}^{\lfloor\pi}\Psi_i,\ {}^{\lfloor\pi}\psi_i\right]'\left[{}^{\lfloor\pi}\Psi_i,\ {}^{\lfloor\pi}\psi_i\right],\nu_i+1\right)}{\mathcal{I}(V_i,\nu_i)},\ i = 1,\ldots,\ {}^{\lfloor m\pi}\mathring{\Psi}. \tag{1.14}$$

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

$$\tilde{\psi}_i = \left[\Psi_{i+1},\ldots,\Psi_{{}^{\lfloor m}\mathring{\Psi}},\ {}^{\lfloor n\pi}\Psi\right],\ i = 1,\ldots,\ {}^{\lfloor m}\mathring{\Psi}. \tag{1.15}$$

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

$$f(\Psi_i|\tilde{\psi}_i) = \left[\frac{\partial\,{}^{\lfloor\pi}T_i(\Psi_i,\tilde{\psi}_i)}{\partial\Psi_i}\right]^{-1}\frac{\mathcal{I}\left(V_i + \left[{}^{\lfloor\pi}T(\Psi_i,\tilde{\psi}_i),\tilde{\psi}_i\right]'\left[{}^{\lfloor\pi}T(\Psi_i,\tilde{\psi}_i),\tilde{\psi}_i\right],nu_i+1\right)}{\mathcal{I}(V_i,\nu_i)}. \tag{1.16}$$

The evaluation 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.*

1.1.5 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 (1.5). 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 trandsormations applied to all delayed variables and invertible for the newest value,
- spline based filters.

1.2 Software implementation

Software representation of the approach of *generalized ARX factors* is discussed in this section.

$\boxed{\text{The reader should be familiar with principles of Mixtools processing.}}$

The ideas introduced in the previous section are implemented as structures build by *constructors.*

Namely, each component is a structure. It has the field *comstr* referred to as *component structure* (S in (**??**)). 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 (1.8).

Each component have a field *comstr* referred to as *component structure.* It describes the richest raw data vector needed by component factors (in cell vector *Facs*). The component structure has two rows with meaning of channel and a time delay (the pair $[0; 1]$ introduces *offset* 1). The structure of factors *str* and modelled channels *ychn* are held *relatively* to the component structure.

The constructors are:

Constructors	
`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

The matrix ARX component has only an auxiliary meaning for data input and interpretation of results.

The argument *Facs* means a cell vector of individual factors, the argument *Coms* a cell vector of individual components.

Commented example follows. 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. The factor structure is specified by absolute structure *str*.

```
ychn = [1; 0];                          % modelled "channel"
str  = [2 1 1 2 2; 0 1 2 1 2];          % factor structure
Fac1 = facarx(ychn, str, comstr)        % factor ARX
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'DL of information matrix
```

The second one dynamic factor is build using relative structures - in this case, the *comstr* can be omitted:

```
ychn = 1;                                % modelled "channel"
str  = [3 4  5 6];                       % factor structure
Fac2 = facarx(ychn, str);                % factor ARX
disp(Fac2.ychn)
     1
disp(Fac2.str)
     3     4     5     6
```

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)
```

1.2.1 Coding

Actual objectcong is summarized:

type	*factor*	*component*	*mixture*
ARX	1	11	21
ARX LS	2	12	22
matrix ARX	-	13	23
matrix ARX LS	-	14	24
ARX projection	101	111	121
ARX LS projection	102	112	122
matrix ARX projection	-	113	123
matrix ARX LS projection	-	124	124

Chapter 2

Learning with normal mixtures

2.1 Factors

The toolbox supports ARX and ARX LS factors. The factor constructors are:

Factor constructors	
`Fac = facarx(ychn, str, comstr)`	build ARX factor
`Fac = facarxls(ychn, str, comstr)`	build ARX LS factor

The obligatory argument *comstr* is structure of component data vector. Conversion between ARX types are done by the function *arx*2*arx*.

2.1.1 Estimation

ARX factor estimation related operations are:

Factor estimation	
`Fac = facupdt(Fac, dvect, weight)`	factor update
`Fac = facfrg(Fac,  rate, FacA)`	factor forgetting
`Fac = facflat(Fac, rate, FacA)`	factor flattening

Note:
current projection operation changes only the factor type (by adding 100).

2.2 Components

The toolbox supports the ARX types of components. The component constructors are:

Component constructors	
`Com = comarx(comstr, Facs)`	build ARX or ARX LS component
`Com = matarxls(ychns, str,comstr)`	build matrix ARX LS component
`Com = matarx(ychns, str, comstr)`	build matrix ARX component

Example

Notes:

- *matrix component* are used for inputs and outputs only ;
- the projection components are created by projection operation.

2.2.1 Estimation

The ARX (and ARX LS) component have the fields $Flts, rawdata, datavect, Facs$ used in component estimation and prediction.

The ARX component estimation has the following steps:

- data vector is extracted according to *comstr* and copied into the field *rawdata*;
- filters are invoked and output values copied the *datavector* field;
- component factors are updated - each data value is extracted from *datavect* .

The processing functions are:

Component estimation	
`Com = comupdt(Com, weight)`	component update
`Com = comdata(Com, psi, str)`	get data vector

Notes:

- the function *comdata* offers the possibility to specify some raw data values *psi* described by structure *str*. This is used in predictions.

Example

2.2.2 Component projections

The projection converts a component into component *projection.* 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 another set of modelled channels referred to as *channels in condition.* By "channels" we mean an item of the *comstr*.

The ARX projection of has the states:

state	*meaning*	*defaults*
pchns	predicted channels	all outputs
cchns	channels in condition	no channels in condition
outs	channels that do not enter projection	

The functions processing functions are:

Component projections	
`Com = com2pro(Com, pchns, cchns)`	projection
`Com = com2marg(Com, chns)`	marginalization
`Com = com2pre(Com)`	prediction

Notes:

- *projection* of a factorized component modifies position of factors to correspond the order $[outs, pchns, cchns]$. The factor positions is not shifted but listed in component states (field *predicted*;
- by *marginalization* we mean the projection operation that gets from the component only the part corresponding to the argument *chns* and rebuilds the component.

Example.

Note:
the projection can be re-built for any new selection of output channels.

2.2.3 Component predictions

By *prediction* we mean substitution of current *Com.datavector* into component projection. The *Com.datavector* can be filled by any means.

Example

The processing functions are:

Component prediction	
`Com = com2pre(Com)`	prediction
`Com = comdata(Com, psi, str)`	get data vector

The *Com.datavect* contains valid data at the end of identification so that no other processing is required. The function *comdata* fills the *Com.datavect* in the steps:

- *Com.rawdata* is extracted from DATA;
- filters are called (without changing states) to obtain *Com.datavect* and jacobian of the transformation
- the value of *psi* replaces the *Com.datavect* values - the argument *str* identifies the items to be replaced. The *str* argument can contain only one row, then zeros are substituted as the second line. The *str* can contain rows that are not in *Com.comstr* - this items are skipped;
- the filters are called to update states;
- the factors are called to update their statistics.

Note: he function *comdata* can be called with any 4th argument - then the first steps above are skipped.

Example

2.2.4 Component conversions

The conversions functions are:

Component conversions	
`Com = com2com(Com, type)`	convert to specified type
`Com = arx2arx(Com)`	convert between ARX and ARX LS form
`Com = arx2mat(Com)`	convert factorized component to matrix one
`Com = mat2arx(Com)`	convert matrix component to factorized one
`Com = comshrink(Com)`	compress component regression coefficients
Support of uxiliary matrix factorized component	
`Can = arx2can(Com)`	ARX LS component into matrix factorized ARX LS
`Com = can2arx(Can)`	convert matrix factorized to factorized one
`Com = can2mat(Can)`	convert matrix factorized component into matrix one

Notes:
- the matrix factorized form of component is used in projection operation
- conversion of factorized component to the matrix one implies lose of factor covariances

Example

- conversions from predictor into projector component is done by prediction operations, see next section.

2.3 Filters

Filters are realized as structures. A filter contains coded filter type *type* and internal states needed for raw data transformation.

All filter types are build by the function *fltconst* and the filter operations are done the function *Filters*:

A filter is called as:

Filters	
`Flt = fltconst(typ, args, ...)`	build filter
`[Com,jacob] =filters(Com, fac, mode)`	filter data

The argument *mode* distinquishes the filterring mode:

mode	operation
1	data are filterred, jacobian of transformation *jakob* is returned
2	data are filtered, filter states updated, 2nd output is empty
3	inverze operation $Com.datavect->Com.rawdata$; inverze jacobian is returned

Notes:

- *Composed filters* are introduced as cell vectors of individual filters;
- multidimensional filters are developing
- there is no limitation on the filters on non-modelled part of Ψ.
- setting of mixture filters is discussed in section 2.4.2 .

2.4 Mixtures

The toolbox supports the same mixture types as components. A mixture is build specifying array of components (cell vector) *Coms* and degrees of freedom of the components $dfcs$:

Mixture constructor	
`Mix = mixconst(Coms, dfcs)`	mixture constructor

Notes:

- the components must be of the same type, the mixture type corresponds to it (by adding 10);
- when a mixture is created it has no states - they are added by processing functions.
- the matrix mixture types are used for inputs and outputs only.
- The mixtures of projection types are created by projection operation.

Example

2.4.1 Construction of prior estimate - initialization

The initialization searches for the mixture structure (i.e.number of mixture components, the structure of mixture factors etc.) that maximize the posterior data likelihood on a learning data sample. The initialization is made by the function *mixinit*.

The relevant theory is in [6], simple examples of are in Guide.

The function *mixinit* is called as:

Iniitialization
`Mix=mixinit(Mix0,frg,ndat,niter,options,belief)` initialization

The arguments meaning:

Mix	initialized estimated mixture
Mix0	initial mixture
frg	forgetting rate
ndat	length of data
niter	number of iterations
options	processing options
belief	belief on a guess of richest structure

Example

2.4.2 Estimation

ARX mixture estimation has the steps:

- generate initial mixture e.g by *genmixe*;
- set processing filters (see subsection);
- run recursive or batch mixture estimation by the same functions as in Mixtools refer to Guide.

Estimation operations	
`Mix = ...`	
`mixest(Mix0, frg, niter, opt)`	iterative mixture estimation [b]
`Mix = mixestim(Mix0, frg, ndat)`	quasi-Bayes mixture estimation
`Mix = mixestim(Mix0, frg)`	recursive quasi-Bayes mixture estimation
`Mix = mixestimp(Mix0, frg, ndat)`	projection based quasi-Bayes estimation
`Mix = mixestimp(Mix0, frg)`	projection based recursive quasi-Bayes estimation
`Mix0 = mixflat(Mix)`	mixture flattening
`Mix = mixstats(Mix, ndat)`	compute estimation statistics
`Mix = mixstats(Mix)`	compute statistics recursively
`Mix0 = genmixe(ncom, ychns, str)`	build initial mixture for estimation

[b] opt - options: 'q', 'b', 'f', 'm' for quasi-Bayes, batch Bayes, forgetting branching and estimation with fixed covariances

The function *mixupdt* refers to physical states physical (if equal 0).

Example

Setting filters

Filters can be easily set "manually" or by the functions:

Setting mixture filters	
`Mix = fltpre(Mix, pre)`	set mixture preprocessing filters
`Mix = fltset(Mix, Flt, typ, args)`	set mixture filter(s)

The function *fltpre* sets preprocessing filters (usually scaling). The function *fltset* sets filter according to *type*; the *args* is a cell vector of *typ* dependent arguments:

typ	*sets*
1	*Flt* to channel *arg* and all delays

Example

2.4.3 Projection

The projection converts a mixture into mixture *projection*. The projection is done for all components see section 2.2.2.

Mixture projection is done by the functions:

Mixture projection	
`pMix = mix2pro(Mix, pchns, cchns)`	mixture projection
`pMix = mix2marg(Mix, pchns)`	mixture marginalization

The arguments pchns and cchns are specified in the form $[channels; time - delays]$. The delays can be omitted - then zeros are substituted.

The functions create state *mixstr* that contains all items from components *comstr*. The mixture states are held relatively to *mixstr*.

Example

2.4.4 Prediction

The projection converts mixture projection into mixture *prediction* by making projection of all components, see section 2.2.3.

The task is solved by functions:

Mixture prediction	
`pMix = profix(pMix, psi, str)`	mixture prediction

Notes:

- The value of *psi* replaces the *Com.datavect* values in all components. The argument *str* identifies the items to be replaced. The *str* argument can contain only one row, then zeros are substituted as the second line;

- the output from *profix* may be the same as in Mixtools;
- *profix* can be called with any 4th argument - then no filtering of data is done. If in this. case no data vector items is changed, *psi* can be empty

Example

2.4.5 Mixture simulation

Mixture simulation is done with ARX LS mixture, raw data are generated, any filtering is ignored.

Auxiliary states are generated at the first simulation step. The states has the same meaning as in Mixtools:.

Example

References

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

[2] 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.

[3] 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, VDI/VDE GMA.

[4] 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.

[5] P. Nedoma, M. Kárný, I. Nagy, and M. Valečková, “Mixtools. MATLAB Toolbox for Mixtures”, Tech. Rep. 1995, ÚTIA AV ČR, Prague, 2000.

[6] M. Kárný, J. Böhm, T. V. Guy, L. Jirsa, I. Nagy, P. Nedoma, and L. Tesař, *Optimized Bayesian Dynamic Advising: Theory and Algorithms*, Springer, London, 2005.

Index

```
zarx2can
prodini


echo off
% matrix factorized component  ARX2CAN CAN2MAT
%
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zarx2can
% build matrix component
ychns =1:3;                            %  "modelled channels"
str   = [1 1  2 2  0                   %  common regressor structure
         1 2  1 2  1];
Com   = matarxls(ychns, str);


% fill it by data
Com.Eth = [1 2 3 4 5; 11 22 33 44 55; 111 222 333 444 555];
Com.cove = [11 0.12 0.13; 0 22 0.23; 0 0 33]';
Com.dfm  = 123;
comprt(Com);
... component type 14 ...
comstr
    1  2  3  1  1  2  2  0
    0  0  0  1  2  1  2  1
ychns    1  2  3
str    4  5  6  7  8
Eth
    1  2  3  4  5
    11  22  33  44  55
    111  222  333  444  555
cove
    11  0  0
    0.12  22  0
    0.13  0.23  33
...........................


Com = mat2arx(Com);                    % convert to ARX LS
Can = arx2can(Com);                    % build matrix factorized component
comprt(Can);
... component type 19 ...
```

```
comstr
    1   2   3   1   1   2   2   0
    0   0   0   1   2   1   2   1
ychns     1   2   3
str     1   2   3   4   5   6   7   8
Eth
    -1   0.12   0.1024   -11.6864   -23.3728   -35.0592   -46.7456   -58.432
    0   -1   0.23   -14.53   -29.06   -43.59   -58.12   -72.65
    0   0   -1   111   222   333   444   555
cove
    11   0   0
    0   22   0
    0   0   33
...........................


Com14 = can2mat(Can);                      % back to matrix
comprt(Com14);
... component type 14 ...
comstr
    1   2   3   1   1   2   2   0
    0   0   0   1   2   1   2   1
ychns     1   2   3
str     4   5   6   7   8
Eth
    1   2   3   4   5
    11   22   33   44   55
    111   222   333   444   555
cove
    11   0   0
    0.12   22   0
    0.13   0.23   33
...........................
```

back   Home

```
zarx2mat
prodini


echo off
% ARX <-> matrix ARX
%
% Design : P. Nedoma
% Updated: July 2003
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zarx2mat.m
% matrix component
echo off


Com14 = Com;
% component
comprt(Com14)
... component type 14 ...
comstr
    1  2  3  1  1  2  2  0
    0  0  0  1  2  1  2  1
ychns    1  2  3
str    4  5  6  7  8
Eth
    10  20  30  40  50
    20  40  60  80  100
    30  60  90  120  150
cove
    10  0  0
    0  20  0
    0  0  30
..........................


Com12   = mat2arx(Com14);
comprt(Com12)
... component type 12 ...
comstr
    1  2  3  1  1  2  2  0
    0  0  0  1  2  1  2  1
ychns    1  2  3
str
--- cell 1 ---    2  3  4  5  6  7  8
--- cell 2 ---    3  4  5  6  7  8
--- cell 3 ---    4  5  6  7  8
Eth
--- cell 1 ---    0  0  10  20  30  40  50
--- cell 2 ---    0  20  40  60  80  100
--- cell 3 ---    30  60  90  120  150
cove    10  20  30


Com14a  = arx2mat(Com12);
comprt(Com14a);
... component type 14 ...
comstr
    1  2  3  1  1  2  2  0
    0  0  0  1  2  1  2  1
ychns    1  2  3
str    4  5  6  7  8
Eth
    10  20  30  40  50
    20  40  60  80  100
    30  60  90  120  150
cove
    10  0  0
    0  20  0
    0  0  30
...........................


% === nondiagonal cove ==========
Com14.cove = 0.1*randn(3,3)+Com14.cove;


Com12   = mat2arx(Com14);
comprt(Com12)
... component type 12 ...
comstr
    1  2  3  1  1  2  2  0
    0  0  0  1  2  1  2  1
ychns    1  2  3
str
--- cell 1 ---    2  3  4  5  6  7  8
--- cell 2 ---    3  4  5  6  7  8
--- cell 3 ---    4  5  6  7  8
Eth
--- cell 1 ---    0.118736  0.213881  1.20883  2.41767  3.6265  4.83533  6.04417
--- cell 2 ---    -0.0504419  21.5133  43.0265  64.5398  86.053  107.566
--- cell 3 ---    30  60  90  120  150
cove    10.0854  19.9776  29.9445


Com14a  = arx2mat(Com12);
comprt(Com14a);
... component type 14 ...
comstr
    1  2  3  1  1  2  2  0
    0  0  0  1  2  1  2  1
ychns    1  2  3
str    4  5  6  7  8
Eth
    10  20  30  40  50
    20  40  60  80  100
    30  60  90  120  150
cove
    10.0854  0  0
    0.118736  19.9776  0
    0.207892  -0.0504419  29.9445
..........................
```

back Home

```
zarx2mat1
prodini


echo off
% Matrix and matrix factorized component
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zarx2mat1.m
% build matrix ARX component
ychns = 1:7;
str   = [1 2 3 4 5 6 7    0; 1 1 1 1 1 1 1    1];
Com   = matarxls(ychns, str);
Eth1  = [1 2 3 4 5 6 7]';
Eth   = [];
for i=1:7, Eth = [Eth,i*Eth1]; end
Com.Eth   = [Eth, ones(7,1)];
Com.cove = ltdl(0.01*rand(7,7) + diag([11 22 33 44 55 66 77]));
Com.dfm   = 123;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Com1  = Com;
Com   = mat2arx(Com);                    % array of factors
Can   = arx2can(Com);
Com   = can2mat(Can);
equal(Com, Com1, 1e-14)
ans =
     1
```

```
zcan2mar
prodini


echo off
% matrix factorized component (CAN2MARG, ARX2CAN)
%
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zcan2mar.m
echo off


Com14
Com14 =
     ychns: [1 2 3 4]
       str: [5 6 7 8 9 10]
       dfm: 123
      type: 14
      cove: [4x4 double]
       Eth: [4x6 double]
       Cth: [6x6 double]
    comstr: [2x10 double]
      Flts: {[1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]  [9]  [10]}
Com14.cove
ans =
    11     0     0     0
     0    22     0     0
     0     0    33     0
     0     0     0    44
prt(Com14.Eth)
    1.1  1.2  1.3  1.4  1.5  1
    2.1  2.2  2.3  2.4  2.5  1
    3.1  3.2  3.3  3.4  3.5  1
    4.1  4.2  4.3  4.4  4.5  1
Com14.comstr
ans =
     1     2     3     4     1     2     3     4     8     0
     0     0     0     0     1     1     1     1     0     1


Com12 = com2com(Com14, 12);
Can   = arx2can(Com12);
```

```
Can
Can =
     ychns: [1 2 3 4]
       str: [1 2 3 4 5 6 7 8 9 10]
       dfm: 123
      type: 19
      cove: [4x4 double]
       Eth: [4x10 double]
    comstr: [2x10 double]
Can.cove
ans =
    11     0     0     0
     0    22     0     0
     0     0    33     0
     0     0     0    44
prt(Can.Eth)
    -1  0  0  0  1.1  1.2  1.3  1.4  1.5  1
    0  -1  0  0  2.1  2.2  2.3  2.4  2.5  1
    0  0  -1  0  3.1  3.2  3.3  3.4  3.5  1
    0  0  0  -1  4.1  4.2  4.3  4.4  4.5  1
Can.comstr
ans =
     1     2     3     4     1     2     3     4     8     0
     0     0     0     0     1     1     1     1     0     1


Can1  = can2marg(Can, [4 2]);
Can1
Can1 =
     ychns: [1 3 4 2]
       str: [1 3 4 2 5 6 7 8 9 10]
       dfm: 123
      type: 19
      cove: [4x4 double]
       Eth: [4x10 double]
    comstr: [2x10 double]
Can1.cove
ans =
   11.0000         0         0         0
         0   33.0000         0         0
         0         0   44.0000         0
         0         0         0   22.0000
prt(Can1.Eth)
    -1  0  0  0  1.1  1.2  1.3  1.4  1.5  1
    0  -1  0  0  3.1  3.2  3.3  3.4  3.5  1
    0   0   -1   0   4.1   4.2   4.3   4.4   4.5   1
    0   0   0   -1   2.1   2.2   2.3   2.4   2.5   1
Can1.comstr
ans =
     1     2     3     4     1     2     3     4     8     0
     0     0     0     0     1     1     1     1     0     1
```

back Home

```
zcnvcom
prodini


echo off
% Component conversion: ARX2MAT, MATARXLS, ARX2CAN. CAN2MAT, ARX2MAT
%
% Design : P. Nedoma
% Updated: July 2003
% Updated: January 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zcnvcom.m
% Build and display a matrix component
echo off
comprt(Com14);
... component type 14 ...
comstr
    1  2  3  1  1  2  2  0
    0  0  0  1  2  1  2  1
ychns    1  2  3
str    4  5  6  7  8
Eth
    10  20  30  40  50
    20  40  60  80  100
    30  60  90  120  150
cove
    10  0  0
    0.118736  20  0
    0.207892  -0.0504419  30
...........................


Com12 = mat2arx(Com14);
comprt(Com12);
... component type 12 ...
comstr
    1  2  3  1  1  2  2  0
    0  0  0  1  2  1  2  1
ychns    1  2  3
str
--- cell 1 ---    2  3  4  5  6  7  8
--- cell 2 ---    3  4  5  6  7  8
--- cell 3 ---    4  5  6  7  8
Eth
```

```
--- cell 1 ---    0.118736  0.213881  1.20883  2.41767  3.6265  4.83533  6.04417
--- cell 2 ---    -0.0504419  21.5133  43.0265  64.5398  86.053  107.566
--- cell 3 ---    30  60  90  120  150
cove    10  20  30


Can = arx2can(Com12);
comprt(Can);
... component type 19 ...
comstr
    1  2  3  1  1  2  2  0
    0  0  0  1  2  1  2  1
ychns    1  2  3
str    1  2  3  4  5  6  7  8
Eth
    -1  0.118736  0.213881  1.20883  2.41767  3.6265  4.83533  6.04417
    0  -1  -0.0504419  21.5133  43.0265  64.5398  86.053  107.566
    0  0  -1  30  60  90  120  150
cove
    10  0  0
    0  20  0
    0  0  30
...........................


Com14a = can2mat(Can);
comprt(Com14a);
... component type 14 ...
comstr
    1  2  3  1  1  2  2  0
    0  0  0  1  2  1  2  1
ychns    1  2  3
str    4  5  6  7  8
Eth
    10  20  30  40  50
    20  40  60  80  100
    30  60  90  120  150
cove
    10  0  0
    0.118736  20  0
    0.207892  -0.0504419  30
...........................


Com14b = arx2mat(Com12);
```

```
comprt(Com14a);
... component type 14 ...
comstr
    1  2  3  1  1  2  2  0
    0  0  0  1  2  1  2  1
ychns    1  2  3
str    4  5  6  7  8
Eth
    10  20  30  40  50
    20  40  60  80  100
    30  60  90  120  150
cove
    10  0  0
    0.118736  20  0
    0.207892  -0.0504419  30
...........................
```

```
zcom2arx
prodini


echo off
% ARX2ARX: components
%
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AV CR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zcom2arx
% factors
comstr = [1 2  1 1  2  0              % component structure
          0 0  1 2  1  1];
ychn  = 1;                             %  modelled channel
str   = [1 1  2 2  0; 1 2  0 1  1];    %  factor structure
Fac2  = facarxls(ychn, str, comstr);


Fac2.Eth = [1 2 3 4 5];
Fac2.cove = 11;
Cth    = 0.1*randn(5,5);
for i=1:5, Cth(i,i)=100*i; end
for i=1:5, for j=i+1:5, Cth(i,j)=0; end; end
Fac2.Cth = Cth;


Fac1  = arx2arx(Fac2);   %  build ARX factor


isequal(Fac1, arx2arx(Fac2))
ans =
     1
equal(Fac2, arx2arx(Fac1), 1e-10)
ans =
     1


% Components
Com11 = comarx(comstr, {Fac1, Fac1});  %  build ARX component
Com12 = comarx(comstr, {Fac2, Fac2});  %  build ARX LS component


equal(Com11, arx2arx(Com12), 1e-10)
```

```
ans =
     1
equal(Com12, arx2arx(Com11), 1e-10)
ans =
     1


% Matrix components
ychns = 1:4;                            % channels
str   = [1 2 3 4  0; 1 1 1 1  1];
Com14 = matarxls(ychns, str);
npsi  = size(Com14.Eth,2);
Eth   = [];
for i = 1 :4, Eth = [Eth; (1:npsi)+i*10]; end
cove  = diag(ychns*100);
Cth   = diag( 0.01 * (1:npsi) );
Com14.Eth  = Eth;
Com14.cove = cove;
Com14.Cth  = Cth;

Com13 = arx2arx(Com14);
equal(Com13, arx2arx(Com14), 1e-10)
ans =
     1
equal(Com14, arx2arx(Com13), 1e-10)
ans =
     1
```

back   Home

```
zcom2ca1
prodini


echo off
% component prediction
%
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% com2ca1
% build matrix ARX component
ychns = 1:7;
str   = [1 2 3 4 5 6 7    0; 1 1 1 1 1 1 1    1];
Com   = matarxls(ychns, str);
Eth1  = [1 2 3 4 5 6 7]';
Eth   = [];
for i=1:7, Eth = [Eth,i*Eth1]; end
Com.Eth  = [Eth, ones(7,1)];
Com.cove = ltdl(0.01*rand(7,7) + diag([11 22 33 44 55 66 77]));
Com.dfm  = 123;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Com1  = Com;
Com   = mat2arx(Com);                   % array of factors
Can   = arx2can(Com);
Com   = can2mat(Can);
equal(Com, Com1, 1e-14)
ans =
     1
```

back Home

```
zcom2com
prodini


echo off
% component conversions
%
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zcom2com.m
% matrix ARX LS component
   ychns = [3 1 2];
   str   = [1 2 3 0; 2 3 4 1];
   Com14 = matarxls(ychns, str);
   row   = [1 2 3 4];
   Com14.Eth  = [10+row;20+row;30+row];
   Com14.cove = ltdl(rand(3,3) + diag([11 22 33]));
   Com14.Cth  = ltdl(rand(4,4) + diag([10 20 30 40]));
   Com14.dfm  = 123;


% -> matrix ARX component
   Com13  = com2com(Com14, 13);
   Com14a = com2com(Com13, 14);
   equal(Com14, Com14a, 1e-13)
ans =
     1


% -> ARX component
   Com11  = com2com(Com14, 11);
   Com14b = com2com(Com11, 14);
   max(max(Com14.Eth - Com14b.Eth))
ans =
     0
   max(max(Com14.cove - Com14b.cove))
ans =
     0


% -> ARX LS component
   Com12  = com2com(Com14, 12);
   Com14c = com2com(Com12, 14);
   max(max(Com14.Eth - Com14c.Eth))
ans =
     0
   max(max(Com14.cove - Com14c.cove))
ans =
     0


%%% predictor %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   Com112 = com2marg(Com12);


% -> matrix ARX component
   Com114 = com2com(Com112, 114);


% -> ARX component
   Com111  = com2com(Com114, 111);
   Com114b = com2com(Com111, 114);
   max(max(Com114.Eth - Com114b.Eth))
ans =
  3.5527e-015
   max(max(Com114.cove - Com114b.cove))
ans =
     0


% -> ARX LS component
   Com112a = com2com(Com114, 112);
   Com114c = com2com(Com112a, 114);
   max(max(Com14.Eth - Com14c.Eth))
ans =
     0
   max(max(Com14.cove - Com14c.cove))
ans =
     0
```

back Home

```
zcom2marg
prodini


echo off
% component marginalization, comparison with projection
%
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AV CR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zcom2marg
% build matrix ARX LS component
echo off


% display component
comprt(Com14);
... component type 14 ...
comstr
    1  2  3  4  1  2  3  4  0
    0  0  0  0  1  1  1  1  1
ychns    1  2  3  4
str    5  6  7  8  9
Eth
    11  12  13  14  15
    21  22  23  24  25
    31  32  33  34  35
    41  42  43  44  45
cove
    100  0  0  0
    0  200  0  0
    0  0  300  0
    0  0  0  400
..........................


% Component marginalization
pchns  = [4 2];                        % predicted channels
Com12  = com2com(Com14,12);            % convert to ARX LS


Com112 = com2marg(Com12, pchns);       % component prediction
Com112
Com112 =
        Facs: {[1x1 struct]  [1x1 struct]}
      comstr: [2x7 double]
        Flts: {[0]  [0]  [0]  [0]  [0]  [0]  [0]}
        type: 112
     rawdata: [0 0 0 0 0 0 0]
    datavect: [0 0 0 0 0 0 0]
      states: [1x1 struct]
comprt(Com112)
... component type 112 ...
comstr
    4  2  1  2  3  4  0
    0  0  1  1  1  1  1
ychns    1  2
str
--- cell 1 ---    2  3  4  5  6  7
--- cell 2 ---    3  4  5  6  7
Eth
--- cell 1 ---    0  41  42  43  44  45
--- cell 2 ---    21  22  23  24  25
cove    400  200


Com112a = com2pro(Com112, pchns);
Com112a
Com112a =
        Facs: {[1x1 struct]  [1x1 struct]}
      comstr: [2x7 double]
        Flts: {[0]  [0]  [0]  [0]  [0]  [0]  [0]}
        type: 112
     rawdata: [0 0 0 0 0 0 0]
    datavect: [0 0 0 0 0 0 0]
      states: [1x1 struct]
comprt(Com112a)
... component type 112 ...
comstr
    4  2  1  2  3  4  0
    0  0  1  1  1  1  1
ychns    1  2
str
--- cell 1 ---    2  3  4  5  6  7
--- cell 2 ---    3  4  5  6  7
Eth
--- cell 1 ---    0  41  42  43  44  45
--- cell 2 ---    21  22  23  24  25
cove    400  200
```

```
zcom2marg1
prodini


echo off
% component marginalization, comparison with projection
% no channels are mentioned
%
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AV CR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zcom2marg1.m
% build matrix ARX LS component
echo off


% display component
comprt(Com14);
... component type 14 ...
comstr
    1  2  3  4  1  2  3  4  0
    0  0  0  0  1  1  1  1  1
ychns    1  2  3  4
str    5  6  7  8  9
Eth
    11  12  13  14  15
    21  22  23  24  25
    31  32  33  34  35
    41  42  43  44  45
cove
    100  0  0  0
    0  200  0  0
    0  0  300  0
    0  0  0  400
...........................


% Component marginalization
Com12  = com2com(Com14,12);            % convert to ARX LS
comprt(Com12);
... component type 12 ...
comstr
    1  2  3  4  1  2  3  4  0
    0  0  0  0  1  1  1  1  1
```

```
ychns    1  2  3  4
str
--- cell 1 ---    2  3  4  5  6  7  8  9
--- cell 2 ---    3  4  5  6  7  8  9
--- cell 3 ---    4  5  6  7  8  9
--- cell 4 ---    5  6  7  8  9
Eth
--- cell 1 ---    0  0  0  11  12  13  14  15
--- cell 2 ---    0  0  21  22  23  24  25
--- cell 3 ---    0  31  32  33  34  35
--- cell 4 ---    41  42  43  44  45
cove    100  200  300  400


Com1 = com2marg(Com12);               % component prediction
comprt(Com1)
... component type 112 ...
comstr
    1  2  3  4  1  2  3  4  0
    0  0  0  0  1  1  1  1  1
ychns    1  2  3  4
str
--- cell 1 ---    2  3  4  5  6  7  8  9
--- cell 2 ---    3  4  5  6  7  8  9
--- cell 3 ---    4  5  6  7  8  9
--- cell 4 ---    5  6  7  8  9
Eth
--- cell 1 ---    0  0  0  11  12  13  14  15
--- cell 2 ---    0  0  21  22  23  24  25
--- cell 3 ---    0  31  32  33  34  35
--- cell 4 ---    41  42  43  44  45
cove    100  200  300  400


Com2 = com2pro(Com12);
comprt(Com2)
... component type 112 ...
comstr
    1  2  3  4  1  2  3  4  0
    0  0  0  0  1  1  1  1  1
ychns    1  2  3  4
str
--- cell 1 ---    2  3  4  5  6  7  8  9
--- cell 2 ---    3  4  5  6  7  8  9
--- cell 3 ---    4  5  6  7  8  9
--- cell 4 ---    5  6  7  8  9
Eth
--- cell 1 ---    0  0  0  11  12  13  14  15
--- cell 2 ---    0  0  21  22  23  24  25
--- cell 3 ---    0  31  32  33  34  35
--- cell 4 ---    41  42  43  44  45
cove    100  200  300  400


isequal(Com1, Com2)
ans =
     1
```

```
zcom2pre
prodini


echo off
% component projection and prediction
%
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AV CR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zcom2pre
% build matrix ARX LS component and fill it by data
echo off


% display component
comprt(Com14);
... component type 14 ...
comstr
    1  2  3  4  1  2  3  4  0
    0  0  0  0  1  1  1  1  1
ychns    1  2  3  4
str    5  6  7  8  9
Eth
    11  12  13  14  15
    21  22  23  24  25
    31  32  33  34  35
    41  42  43  44  45
cove
    100  0  0  0
    0  200  0  0
    0  0  300  0
    0  0  0  400
...........................


% Component projection
pchns  = [4 2];                        % predicted channels
Com112 = com2pro(Com14, pchns);        % component prediction
Com112
Com112 =
        Facs: {[1x1 struct]  [1x1 struct]  [1x1 struct]  [1x1 struct]}
      comstr: [2x9 double]
        Flts: {[0]  [0]  [0]  [0]  [0]  [0]  [0]  [0]  [0]}
        type: 112
     rawdata: [0 0 0 0 0 0 0 0 0]
    datavect: [0 0 0 0 0 0 0 0 0]
      states: [1x1 struct]
comprt(Com112)
... component type 112 ...
comstr
    1   2   3   4   1   2   3   4   0
    0   0   0   0   1   1   1   1   1
ychns     1   2   3   4
str
--- cell 1 ---     3   4   2   5   6   7   8   9
--- cell 2 ---     5   6   7   8   9
--- cell 3 ---     4   2   5   6   7   8   9
--- cell 4 ---     2   5   6   7   8   9
Eth
--- cell 1 ---     0   0   0   11   12   13   14   15
--- cell 2 ---     21   22   23   24   25
--- cell 3 ---     0   0   31   32   33   34   35
--- cell 4 ---     0   41   42   43   44   45
cove     100   200   300   400


Com112a = pro2pre(Com112);
comprt(Com112a);
... component type 114 ...
comstr
    4   2   0
    0   0   1
ychns     1   2
str     3
Eth     0   0
cove
    400   0
    0   200
...........................
```

```
zcom2pre1
prodini


echo off
% component projection and prediction
%
% Autor  : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zcom2pre1.m
echo off


% factorized component is build and displayed
Com12
Com12 =
        Facs: {[1x1 struct]  [1x1 struct]  [1x1 struct]}
      comstr: [2x8 double]
        Flts: {[10]  [20]  [30]  [40]  [50]  [60]  [70]  [80]}
        type: 12
     rawdata: [1 2 3 4 5 6 7 8]
    datavect: [10 20 30 40 50 60 70 80]
comprt(Com12);
... component type 12 ...
comstr
    1  2  3  1  2  3  4  0
    0  0  0  1  2  3  4  1
ychns    2  3  1
str
--- cell 1 ---    3  7  8
--- cell 2 ---    4  5
--- cell 3 ---    2  3  4  6  7  8
Eth
--- cell 1 ---    2.3  244  1
--- cell 2 ---    311  322
--- cell 3 ---    1.2  1.3  111  133  144  1
cove    22  33  11


% ----------------------------------------
pchns = 3;
Com112 = com2pro(Com12, pchns);
Com112.states
```

```
ans =
    pchns: 3
    cchns: []
     outs: [2 1]
comprt(Com112);
... component type 112 ...
comstr
    1   2   3   1   2   3   4   0
    0   0   0   1   2   3   4   1
ychns    2   3   1
str
--- cell 1 ---    1   3   4   5   6   7   8
--- cell 2 ---    4   5   6   7   8
--- cell 3 ---    3   4   5   6   7   8
Eth
--- cell 1 ---    0.618557   -0.21134   -68.6598   0   -82.268   -26.1856   -0.360825
--- cell 2 ---    311   322   0   -2.04088e-014   -2.39165e-016
--- cell 3 ---    4.06   111   0   133   436.8   2.2
cove    5.6701   33   42.68


[C114, scale] = pro2pre(Com112)
C114 =
     ychns: 1
       str: 2
       dfm: 1
      type: 114
      cove: 42.6800
       Eth: 30
       Cth: []
    comstr: [2x2 double]
scale =
     0
comprt(C114)
... component type 114 ...
comstr
    3   0
    0   1
ychns    1
str    2
Eth    30
cove    42.68
...........................
```

back Home

```
zcom2pre2
prodini


echo off
% component prediction
%
% Autor  : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% debugging to be compared with Mixtools
% zcom2pre2.m
% build matrix ARX component and fill it by data
ychns = 1:7;
str   = [1 2 3 4 5 6 7   8  0; 1 1 1 1 1 1 1  0   1];
comstr = [ [ychns;zeros(1, 7)], str ];
Com   = matarxls(ychns, str);
Eth1  = [1 2 3 4 5 6 7]';
Eth   = [];
for i=1:8, Eth = [Eth,i*Eth1]; end
Com.Eth  = [Eth, ones(7,1)];
Com.cove = ltdl(0.01*rand(7,7) + diag([11 22 33 44 55 66 77]));
Com.dfm  = 123;


%  create predictor
Com112 = mat2arx(Com);
pchns = [3 4 5];                      % marginal channels
cchns = [6 7];                        % channels in condition
Com112a = com2pro(Com112, pchns, cchns); % create predictor components


% create prediction
psi0 = [666 999 777 888               % resressor vector (Mixtools)
         6   11  7   8];
prt(Com112a.comstr)
    1   2   3   4   5   6   7   1   2   3   4   5   6   7   8   0
    0   0   0   0   0   0   0   1   1   1   1   1   1   1   0   1
dv = Com112a.datavect;
dv(1:7) = 11:17;
dv(8:15) = 1:8;
dv(length(dv))=1;
dv(6) = 666;
dv(7) = 777;
dv(15) = 888;
dv
dv =
  Columns 1 through 13
    11     12     13     14     15    666    777      1      2      3      4      5      6
  Columns 14 through 16
     7    888      1
Com112a.datavect = dv;


[Com, scale] = pro2pre(Com112a)      % create predictor component
Com =
     ychns: [1 2 3]
       str: 4
       dfm: 123
      type: 114
      cove: [3x3 double]
       Eth: [3x1 double]
       Cth: []
    comstr: [2x4 double]
scale =
 -3.0066e+007
comprt(Com)
... component type 114 ...
comstr
    3   4   5   0
    0   0   0   1
ychns    1   2   3
str    4
Eth    21727.2   28971.6   36211.3
cove
    33.0013   0   0
    1.42478e-005   44.0036   0
    3.42652e-005   0.000155166   55.0056
...........................
```

back Home

```
zcom2pro
prodini


echo off
% component projection
%
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AV CR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zcom2pro.m
echo off
Com                                    % trial dynamic component
Com =
     ychns: [1 2 3 4]
       str: [5 6 7 8 9 10]
       dfm: 1
      type: 14
      cove: [4x4 double]
       Eth: [4x6 double]
       Cth: [6x6 double]
    comstr: [2x10 double]
      Flts: {[1]  [2]  [3]  [4]  [5]  [6]  [7]  [8]  [9]  [10]}
Com.comstr, Com.Flts                   % component structure and filters
ans =
     1     2     3     4     1     2     3     4     6     0
     0     0     0     0     1     1     1     1     0     1
ans =
    [1]    [2]    [3]    [4]    [5]    [6]    [7]    [8]    [9]    [10]
Com.comstr(:, Com.str)                 % absolute component structure
ans =
     1     2     3     4     6     0
     1     1     1     1     0     1
Com.Eth                                % regression coefficients
ans =
    11    12    13    14    15    16
    21    22    23    24    25    26
    31    32    33    34    35    36
    41    42    43    44    45    46
Com.cove                               % component noise covariance
ans =
   100     0     0     0
     0   200     0     0
     0     0   300     0
     0     0     0   400


pause % =================================================================


pchns = [4 2];                          % predicted channels
Com1  = com2pro(Com, pchns);            % component prediction
Com1.states                             % description of projection
ans =
    pchns: [4 2]
    cchns: []
     outs: [1 3]
ychns = getflds(Com1.Facs,'ychn');      % relative modelled channels
ychns = Com1.comstr(:, ychns)           % absolute modelled channels
ychns =
     1     2     3     4
     0     0     0     0


Com1.comstr, Com1.Flts                  % no change
ans =
     1     2     3     4     1     2     3     4     6     0
     0     0     0     0     1     1     1     1     0     1
ans =
    [0]    [0]    [0]    [0]    [0]    [0]    [0]    [0]    [0]    [0]
Com2 = com2com(Com1, 114);              % convert to matrix ARX LS
Com2.Eth                                % note permutation of rows
ans =
   11.0000   12.0000   13.0000   14.0000   15.0000   16.0000
   21.0000   22.0000   23.0000   24.0000   25.0000   26.0000
   31.0000   32.0000   33.0000   34.0000   35.0000   36.0000
   41.0000   42.0000   43.0000   44.0000   45.0000   46.0000
Com2.cove                               % note change
ans =
  100.0000         0         0         0
         0  200.0000         0         0
         0         0  300.0000         0
         0         0         0  400.0000


C1 = com2pro(Com1, [1 2 3 4]);
C  = com2pro(Com,  [1 2 3 4]);
equal(C1,C, 1e-13)
ans =
     1
pause % ------------------------------------------------------------


Com3  = com2marg(Com, pchns);          % re-build projection
Com3.states                            % added states
ans =
    pchns: [1 2]
    cchns: []
     outs: []
ychns = getflds(Com3.Facs,'ychn');     % relative modelled channels
ychns = Com3.comstr(:, ychns)          % absolute modelled channels
ychns =
     4     2
     0     0


Com3.comstr, Com3.Flts                 % changed component structure
ans =
     4     2     1     2     3     4     6     0
     0     0     1     1     1     1     0     1
ans =
    [0]    [0]    [0]    [0]    [0]    [0]    [0]    [0]
Com4 = com2com(Com3, 114);             % component to matrix ARX LS
Com4.Eth                               % regressiong coefficients
ans =
   41.0000   42.0000   43.0000   44.0000   45.0000   46.0000
   21.0000   22.0000   23.0000   24.0000   25.0000   26.0000
Com4.cove                              % noise covariance
ans =
  400.0000         0
         0  200.0000


% mixture level
Mix      = mixconst({Com, Com}, [111 222])
Mix =
    Coms: {[1x1 struct]  [1x1 struct]}
    dfcs: [111 222]
    mmod: 4
    type: 24
pMix = mix2pro(Mix, pchns)             % mixture projection
pMix =
      Coms: {[1x1 struct]  [1x1 struct]}
      dfcs: [111 222]
      mmod: 4
      type: 122
    states: [1x1 struct]
pMix.states                             % mixture states
ans =
      predicted: [4 2]
    incondition: []
         mixstr: [2x10 double]
equal(Com1, pMix.Coms{1}, 1e-13)
ans =
     1


pMix  = mix2marg(Mix, pchns)           % re-build projection
pMix =
      Coms: {[1x1 struct]  [1x1 struct]}
      dfcs: [111 222]
      mmod: 4
      type: 122
    states: [1x1 struct]
equal(Com3, pMix.Coms{1}, 1e-13)
ans =
     1
```

back   Home

```
zcom2pro1
prodini


echo off
% component projection
% set dbg=1 in com2pro.m
%
% Autor  : P. Nedoma
% Updated: January 2005
% Project: GACR 102/03/0049, AV CR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zcom2pro1
echo off


% factorized component is build and displayed
Com12
Com12 =
        Facs: {[1x1 struct]  [1x1 struct]  [1x1 struct]}
      comstr: [2x8 double]
        Flts: {[10]  [20]  [30]  [40]  [50]  [60]  [70]  [80]}
        type: 12
     rawdata: [1 2 3 4 5 6 7 8]
    datavect: [10 20 30 40 50 60 70 80]
comprt(Com12);
... component type 12 ...
comstr
    1  2  3  1  2  3  4  0
    0  0  0  1  2  3  4  1
ychns    2  3  1
str
--- cell 1 ---    3  7  8
--- cell 2 ---    4  5
--- cell 3 ---    2  3  4  6  7  8
Eth
--- cell 1 ---    2.3  244  1
--- cell 2 ---    311  322
--- cell 3 ---    1.2  1.3  111  133  144  1
cove    22  33  11


% ----------------------------------------
pchns = 3;
Com112 = com2pro(Com12, pchns);
```

```
Com112
Com112 =
        Facs: {[1x1 struct]  [1x1 struct]  [1x1 struct]}
      comstr: [2x8 double]
        Flts: {[10]  [20]  [30]  [40]  [50]  [60]  [70]  [80]}
        type: 112
     rawdata: [1 2 3 4 5 6 7 8]
    datavect: [10 20 30 40 50 60 70 80]
      states: [1x1 struct]
comprt(Com112);
... component type 112 ...
comstr
    1  2  3  1  2  3  4  0
    0  0  0  1  2  3  4  1
ychns    2  3  1
str
--- cell 1 ---    1  3  4  5  6  7  8
--- cell 2 ---    4  5  6  7  8
--- cell 3 ---    3  4  5  6  7  8
Eth
--- cell 1 ---    0.618557  -0.21134  -68.6598  0  -82.268  -26.1856  -0.360825
--- cell 2 ---    311  322  0  -2.04088e-014  -2.39165e-016
--- cell 3 ---    4.06  111  0  133  436.8  2.2
cove    5.6701  33  42.68
% ------------------------------------------
pchns   = [2 3 1];
Com112a = com2pro(Com112, pchns);
Com112a
Com112a =
        Facs: {[1x1 struct]  [1x1 struct]  [1x1 struct]}
      comstr: [2x8 double]
        Flts: {[10]  [20]  [30]  [40]  [50]  [60]  [70]  [80]}
        type: 112
     rawdata: [1 2 3 4 5 6 7 8]
    datavect: [10 20 30 40 50 60 70 80]
      states: [1x1 struct]
comprt(Com112a);
... component type 112 ...
comstr
    1  2  3  1  2  3  4  0
    0  0  0  1  2  3  4  1
ychns    2  3  1
str
--- cell 1 ---    3  1  4  5  6  7  8
--- cell 2 ---    1  4  5  6  7  8
--- cell 3 ---    4  5  6  7  8
Eth
--- cell 1 ---    -0.21134  0.618557  -68.6598  0  -82.268  -26.1856  -0.360825
--- cell 2 ---    0.228386  -2.7245  23.4266  -30.3753  -99.7589  -0.502449
--- cell 3 ---    1373.66  1307.32  133  436.8  2.2
cove    5.6701  2.40086  586.639
```

```
zcom2pro2
prodini


echo off
% component projection
%
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AV CR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% com2pro2.m
echo off


% factorized component is build and displayed
Com12
Com12 =
        Facs: {[1x1 struct]  [1x1 struct]  [1x1 struct]}
      comstr: [2x8 double]
        Flts: {[10]  [20]  [30]  [40]  [50]  [60]  [70]  [80]}
        type: 12
     rawdata: [1 2 3 4 5 6 7 8]
    datavect: [10 20 30 40 50 60 70 80]
comprt(Com12);
... component type 12 ...
comstr
    1  2  3  1  2  3  4  0
    0  0  0  1  2  3  4  1
ychns    2  3  1
str
--- cell 1 ---    3  7  8
--- cell 2 ---    4  5
--- cell 3 ---    2  3  4  6  7  8
Eth
--- cell 1 ---    2.3  244  1
--- cell 2 ---    311  322
--- cell 3 ---    1.2  1.3  111  133  144  1
cove    22  33  11


% ----------------------------------------
pchns = 3;
Com112 = com2pro(Com12, pchns);
Com112.states
```

```
ans =
    pchns: 3
    cchns: []
     outs: [2 1]
comprt(Com112);
... component type 112 ...
comstr
    1  2  3  1  2  3  4  0
    0  0  0  1  2  3  4  1
ychns    2  3  1
str
--- cell 1 ---    1  3  4  5  6  7  8
--- cell 2 ---    4  5  6  7  8
--- cell 3 ---    3  4  5  6  7  8
Eth
--- cell 1 ---    0.618557  -0.21134  -68.6598  0  -82.268  -26.1856  -0.360825
--- cell 2 ---    311  322  0  -2.04088e-014  -2.39165e-016
--- cell 3 ---    4.06  111  0  133  436.8  2.2
cove    5.6701  33  42.68
% -------------------------------------
pchns   = [2 3 1];
Com112a = com2pro(Com112, pchns);
comprt(Com112a);
... component type 112 ...
comstr
    1  2  3  1  2  3  4  0
    0  0  0  1  2  3  4  1
ychns    2  3  1
str
--- cell 1 ---    3  1  4  5  6  7  8
--- cell 2 ---    1  4  5  6  7  8
--- cell 3 ---    4  5  6  7  8
Eth
--- cell 1 ---    -0.21134  0.618557  -68.6598  0  -82.268  -26.1856  -0.360825
--- cell 2 ---    0.228386  -2.7245  23.4266  -30.3753  -99.7589  -0.502449
--- cell 3 ---    1373.66  1307.32  133  436.8  2.2
cove    5.6701  2.40086  586.639
%--------------------------------------
pchns   = [3 2 1];
Com112b = com2pro(Com112, pchns);
comprt(Com112b);
... component type 112 ...
comstr
    1  2  3  1  2  3  4  0
    0  0  0  1  2  3  4  1
```

```
ychns    2  3  1
str
--- cell 1 ---    1  4  5  6  7  8
--- cell 2 ---    2  1  4  5  6  7  8
--- cell 3 ---    4  5  6  7  8
Eth
--- cell 1 ---    0.57029  -68.084  -4.95099  -75.8485  -5.10249  -0.254637
--- cell 2 ---    -0.0878258  0.278472  -8.70403  22.9918  -37.0368  -100.207  -0.524813
--- cell 3 ---    1373.66  1307.32  133  436.8  2.2
cove    5.77734  2.3563  586.639


pchns   = [1 2 3];
Com112c = com2pro(Com112, pchns);
comprt(Com112c);
... component type 112 ...
comstr
    1  2  3  1  2  3  4  0
    0  0  0  1  2  3  4  1
ychns    2  3  1
str
--- cell 1 ---    3  4  5  6  7  8
--- cell 2 ---    4  5  6  7  8
--- cell 3 ---    2  3  4  5  6  7  8
Eth
--- cell 1 ---    2.3  0  0  3.37101e-015  244  1
--- cell 2 ---    311  322  1.47393e-014  -2.04088e-014  -1.19583e-016
--- cell 3 ---    1.2  1.3  111  0  133  144  1
cove    22  33  11
```

```
zcom2pro3
prodini


echo off
% case study: component projection
%
% Design : P. Nedoma
% Updated: November 2003
% Project: GACR 102/03/0049, AV CR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zcom2ori3
% build matrix ARX LS component and fill it by data
echo off


% display component
comprt(Com14);
... component type 14 ...
comstr
    1  2  3  4  1  2  3  4  0
    0  0  0  0  1  1  1  1  1
ychns    1  2  3  4
str    5  6  7  8  9
Eth
    11  12  13  14  15
    21  22  23  24  25
    31  32  33  34  35
    41  42  43  44  45
cove
    100  0  0  0
    0  200  0  0
    0  0  300  0
    0  0  0  400
...........................


% Component projection
pchns  = [4 2];                        % predicted channels
Com12  = com2com(Com14,12);            % convert to ARX LS
Com112 = com2pro(Com12, pchns);        % component prediction
Com112
Com112 =
        Facs: {[1x1 struct]  [1x1 struct]  [1x1 struct]  [1x1 struct]}
      comstr: [2x9 double]
```

```
        Flts: {[0]  [0]  [0]  [0]  [0]  [0]  [0]  [0]  [0]}
        type: 112
     rawdata: [0 0 0 0 0 0 0 0 0]
    datavect: [0 0 0 0 0 0 0 0 0]
      states: [1x1 struct]
comprt(Com112);                    % description of projection
... component type 112 ...
comstr
    1   2   3   4   1   2   3   4   0
    0   0   0   0   1   1   1   1   1
ychns    1   2   3   4
str
--- cell 1 ---    3   4   2   5   6   7   8   9
--- cell 2 ---    5   6   7   8   9
--- cell 3 ---    4   2   5   6   7   8   9
--- cell 4 ---    2   5   6   7   8   9
Eth
--- cell 1 ---    0   0   0   11   12   13   14   15
--- cell 2 ---    21   22   23   24   25
--- cell 3 ---    0   0   31   32   33   34   35
--- cell 4 ---    0   41   42   43   44   45
cove    100   200   300   400


% convert to matrix projection
Com114 = com2com(Com112, 114);         % convert to matrix ARX LS
Com114
Com114 =
     ychns: [1 2 3 4]
       str: [5 6 7 8 9]
       dfm: 1
      type: 114
      cove: [4x4 double]
       Eth: [4x5 double]
       Cth: [5x5 double]
    comstr: [2x9 double]
comprt(Com114);
... component type 114 ...
comstr
    1   2   3   4   1   2   3   4   0
    0   0   0   0   1   1   1   1   1
ychns    1   2   3   4
str    5   6   7   8   9
Eth
    11   12   13   14   15
    21  22  23  24  25
    31  32  33  34  35
    41  42  43  44  45
cove
    100  0  0  0
    0  200  0  0
    0  0  300  0
    0  0  0  400
...........................


% re-building projection
Com112a = com2pro(Com112, [1 2 3 4]); % re-build projection
comprt(Com112a);
... component type 112 ...
comstr
    1  2  3  4  1  2  3  4  0
    0  0  0  0  1  1  1  1  1
ychns    1  2  3  4
str
--- cell 1 ---    2  3  4  5  6  7  8  9
--- cell 2 ---    3  4  5  6  7  8  9
--- cell 3 ---    4  5  6  7  8  9
--- cell 4 ---    5  6  7  8  9
Eth
--- cell 1 ---    0  0  0  11  12  13  14  15
--- cell 2 ---    0  0  21  22  23  24  25
--- cell 3 ---    0  31  32  33  34  35
--- cell 4 ---    41  42  43  44  45
cove    100  200  300  400


pchns =[4 2];
Com112a = com2marg(Com14, pchns);      % marginal projection
comprt(Com112a)
... component type 112 ...
comstr
    4  2  1  2  3  4  0
    0  0  1  1  1  1  1
ychns    1  2
str
--- cell 1 ---    2  3  4  5  6  7
--- cell 2 ---    3  4  5  6  7
Eth
--- cell 1 ---    0  41  42  43  44  45
--- cell 2 ---    21  22  23  24  25
cove    400  200


return
```

back Home

```
zcomconst
prodini


echo off
% component constructors
%
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zcomconst.m
% build factors
comstr = [1 2  1 1  2  0              % component structure
          0 0  1 2  1  1];
ychn  = 1;                             %  modelled channel
str   = [1 1  2 2  0; 1 2  0 1  1];    %  factor structure
Fac1  = facarx  (ychn, str, comstr);   %  build ARX factor
Fac2  = facarxls(ychn, str, comstr);   %  build ARX LS factor
ychn = 2;
Fac1a = facarx  (ychn, str, comstr);   %  build ARX factor
Fac2a = facarxls(ychn, str, comstr);   %  build ARX LS factor
Fac1
Fac1 =
    ychn: 1
     str: [3 4 2 5 6]
     dfm: 1
    type: 1
      LD: [6x6 double]
Fac2
Fac2 =
    ychn: 1
     str: [3 4 2 5 6]
     dfm: 1
    type: 2
    cove: 1.0000e-004
     Eth: [0 0 0 0 0]
     Cth: [5x5 double]


% components
Com11  = comarx(comstr, {Fac1, Fac1a}); %  build ARX component
Com12  = comarx(comstr, {Fac2, Fac2a}); %  build ARX LS component
Com11
Com11 =
        Facs: {[1x1 struct]  [1x1 struct]}
      comstr: [2x6 double]
        Flts: {[0]  [0]  [0]  [0]  [0]  [0]}
        type: 11
     rawdata: [0 0 0 0 0 0]
    datavect: [0 0 0 0 0 0]
Com12
Com12 =
        Facs: {[1x1 struct]  [1x1 struct]}
      comstr: [2x6 double]
        Flts: {[0]  [0]  [0]  [0]  [0]  [0]}
        type: 12
     rawdata: [0 0 0 0 0 0]
    datavect: [0 0 0 0 0 0]


% matrix components
ychns = [1 2 3];                        %  modelled channels
str   = [1 1  2 2  0; 1 2  1 2  1];     %  common regressor structure
Com13  = matarx(ychns,   str);          %  build matrix ARX component
Com14  = matarxls(ychns, str);          %  build matrix ARX LS component
Com13
Com13 =
     ychns: [1 2 3]
       str: [4 5 6 7 8]
       dfm: 1
      type: 13
        LD: [8x8 double]
    comstr: [2x8 double]
Com14
Com14 =
     ychns: [1 2 3]
       str: [4 5 6 7 8]
       dfm: 1
      type: 14
      cove: [3x3 double]
       Eth: [3x5 double]
       Cth: [5x5 double]
    comstr: [2x8 double]
```

back   Home

```
zcomest
prodini


echo off
% component estimation
%
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zcomest.m
%  data sample
ndat = 100;                             % 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


Com  = Sim.Coms{1};
Com  = arx2arx(Com);


Mix = mixconst(Com, 1);


for TIME = 1:ndat
     Com  =  comupdt(Com);
    Mix  =  mixupdt(Mix, 0);
echo off; end


isequal(Com, Mix.Coms{1})
ans =
     1
```

```
zconst
prodini


echo off
% Mixtools constructors
%
% Autor  : P. Nedoma
% Updated: June 2001
% Project: ProDaCTools
% zconst.m
% factors: comstr must be known
comstr = [1 2  1 1  2  0              % component structure
          0 0  1 2  1  1];
% Mixtools-like definition
ychn  = 1;                             %  modelled channel
str   = [1 1  2 2  0; 1 2  0 1  1];    %  factor structure
Fac1  = facarx(ychn, str, comstr)      %  build ARX factor
Fac1 =
    ychn: 1
     str: [3 4 2 5 6]
     dfm: 1
    type: 1
      LD: [6x6 double]


ychn  = [1; 0];
Fac1a = facarx(ychn, str, comstr)      %  build ARX factor
Fac1a =
    ychn: 1
     str: [3 4 2 5 6]
     dfm: 1
    type: 1
      LD: [6x6 double]
isequal(Fac1, Fac1a)
ans =
     1


% filtered version
ychn  = 1;
str1  = [3 4  2 5 6];                  %  relative
Fac1a = facarx(ychn, str1);            %  build ARX factor
Fac1a = facarx(ychn, str, comstr)      %  build ARX factor
Fac1a =
    ychn: 1
     str: [3 4 2 5 6]
     dfm: 1
    type: 1
      LD: [6x6 double]
isequal(Fac1, Fac1a)
ans =
     1


% ARX LS factor
comstr
comstr =
     1     2     1     1     2     0
     0     0     1     2     1     1
str
str =
     1     1     2     2     0
     1     2     0     1     1
ychn  =  1;
Fac2  = facarxls(ychn, str, comstr)    %  build ARX factor
Fac2 =
    ychn: 1
     str: [3 4 2 5 6]
     dfm: 1
    type: 2
    cove: 1.0000e-004
     Eth: [0 0 0 0 0]
     Cth: [5x5 double]


ychn  = [1; 0];
Fac2a = facarxls(ychn, str, comstr)      %  build ARX factor
Fac2a =
    ychn: 1
     str: [3 4 2 5 6]
     dfm: 1
    type: 2
    cove: 1.0000e-004
     Eth: [0 0 0 0 0]
     Cth: [5x5 double]
isequal(Fac2, Fac2a)
ans =
     1
```

```
% filtered version
ychn  = 1;
str1  = [3 4  2 5 6];                   %  relative
Fac2a = facarxls(ychn, str1);           %  build ARX factor
Fac2a = facarxls(ychn, str, comstr)     %  build ARX factor
Fac2a =
    ychn: 1
     str: [3 4 2 5 6]
     dfm: 1
    type: 2
    cove: 1.0000e-004
     Eth: [0 0 0 0 0]
     Cth: [5x5 double]
isequal(Fac2, Fac2a)
ans =
     1


Com1  = comarx(comstr, {Fac1, Fac1})    %  build ARX component
Com1 =
        Facs: {[1x1 struct]  [1x1 struct]}
      comstr: [2x6 double]
        Flts: {[0]  [0]  [0]  [0]  [0]  [0]}
        type: 11
     rawdata: [0 0 0 0 0 0]
    datavect: [0 0 0 0 0 0]
Com2  = comarx(comstr, {Fac2, Fac2})    %  build ARX LS component
Com2 =
        Facs: {[1x1 struct]  [1x1 struct]}
      comstr: [2x6 double]
        Flts: {[0]  [0]  [0]  [0]  [0]  [0]}
        type: 12
     rawdata: [0 0 0 0 0 0]
    datavect: [0 0 0 0 0 0]


ychns = [3 2 1];                        %  modelled channels
str   = [1 1  2 2  0; 1 2  1 2  1];     %  common regressor structure
Com3  = matarx(ychns,   str)            %  build matrix ARX component
Com3 =
     ychns: [3 2 1]
       str: [4 5 6 7 8]
       dfm: 1
      type: 13
```

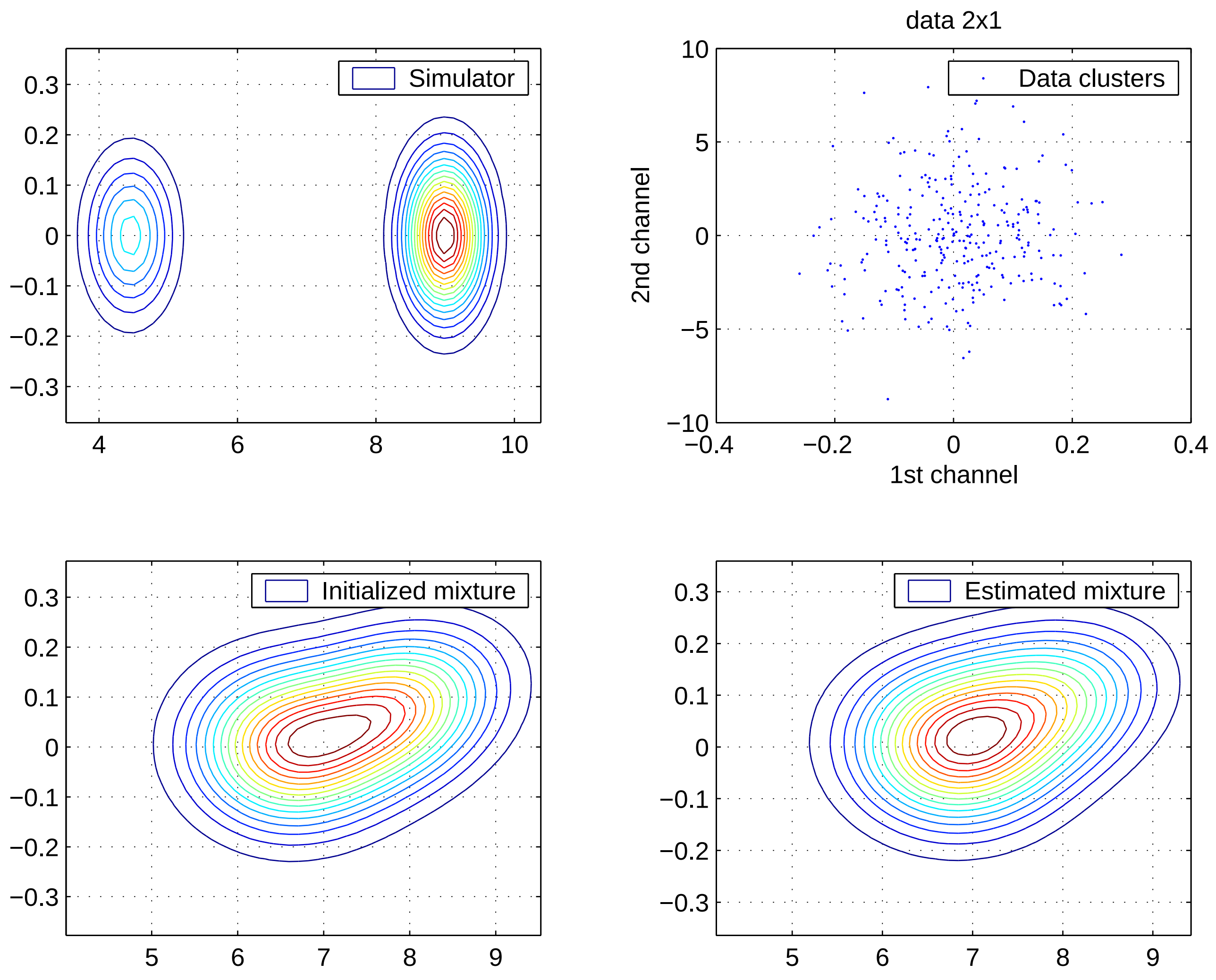
```
        LD: [8x8 double]
    comstr: [2x8 double]
Com4  = matarxls(ychns, str)           %  build matrix ARX LS component
Com4 =
     ychns: [3 2 1]
       str: [4 5 6 7 8]
       dfm: 1
      type: 14
      cove: [3x3 double]
       Eth: [3x5 double]
       Cth: [5x5 double]
    comstr: [2x8 double]


dfcs = 10:11;
Mix1 = mixconst({Com1 Com1}, dfcs)
Mix1 =
    Coms: {[1x1 struct]  [1x1 struct]}
    dfcs: [10 11]
    mmod: 2
    type: 21
Mix2 = mixconst({Com2 Com2}, dfcs)
Mix2 =
    Coms: {[1x1 struct]  [1x1 struct]}
    dfcs: [10 11]
    mmod: 2
    type: 22
Mix3 = mixconst({Com3 Com3}, dfcs)
Mix3 =
    Coms: {[1x1 struct]  [1x1 struct]}
    dfcs: [10 11]
    mmod: 3
    type: 23
Mix4 = mixconst({Com4 Com4}, dfcs)
Mix4 =
    Coms: {[1x1 struct]  [1x1 struct]}
    dfcs: [10 11]
    mmod: 3
    type: 24
```

back   Home

```
zestdyn
prodini


echo off
% Tutorial on dynamic mixtures learning
%
% Design : P. Nedoma
% Updated: November 2002
% Project: ProDaCTools
% zestdyn
randn('seed', 987);
ychn = 1;                               % output channel
uchn = 2;                               % input  channel
comstr = [1 2  1 1 1 1 2 2;  0 0 1 2 3 4 3 4]; % component structure
str    = [1 1 1 1 2 2;  1 2 3 4 3 4];  % common dynamic factor structure


Fac = facarxls(ychn, str, comstr);     % model of output channel


% 1st dynamic system
Fac.Eth  = [1.41833  -1.5894  1.3161  -0.88642  0.2826  0.50666];
Fac.cove = 0.39463^2;                   % variance of output noise
Facs{1}  = Fac;


Fac = facarxls(uchn, [], comstr);      % model of input channel for simulation
Fac.cove = 0.01;                        % variance of output noise
Facs{2}  = Fac;


Coms{1} = comarx(comstr, Facs);


% 2nd dynamic system
Fac      = facarxls(ychn,str,comstr);  % model of output channel
Fac.Eth  = [2.0968  -2.3196  1.9335  -0.8713  0.641  0.1041];
Fac.cove = 0.37255^2;                   % variance of output noise
Facs{1}  = Fac;


Fac = facarxls(uchn, [], comstr);      % model of input channel for simulation
Fac.cove = 0.01;                        % variance of output noise
Facs{2}  = Fac;
```

```
Coms{2}  = comarx(comstr, Facs);


% mixture simulator
Sim      = mixconst(Coms, [0.3 0.7]);  % mixture simulator


% get data sample
ndat = 300;                            % sample size
DATA = zeros(2, ndat);                 % pre-allocated DATA
Sim = mixsimul(Sim, ndat);                   % build data sample


TIME = ndat;
sub = 220;
sub = sub + 1; subplot(sub);
[x,y,z] = mixgrid(Sim); contour(x,y,z,15); grid on
H=legend('Simulator');drawnow


sub = sub + 1; subplot(sub);
datascan([2;1], 1);                    % scatergram of data
H=legend('Data clusters');
xlabel('1st channel');
ylabel('2nd channel')
drawnow


% mixture initialization
% richest mixture
maxstr   = [1 1 1 1 1 1  2 2 2 2 0; 1 2 3 4 5 6  3 4 5 6  1];
Mix0     = genmixe(1,[ychn uchn], maxstr);
niter = 2;
frg   = defaults('frg');               % default forgetting rate
Mix   = mixinit(Mix0, frg, ndat, niter); % mixture initialization
Mix.states.mixll
ans =
 -1.0428e+003
ncom = length(Mix.dfcs)                % number of components
ncom =
     4
Mix.dfcs/sum(Mix.dfcs)                 % component weights
ans =
    0.2184    0.1666    0.4676    0.1474
   M=mix2mix(Mix, 22);
```

```
prt(M);
=== mixture type 22 ===
dfcs :    70.7459  53.9773  151.486  47.7384
--- 1st component ---
... component type 12 ...
comstr
    1  2  1  1  1  1  1  1  2  2  2  2  0
    0  0  1  2  3  4  5  6  3  4  5  6  1
ychns    1  2
str
--- cell 1 ---    2  3  4  5  6  7  8  9  10  11  12  13
--- cell 2 ---    3  4  5  6  7  8  9  10  11  12  13
Eth
--- cell 1 ---    0.739289  1.89715  -2.08465  1.72547  -0.83692  -0.0484994  0.0679966  
--- cell 2 ---    -0.00196533  -0.0186509  0.038387  -0.0465346  0.0387537  -0.024458  0.
cove    0.492133  0.00933535


TIME=ndat;
sub=sub+1; subplot(sub);
[x,y,z] = mixgrid(Mix); contour(x,y,z,15); grid on
H=legend('Initialized mixture');
drawnow


% iterative estimation after initialization
Mix0  = mixflat(Mix);                  % mixture flattenning
niter = 5;                             % number of iterations
Mix   = mixest(Mix0, frg, ndat, 10);   % iterative mixture estimation
Mix.states.mixll
ans =
 -1.1743e+003


TIME = ndat;
sub=sub+1; subplot(sub);
[x,y,z] = mixgrid(Mix); contour(x,y,z,15); grid on
H=legend('Estimated mixture');
figure(1);


plt('zestdyn')
EDIT
```


Simulator
data 2x1
Data clusters
2nd channel
1st channel
Initialized mixture
Estimated mixture

;

Figure 2.1:

```
zestdyn1
prodini


echo off
% Tutorial on dynamic mixtures learning
% NOT FINISHED
%
% Design : P. Nedoma
% Updated: November 2002
% Project: ProDaCTools
% zestdyn1.m
randn('seed', 987);
ychn = 1;                              % output channel
uchn = 2;                              % input  channel
comstr = [1 2  1 1 1 1 2 2;  0 0 1 2 3 4 3 4]; % component structure
str    = [1 1 1 1 2 2;  1 2 3 4 3 4];     % common dynamic factor structure


Fac = facarxls(ychn, str, comstr);     % model of output channel


% 1st dynamic system
Fac.Eth  = [1.41833  -1.5894  1.3161  -0.88642  0.2826  0.50666];
Fac.cove = 0.39463^2;                  % variance of output noise
Facs{1}  = Fac;


Fac = facarxls(uchn, [], comstr);      % model of input channel for simulation
Fac.cove = 0.01;                       % variance of output noise
Facs{2}  = Fac;


Coms{1} = comarx(comstr, Facs);


% 2nd dynamic system
Fac      = facarxls(ychn,str,comstr);  % model of output channel
Fac.Eth  = [2.0968  -2.3196  1.9335  -0.8713  0.641  0.1041];
Fac.cove = 0.37255^2;                  % variance of output noise
Facs{1}  = Fac;


Fac = facarxls(uchn, [], comstr);      % model of input channel for simulation
Fac.cove = 0.01;                       % variance of output noise
```

```
Facs{2}  = Fac;
Coms{2}  = comarx(comstr, Facs);


% mixture simulator
Sim      = mixconst(Coms, [0.3 0.7]);     % mixture simulator


% get data sample
ndat = 1000;                             % sample size
DATA = zeros(2, ndat);                   % pre-allocated DATA
Sim = mixsimul(Sim, ndat);               % build data sample


TIME = ndat;
sub = 220;
sub = sub + 1; subplot(sub);
[x,y,z] = mixgrid(Sim); contour(x,y,z,15); grid on
title('Simulator, time 2000');


sub = sub + 1; subplot(sub);
datascan([2;1], 1);                      % scatergram of data
title('Data clusters');
xlabel('1st channel');
ylabel('2nd channel')


Mix0 = mix2mix(Sim, 21);
Mix0 = mix2mix0(Mix0);


niter = 10;                              % number of iterations
Mix   = mixestim(Mix0, 1, ndat);   % iterative mixture estimation


TIME=ndat;
sub=sub+1; subplot(sub);
[x,y,z] = mixgrid(Mix); contour(x,y,z,15); grid on
title('estimated mixture, time 2000');


es = relep(Mix, ndat);
es
es =
    0.2403
    1.0000
return
```

back Home

```
zestim
prodini


echo off
% Case study: tutorial on static mixture simulation and learning
% NOT FINISHED
%
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zestim.m
randn('seed', 12321);


%  data sample
ndat = 300;                           % 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


sub  = 220;
LegendSize=10;


sub=sub+1; subplot(sub);
[x,y,z] = mixgrid(Sim);
contour(x,y,z,15); grid on
H=legend('Mixture simulator',0);
set(H, 'fontSize', LegendSize); grid on
drawnow


% build initial mixture
Mix0 = genmixe(ncom);                 % build initial mixture
frg  = 1;                             % without forgetting


% iterative estimation after initialization
Mix  = mixestim(Mix0, frg, ndat);     % mixture estimation
Mix.states.mixll
ans =
 -980.4644
```

```
sub = sub+1; subplot(sub);
[x,y,z] = mixgrid(Mix); contour(x,y,z,15);
H=legend('Estimated without filters'); grid on
set(H, 'fontSize', LegendSize); grid on
drawnow
```

```
pre  = {'scale', []};                 % preprocessing requirement
Mix0 = fltpre(Mix0, pre);             % set filters
Mix  = mixestim(Mix0, frg, ndat);     % mixture estimation
Mix.states.mixll
ans =
 -1.3306e+003
```

```
sub = sub+1; subplot(sub);
[x,y,z] = mixgrid(Mix); contour(x,y,z,15);
H = legend('Estimated with filters', 0); grid on
drawnow
```

```
niter = 10;
Mix = mixest(Mix0, frg, ndat, niter);  % iterative quasi-Bayes estimation
Mix.states.mixll
ans =
 -713.6514
```

```
sub = sub+1; subplot(sub);
[x,y,z] = mixgrid(Mix); contour(x,y,z,15);
H=legend('Iterative estimated, filters'); grid on
H = legend('Estimated with filters', 0); grid on
drawnow
setaxis(221:224);
```

back Home

```
zestim1
prodini


echo off
% estimation with scaled data and increments
%      static model
%
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zestim1.m
randn('seed', 123);


%  data sample
ndat = 500;                           % 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


% build initial mixture
Mix0 = genmixe(ncom);                 % build initial mixture
Mix00= Mix0;
frg  = 1;                             % without forgetting


% iterative estimation after initialization
Mix  = mixestim(Mix0, frg, ndat);     % mixture estimation


% filter: scale
pre  = {'scale', []};                 % preprocessing requirement
Mix0 = fltpre(Mix0, pre);             % set filters
Mix1 = mixestim(Mix0, frg, ndat);     % mixture estimation


% filter: increment
Flt  = fltconst(2);
Mix0 = fltset(Mix00, Flt, 1, 1);
Mix0 = fltset(Mix0,  Flt, 1, 2);
Mix2 = mixestim(Mix0, frg, ndat);     % iterative quasi-Bayes estimation
```

```
echo off % display
```

back Home

```
zestim2
prodini


echo off
% estimation with scaled data and increments
%
% Design : P. Nedoma
% Updated: November 2002
% Project: ProDaCTools
% zestim2.m
randn('seed', 987);
ychn = 1;                               % output channel
uchn = 2;                               % input  channel
comstr = [1 2  1 1 1 1 2 2;  0 0 1 2 3 4 3 4]; % component structure
str    = [1 1 1 1 2 2;  1 2 3 4 3 4];     % common dynamic factor structure


Fac = facarxls(ychn, str, comstr);     % model of output channel


% 1st dynamic system
Fac.Eth  = [1.41833  -1.5894  1.3161  -0.88642  0.2826  0.50666];
Fac.cove = 0.39463^2;                  % variance of output noise
Facs{1}  = Fac;


Fac = facarxls(uchn, [], comstr);      % model of input channel for simulation
Fac.cove = 0.01;                       % variance of output noise
Facs{2}  = Fac;


Coms{1} = comarx(comstr, Facs);


% 2nd dynamic system
Fac      = facarxls(ychn,str,comstr);  % model of output channel
Fac.Eth  = [2.0968  -2.3196  1.9335  -0.8713  0.641  0.1041];
Fac.cove = 0.37255^2;                  % variance of output noise
Facs{1}  = Fac;


Fac = facarxls(uchn, [], comstr);      % model of input channel for simulation
Fac.cove = 0.01;                       % variance of output noise
Facs{2}  = Fac;
```

```
Coms{2}  = comarx(comstr, Facs);


% mixture simulator
Sim      = mixconst(Coms, [0.3 0.7]);     % mixture simulator


% get data sample
ndat = 1000;                              % sample size
DATA = zeros(2, ndat);                    % pre-allocated DATA
Sim = mixsimul(Sim, ndat);                      % build data sample

% initial mixture
Mix0 = mix2mix(Sim, 21);
Mix0 = mix2mix0(Mix0);
Mix00 = Mix0;


% estimation
Mix  = mixestim(Mix0, 1, ndat);      % mixture estimation


% filter: scale
pre  = {'scale', []};                     % preprocessing requirement
Mix0  = fltpre(Mix0, pre);                 % set filters
Mix1 = mixestim(Mix0, 1, ndat);      % mixture estimation


% filter: increment
Flt  = fltconst(2);
Mix0 = fltset(Mix00, Flt, 1, 1);
Mix0 = fltset(Mix0,  Flt, 1, 2);
Mix2 = mixestim(Mix0, 1, ndat);      % iterative quasi-Bayes estimation


echo off % display
%
% Design:  P. Nedoma
% Updated: October 2002
% Project: ProDaCTool
% zestiter.m
randn('seed',135531);


%  data sample
```

```
ndat = 200;                            % 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


% standalone estimations
Mix0  = genmixe(ncom);
niter = 5;                              % maximum number of iterations
frg   = defaults('frg');                % default forgetting rate
Mix   = mixest(Mix0,frg,ndat,niter);    % iterative mixture estimation


[x,y,z] = mixgrid(Mix); contour(x,y,z,15);
xlabel('1st channel'); ylabel('2nd channel');
title('Direct estimation'); grid on
Mix.states
ans =
     mixll: -374.3196
    facwgs: [17.2621 17.2621 67.2449 67.2449 89.9204 89.9204 25.5726 25.5726]
    faclls: [1x8 double]
    faceps: [0.4267 0.3209 0.2382 0.8218 -0.4336 1.1438 -0.3606 -0.3406]
    comwgs: [17.2621 67.2449 89.9204 25.5726]
    comlls: [-7.2113e+003 -2.7747e+003 -1.8869e+003 -6.0263e+003]
      dfm0: [100 100 100 100 100 100 100 100]
     dfcs0: [20 20 20 20]
Mix  = mixstats(Mix, ndat);
Mix.states
ans =
     mixll: -369.5807
    facwgs: [17.3702 17.3702 67.1984 67.1984 89.9803 89.9803 25.4512 25.4512]
    faclls: [1x8 double]
    faceps: [0.4267 0.3209 0.2382 0.8218 -0.4336 1.1438 -0.3606 -0.3406]
    comwgs: [17.3702 67.1984 89.9803 25.4512]
    comlls: [-7.1552e+003 -2.8821e+003 -1.8677e+003 -6.2037e+003]
      dfm0: [100 100 100 100 100 100 100 100]
     dfcs0: [20 20 20 20]
```

back Home

```
zestonly
prodini


echo off
% mixture estimation
%             dependency on data sample
%             comparison of mixestim with iterative quasi-bayes
%
% Autor  : P. Nedoma
% Updated: October 2002
% Project: ProDaCTools
% zestonly.m
ndat = 500;                           % size of data sample
ncom = 4;                             % number of components
cove = ltdl([0.1 0.01; 0.01 0.1]);    % component noise covariance
Sim  = statsim(ndat, ncom, cove);     % get data sample


sub  = 230;
sub  = sub + 1; subplot(sub);
[x,y,z] = mixgrid(Sim); contour(x,y,z,15); grid on
title('Simulator'); drawnow


% build initial mixture for estimation
Mix0 = genmixe(ncom);                 % build initial mixture
frg  = defaults('frg');               % default forgetting rate


for i=1:4
    randn('seed',20*i);               % new random sample
    mixsimul(Sim, ndat);              % get data sample
    Mix  = mixestim(Mix0, frg, ndat); % mixture estimation
    mixlls(i) = Mix.states.mixll;     % record posterior data likelihood


    sub  = sub + 1; subplot(sub);
    [x,y,z] = mixgrid(Mix);
    contour(x,y,z,15); grid on
    drawnow; figure(1);
echo off


sub  = sub + 1; subplot(sub);
```

```
plot(mixlls,'o'); hold on; plot(mixlls,':'); grid on
title('mixlls');
%print -deps zeston1
return
```

back Home

```
zfacstr
prodini


echo off
% case study: simulation, estimation and structure estimation
%             of dynamic mixture
%
% Design : P. Nedoma, modified by L. Tesar
% Updated: September 2001, April 2003(LT), July 2003(LT), March 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
ychn = 1;                              % output channel
uchn = 2;                              % input  channel


str     = [1 1 1 1 2 2;  1 2 3 4 3 4];     % common structure
comstr  = [ [1 2; 0 0], str];


Fac = facarxls(ychn, str, comstr);             % model of output channel


% 1st dynamic system
Fac.Eth  = [1.41833  -1.5894  1.3161  -0.88642  0.2826  0.50666];
Fac.cove = 0.39463^2;                  % variance of output noise
Facs1{1}  = Fac;


% 2nd dynamic system
Fac.Eth  = [2.0968  -2.3196  1.9335  -0.8713  0.641  0.1041];
Fac.cove = 0.37255^2;                  % variance of output noise
Facs2{1} = Fac;


% input channel - noise
Fac = facarxls(uchn,[],comstr);               % model of input channel for simulation
Fac.cove = 0.16;                       % variance of output noise
Facs1{2}  = Fac;
Facs2{2}  = Fac;


% components
Coms{1} = comarx(comstr, Facs1);
Coms{2} = comarx(comstr, Facs2);
```

```
% mixture simulator
alpha = 0.3;                           % switching probability
dfcs0 = [alpha, (1-alpha)];            % degree of freedom of components
Sim   = mixconst(Coms, dfcs0);         % mixture simulator


% get data sample
ndat = 2000;                            % sample size
DATA = zeros(2, ndat);                  % pre-allocated DATA
mixsimul(Sim, ndat);                    % build data sample


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% richest mixture
maxstr   = [ones(1,6), 1+ones(1,7), 0
            1:6,       0:6,         1];
comstr   = [ [1; 0], maxstr];
Fac      = facarxls(ychn, maxstr, comstr);     % initial dynamic factor
Fac.cove = 0.01;                       % setting factor fields
Fac.Cth  = eye(length(Fac.Cth));
Facs{1}  = arx2arx(Fac);
Fac      = facarxls(uchn, maxstr, comstr);     % initial noise factor
Fac.cove = 0.01;                       % setting factor fields
Fac.Cth  = eye(length(Fac.Cth));
Facs{2}  = arx2arx(Fac);


Com      = comarx(comstr, Facs);


Mix0     = mixconst(Com, 1);            % initial mixture
frg      = defaults('frg');             % default forgetting rate
Mix      = mixestim(Mix0, frg, ndat);   % mixture estimatio


% repeated estimation
max_nrep  =  1000;                      % maximal number of iteration (optimization) runs
lambda = 0.7;                           % forces making nrep iteration runs
nbest =  10;                            % number of best regressors to hold


Fac   = Mix.Coms{1}.Facs{1};                    % 1st factor
Fac0  = Mix0.Coms{1}.Facs{1};                   % initial 1st factor
```

```
%[MAPstr, lhs, statistics] = facstr(arg(Fac,comstr),arg(Fac0,comstr), [], nbest, max_nrep
 [MAPstr, lhs] = facstr(Fac, Fac0, [], nbest, 100, []);


MAPstr = Fac.str(MAPstr)
MAPstr =
     2     3     4     5    11
MAPstr = comstr(:,MAPstr)
MAPstr =
     1     1     1     1     2
     1     2     3     4     3


vlh = lhs{1};                           % value of log. likelihood
ilh = lhs{2};                           % the best MAP estimates of the structure


% convert values of the likelihood to probabilities
max(vlh)
ans =
 -4.6948e+003
vlh = vlh-max(vlh);llh=vlh;
vlh = exp(vlh);
vlh = vlh/sum(vlh);


% display the best MAP estimates with the probabilities
ilh = ilh';
echo off
 1  1  1  1  1  1  2  2  2  2  2  2  2  0 structure
 1  2  3  4  5  6  0  1  2  3  4  5  6  1 probability
------------------------------------------------------
 1  1  1  1  0  0  0  0  0  1  0  0  0  0    0.644 |  0.0000
 1  1  1  1  0  0  0  1  0  1  0  0  0  0    0.143 | -1.5043
 1  1  1  1  0  0  0  0  0  1  0  0  1  0    0.069 | -2.2303
 1  1  1  1  0  0  0  0  0  1  1  0  0  0    0.063 | -2.3183
 1  1  1  1  0  0  0  0  1  1  0  0  0  0    0.042 | -2.7258
 1  1  1  1  0  1  0  0  0  1  0  0  0  0    0.019 | -3.5417
 1  1  1  1  0  0  0  0  0  1  1  0  1  0    0.007 | -4.5573
 1  1  1  1  0  0  0  0  1  1  0  0  1  0    0.005 | -4.9615
 1  1  1  1  0  0  0  0  0  0  0  0  0  0    0.004 | -4.9965
 1  1  1  1  0  1  0  1  0  1  0  0  0  0    0.004 | -4.9965
```

```
zfacstr1
prodini


echo off
% case study: simulation, estimation and structure estimation
%             of dynamic mixture
%
% Design : P. Nedoma, modified by L. Tesar
% Updated: September 2001, April 2003(LT), July 2003(LT)
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zfacstr1.m
ychn = 1;                             % output channel
uchn = 2;                             % input  channel


str    = [1 1 1 1 2 2;  1 2 3 4 3 4];     % common structure
comstr  = [ [1 2; 0 0], str];


Fac = facarxls(ychn, str, comstr);            % model of output channel


% 1st dynamic system
Fac.Eth  = [1.41833  -1.5894  1.3161  -0.88642  0.2826  0.50666];
Fac.cove = 0.39463^2;                 % variance of output noise
Facs1{1}  = Fac;


% 2nd dynamic system
Fac.Eth  = [2.0968  -2.3196  1.9335  -0.8713  0.641  0.1041];
Fac.cove = 0.37255^2;                 % variance of output noise
Facs2{1} = Fac;


% input channel - noise
Fac = facarxls(uchn,[],comstr);               % model of input channel for simulation
Fac.cove = 0.16;                      % variance of output noise
Facs1{2}  = Fac;
Facs2{2}  = Fac;


% components
Coms{1} = comarx(comstr, Facs1);
```

```
Coms{2} = comarx(comstr, Facs2);


% mixture simulator
alpha = 0.3;                           % switching probability
dfcs0 = [alpha, (1-alpha)];            % degree of freedom of components
Sim   = mixconst(Coms, dfcs0);         % mixture simulator


% get data sample
ndat = 2000;                            % sample size
DATA = zeros(2, ndat);                  % pre-allocated DATA
mixsimul(Sim, ndat);                    % build data sample


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% richest mixture
maxstr   = [ones(1,6), 1+ones(1,7), 0
            1:6,       0:6,         1];
comstr   = [ [1; 0], maxstr];
Fac      = facarxls(ychn, maxstr, comstr);     % initial dynamic factor
Fac.cove = 0.01;                       % setting factor fields
Fac.Cth  = eye(length(Fac.Cth));
Facs{1}  = arx2arx(Fac);
Fac      = facarxls(uchn, maxstr, comstr);     % initial noise factor
Fac.cove = 0.01;                       % setting factor fields
Fac.Cth  = eye(length(Fac.Cth));
Facs{2}  = arx2arx(Fac);


Com      = comarx(comstr, Facs);


Mix0     = mixconst(Com, 1);            % initial mixture
frg      = defaults('frg');             % default forgetting rate
Mix      = mixestim(Mix0, frg, ndat);   % mixture estimatio


% repeated estimation
max_nrep  =  1000;                      % maximal number of iteration (optimization) runs
lambda = 0.7;                           % forces making nrep iteration runs
nbest =  10;                            % number of best regressors to hold


Fac   = Mix.Coms{1}.Facs{1};                     % 1st factor
```

```
Fac0  = Mix0.Coms{1}.Facs{1};                   % initial 1st factor


%[MAPstr, lhs, statistics] = facstr(arg(Fac,comstr),arg(Fac0,comstr), [], nbest, max_nrep
 [MAPstr, lhs] = facstr(Fac, Fac0, [], nbest, 100, []);  % older version


vlh = lhs{1};                          % value of log. likelihood
ilh = lhs{2};                          % the best MAP estimates of the structure


% convert values of the likelihood to probabilities
max(vlh)
ans =
 -4.6948e+003
vlh = vlh-max(vlh);llh=vlh;
vlh = exp(vlh);
vlh = vlh/sum(vlh);


% display the best MAP estimates with the probabilities
ilh = ilh';
echo off
 1  1  1  1  1  1  2  2  2  2  2  2  2  0 structure
 1  2  3  4  5  6  0  1  2  3  4  5  6  1 probability
-------------------------------------------------------
 1  1  1  1  0  0  0  0  0  1  0  0  0  0    0.644 |  0.0000
 1  1  1  1  0  0  0  1  0  1  0  0  0  0    0.143 | -1.5043
 1  1  1  1  0  0  0  0  0  1  0  0  1  0    0.069 | -2.2303
 1  1  1  1  0  0  0  0  0  1  1  0  0  0    0.063 | -2.3183
 1  1  1  1  0  0  0  0  1  1  0  0  0  0    0.042 | -2.7258
 1  1  1  1  0  1  0  0  0  1  0  0  0  0    0.019 | -3.5417
 1  1  1  1  0  0  0  0  0  1  1  0  1  0    0.007 | -4.5573
 1  1  1  1  0  0  0  0  1  1  0  0  1  0    0.005 | -4.9615
 1  1  1  1  0  0  0  0  0  0  0  0  0  0    0.004 | -4.9965
 1  1  1  1  0  1  0  1  0  1  0  0  0  0    0.004 | -4.9965
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Fac   = Mix.Coms{1}.Facs{1};                    % 1st factor
Fac0  = Mix0.Coms{1}.Facs{1};                   % initial 1st factor


npsi   = length(Fac.str);              % length of the factor structure
belief = 4+zeros(1,npsi);              % belief on a guess of maximum structure
MAPstr = facstr(Fac, Fac0, belief)   % MAP estimate of the factor structure
MAPstr =
```

[]

back Home

```
zfltcmp
prodini


echo off
% case study: COMPARISON OF FILTERS
%
% Design : P. Nedoma
% Updated: November 2004
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zfltcmp.m
ndat = 500;                            % size of data sample
ncom = 5;                              % number of components
diac = 0.1;
frg  = 1;                              % forgetting rate
seed = 123;                            % seed of random generator


cove = [diac 0.1*diac; 0.1*diac diac]; % component noise variance
if seed, randn('seed', seed); end


Sim  = statsim(ndat, ncom, cove);      % data sample


sub  = 220;
LegendSize=10;


sub=sub+1; subplot(sub);
datascan
legend('scaled data clusters');


% build initial mixture
Mix0 = genmixe(ncom);                   % build initial mixture
Mix00 = Mix0;


% filter squared
Flt  = fltconst(98);
Mix0 = fltset(Mix0, Flt, 1, 1);
```

```
% estimation
Mix  = mixestim(Mix0, frg, ndat);       % mixture estimation
pMix = mix2pro(Mix,1,2);
ep1  = relep(pMix,ndat);
vll1 = Mix.states.mixll;


sub = sub+1; subplot(sub);
[x,y,z] = mixgrid(Mix); contour(x,y,z,15);
H=legend('filter 98',0);
set(H, 'fontSize', LegendSize); grid on


Flt  = fltconst(99);
Mix0 = fltset(Mix00, Flt, 1, 1);


Mix  = mixestim(Mix0, frg, ndat);      % mixture estimation
pMix = mix2pro(Mix,1,2);
ep2  = relep(pMix, ndat);
vll2 = Mix.states.mixll;


sub = sub+1; subplot(sub);
[x,y,z] = mixgrid(Mix); contour(x,y,z,15);
legend('filter 99',0); grid on


if 1
sub = sub+1; subplot(sub);
plot(0,0,'.');
H = legend(...
   'filter 98 .....',...
  ['v-log-likelihood=',num2str(vll1)],...
  ['prediction-errors norm=', num2str(ep1)], ...
   'filter 99.....',...
  ['v-log-likelihood=',num2str(vll2)],...
  ['prediction-errors norm=', num2str(ep2)], ...
    0); grid on
```

back Home

```
zfltset
prodini


echo off
% setting mixture filters
%
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zfltset.m
covy  = 0.16;
covu  = 0.1;
alpha = 0.3;
diam  = 0.5;


ndat  = 100;
seed  = 123;


niter= 10;
flt  = 1;                               % coded filter type


frg=1;                                  % hiden forgeting


ychn = 1;                               % output channel
uchn = 2;                               % input  channel
str    = [    1 1  1 1 2 2;      1 2 3 4 3 4];    % common dynamic factor structure
comstr = [1 2  1 1 1 1 2 2;  0 0 1 2 3 4 3 4];    % component structure


Fac = facarxls(ychn, str, comstr);     % model of output


% 1st dynamic system
Fac.Eth  = [1.41833  -1.5894  1.3161  -0.88642  0.2826  0.50666];
Fac.cove = covy;                        % variance of output noise
Facs{1}  = Fac;


Fac = facarxls(uchn, [], comstr);      % model of input channel for simulation
```

```
Fac.cove = covu;                        % variance of output noise
Facs{2}  = Fac;


Coms{1}  = comarx(comstr, Facs);


% 2nd dynamic system
Fac      = facarxls(ychn,str,comstr);  % model of output channel
Fac.Eth  = [2.0968  -2.3196  1.9335  -0.8713  0.641  0.1041];
Fac.cove = covy;                        % variance of output noise
Facs{1}  = Fac;


Fac = facarxls(uchn, [], comstr);      % model of input channel for simulation
Fac.cove = covu;                        % variance of output noise
Facs{2}  = Fac;


Coms{2}  = comarx(comstr, Facs);


echo on
Mix      = mixconst(Coms, [alpha, 1-alpha]);  % mixture simulator
DATA     = zeros(2,ndat);


randn('seed', seed);
mixsimul(Mix, ndat);                    % build data sample

prt(Mix)
=== mixture type 22 ===
dfcs :    0.3  0.7
--- 1st component ---
... component type 12 ...
comstr
    1  2  1  1  1  1  2  2
    0  0  1  2  3  4  3  4
ychns    1  2
str
--- cell 1 ---    3  4  5  6  7  8
--- cell 2 ---
Eth
--- cell 1 ---    1.41833  -1.5894  1.3161  -0.88642  0.2826  0.50666
--- cell 2 ---
```

```
cove    0.16  0.1


means = mean(DATA')
means =
    0.2426   -0.0042
stds  = std(DATA')
stds =
    4.7592    0.3157
1./stds
ans =
    0.2101    3.1673


Mix1 = fltpre(Mix,{ 'scale' []} );
Mix1.Coms{1}.comstr
ans =
     1     2     1     1     1     1     2     2
     0     0     1     2     3     4     3     4
Mix1.Coms{1}.Flts
ans =
  Columns 1 through 4
    [1x1 struct]    [1x1 struct]    [1x1 struct]    [1x1 struct]
  Columns 5 through 8
    [1x1 struct]    [1x1 struct]    [1x1 struct]    [1x1 struct]


Mix1.Coms{1}.Flts{1:2}
ans =
     add: -0.2426
     mul: 0.2101
    type: 1
ans =
     add: 0.0042
     mul: 3.1673
    type: 1


% using fltset
Flt  = fltconst(1, -means(1), 1/stds(1)  )
Flt =
     add: -0.2426
     mul: 0.2101
    type: 1
chn  = 1;
```

```
Mix2 = fltset(Mix, Flt, 1, chn)
Mix2 =
    Coms: {[1x1 struct]  [1x1 struct]}
    dfcs: [0.3000 0.7000]
    mmod: 2
    type: 21
Mix2.Coms{1}.Flts
ans =
  Columns 1 through 5
    [1x1 struct]    [0]    [1x1 struct]    [1x1 struct]    [1x1 struct]
  Columns 6 through 8
    [1x1 struct]    [0]    [0]
```

back Home

```
zjumps
prodini


echo off
% static mixture simulation
%       {4 dimensions, 4 components; Markov component jumps
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zjumps.m
ncom  = 4;                            % number of components
ndat  = 1000;                         % length of data sample
DATA  = zeros(4, ndat);               % pre-allocated data sample


str   = [0; 1];                       % static component structure
ychns = [1 2 3 4];                    % modelled channels
Com   = matarxls(ychns, str);         % build matrix ARX LS component
cove  = 0.2*eye(4);                   % diagonal part of noise covariance
Eth   = zeros(4,1);                   % regression coefficients


for com = 1:ncom                      % generate array of components
    Com.Eth   = Eth  + 2*rand(4,1);
    Com.cove  = ltdl(cove + 0.2*rand(4,4));
    Coms{com} = Com;
    Eth = Eth + ones(4,1);
    Com.Eth   = Eth  + 2*rand(4,1);
    Com.cove  = ltdl(cove + 0.2*rand(4,4));
    Coms{com} = Com;
    Eth = Eth + ones(4,1);
    Com.Eth   = Eth  + 2*rand(4,1);
    Com.cove  = ltdl(cove + 0.2*rand(4,4));
    Coms{com} = Com;
    Eth = Eth + ones(4,1);
    Com.Eth   = Eth  + 2*rand(4,1);
    Com.cove  = ltdl(cove + 0.2*rand(4,4));
    Coms{com} = Com;
    Eth = Eth + ones(4,1);
end
Sim   = mixconst(Coms,[4 3 2 1]);     % mixture simulator
```

```
global CUMTAB ACTIVE
   CUMTAB = [0.94 0.02 0.02 0.02       % Markov component change table
             0.02 0.94 0.02 0.02
             0.02 0.02 0.94 0.02
             0.02 0.02 0.02 0.94];
actives = zeros(1,ndat);               % trajectory of active component
for TIME=1:ndat                        % simulation cycle
    mixsimul(Sim);                     % one simulation step
    actives(TIME) = ACTIVE;            % get trajectory
echo off
% matrix ARX -> ARX
%
% Design : P. Nedoma
% Updated: July 2003
% Project: GA CR 102/03/0049, AV CR S1075351, S1075102, DESIGNER
% zmat2arx.m
% Matrix components
ychns = [3 2 1];                       %  modelled channels
str   = [1 1  2 2  0; 1 2  1 2  1];    %  common regressor structure


Com   = matarxls(ychns,   str);        %  build matrix ARX component
eth   = [0.1 0.2 0.3 0.4 0.5];
for i=1:3, Com.Eth(i, :) = i+eth; end
%Com.cove = randn(3,3) + diag([11 22 33]);
Com.cove = diag([11 22 33]);


prt(Com)
ychns       :    3  2  1
str         :    4  5  6  7  8
dfm         :    1
Eth         :
    1.1  1.2  1.3  1.4  1.5
    2.1  2.2  2.3  2.4  2.5
    3.1  3.2  3.3  3.4  3.5
diag(Cth)   :    1e+006  1e+006  1e+006  1e+006  1e+006
cove        :
    11  0  0
    0  22  0
    0  0  33
Com1   = mat2arx(Com);
prt(Com1);
... component type 12 ...
comstr
```

```
    3   2   1   1   1   2   2   0
    0   0   0   1   2   1   2   1
ychns    3   2   1
str
--- cell 1 ---    2   1   4   5   6   7   8
--- cell 2 ---    1   4   5   6   7   8
--- cell 3 ---    4   5   6   7   8
Eth
--- cell 1 ---    0   0   1.1   1.2   1.3   1.4   1.5
--- cell 2 ---    0   2.1   2.2   2.3   2.4   2.5
--- cell 3 ---    3.1   3.2   3.3   3.4   3.5
cove    11   22   33


Com2 = arx2mat(Com1);
equal(Com, Com2, 1e-14)
ans =
     1
```

back   Home

```
zmix2marg
prodini


echo off
% Mixture marginalization
%
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zmix2marg.m
% matrix ARX LS component
   ychns = [3 1 2];
   str   = [1 2 3 0; 2 3 4 1];
   Com   = matarxls(ychns, str);


   row   = [1 2 3 4];
   Com.Eth  = [10+row;20+row;30+row];
   Com.cove = ltdl(rand(3,3) + diag([11 22 33]));
   Com.Cth  = ltdl(rand(4,4) + diag([10 20 30 40]));
   Com.dfm  = 123;


   Mix12  = mixconst({Com, Com, Com}, [11 22 33]);


pMix1 = mix2pro(Mix12, 1:2);
pMix2 = mix2marg(Mix12, 1:2);
equal(pMix2, mix2marg(Mix12, [1:2; 0 0]))
ans =
     1


prt(pMix1);
=== mixture type 122 ===
dfcs :    11  22  33
--- 1st component ---
... component type 112 ...
comstr
    3  1  2  1  2  3  0
    0  0  0  2  3  4  1
ychns    3  1  2
str
```

```
--- cell 1 ---    4  5  6  7
--- cell 2 ---    2  3  4  5  6  7
--- cell 3 ---    3  4  5  6  7
Eth
--- cell 1 ---    11  12  13  14
--- cell 2 ---    0.0290266  0.0179525  19.9027  20.8557  21.8087  22.7618
--- cell 3 ---    0.0748909  30.1762  31.1013  32.0264  32.9515
cove    11.1388  22.1597  33.7955
states
      predicted: [2 3]
    incondition: []
         mixstr: [2x7 double]
prt(pMix2);
=== mixture type 122 ===
dfcs :    11  22  33
--- 1st component ---
... component type 112 ...
comstr
    3  1  2  1  2  3  0
    0  0  0  2  3  4  1
ychns    1  2
str
--- cell 1 ---    2  3  4  5  6  7
--- cell 2 ---    3  4  5  6  7
Eth
--- cell 1 ---    0.0290266  0.0179525  19.9027  20.8557  21.8087  22.7618
--- cell 2 ---    0.0748909  30.1762  31.1013  32.0264  32.9515
cove    22.1597  33.7955
states
      predicted: [2 3]
    incondition: []
         mixstr: [2x7 double]
```

```
zmix2marg1
prodini


echo off
% Mixture marginalization
%
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zmix2marg1.m
% matrix ARX LS component
   ychns = [3 1 2];
   str   = [1 2 3 0; 2 3 4 1];
   Com   = matarxls(ychns, str);


   row   = [1 2 3 4];
   Com.Eth  = [10+row;20+row;30+row];
   Com.cove = ltdl(rand(3,3) + diag([11 22 33]));
   Com.Cth  = ltdl(rand(4,4) + diag([10 20 30 40]));
   Com.dfm  = 123;


   Mix24  = mixconst({Com, Com, Com}, [11 22 33]);


   pMix = mix2marg(Mix24, [2 1]);


prt(pMix);
=== mixture type 122 ===
dfcs :    11  22  33
--- 1st component ---
... component type 112 ...
comstr
    1  3  2  1  2  3  0
    0  0  0  2  3  4  1
ychns    1  2
str
--- cell 1 ---    4  5  6  7
--- cell 2 ---    3  1  4  5  6  7
Eth
--- cell 1 ---    31  32  33  34
--- cell 2 ---    0.0179525  0.0290266  19.9027  20.8557  21.8087  22.7618
cove    33.858  22.1597
states
      predicted: [3 1]
    incondition: []
         mixstr: [2x7 double]
pMix.states.mixstr
ans =
     1     3     2     1     2     3     0
     0     0     0     2     3     4     1
```

```
zmix2mix
prodini


echo off
% conversion of mixtures
%
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zmix2mix
% matrix ARX LS component
   ychns = [3 1 2];
   str   = [1 2 3 0; 2 3 4 1];
   Com   = matarxls(ychns, str);


   row   = [1 2 3 4];
   Com.Eth  = [10+row;20+row;30+row];
   Com.cove = ltdl(rand(3,3) + diag([11 22 33]));
   Com.Cth  = ltdl(rand(4,4) + diag([10 20 30 40]));
   Com.dfm  = 123;


   Mix24  = mixconst({Com, Com, Com}, [11 22 33]);
   Mix23  = mix2mix(Mix24, 23);
   Mix24a = mix2mix(Mix23, 24);
   Mix21  = mix2mix(Mix24, 21);


   Mix24b = mix2mix(Mix21, 24);
   Mix22  = mix2mix(Mix24, 22);
   Mix24c = mix2mix(Mix22, 24);


   Mix22  = mix2mix(Mix24, 22);
```

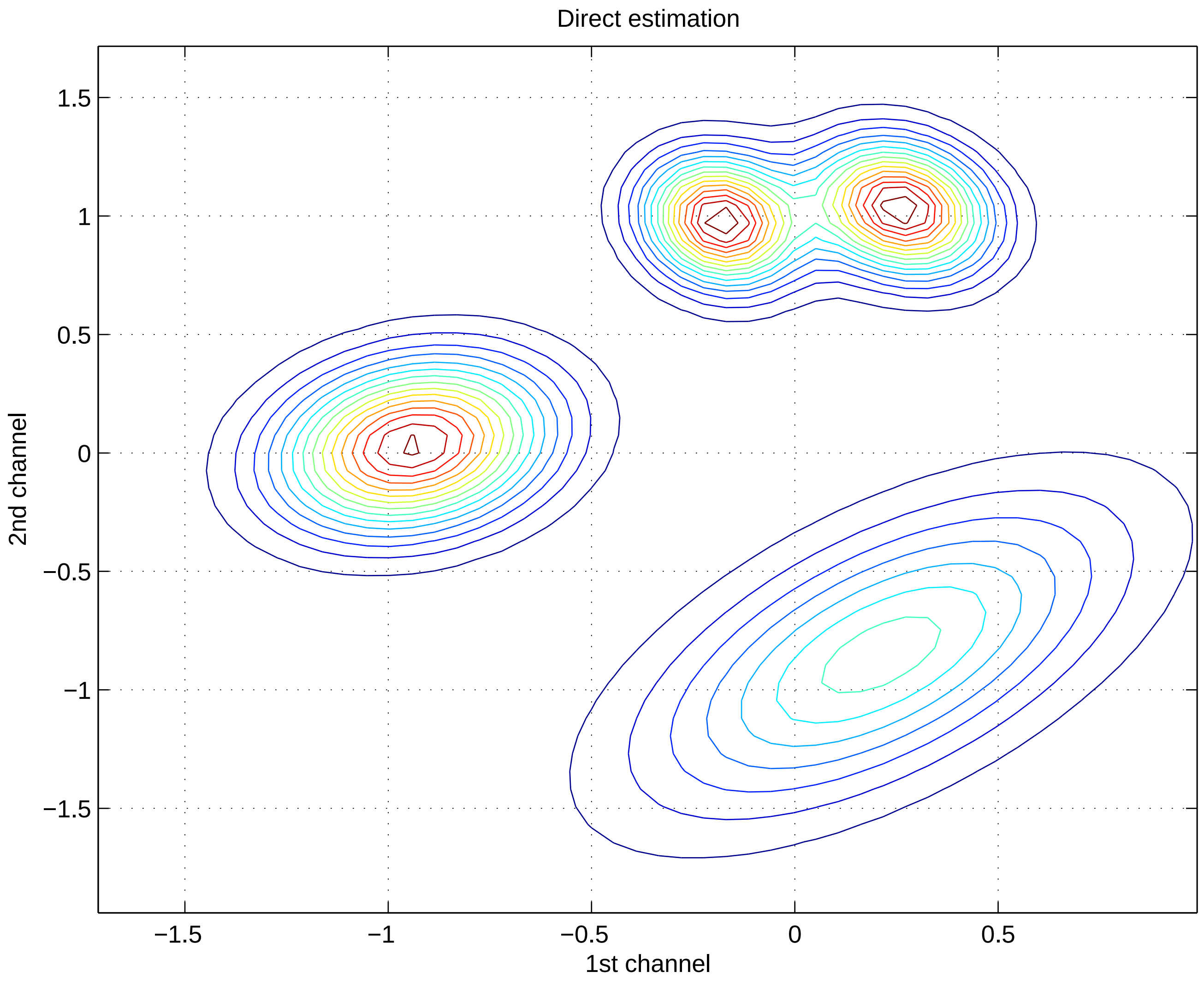
back Home

```
zmixcons
prodini


echo off
% mixture constructor
%
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zmixcons.m
randn('seed', 135531);


%  data
ndat = 300;                           % 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);     % generate data


% estimations
Mix0  = genmixe(ncom)                 % generate initial mixture
Mix0 =
    Coms: {[1x1 struct]  [1x1 struct]  [1x1 struct]  [1x1 struct]}
    dfcs: [30 30 30 30]
    mmod: 2
    type: 24


Mix   = mixestim(Mix0, 1, ndat);      % mixture estimation
Mix
Mix =
      Coms: {[1x1 struct]  [1x1 struct]  [1x1 struct]  [1x1 struct]}
      dfcs: [58.3641 128.6313 68.3444 164.6602]
      mmod: 2
      type: 21
    states: [1x1 struct]
Mix.states
ans =
     mixll: -978.6976
    facwgs: [1x8 double]
    faclls: [1x8 double]
    faceps: [0.0480 2.5999 -3.0184 3.6067 1.0044 2.2357 2.9654 -2.1102]
    comwgs: [28.3641 98.6313 38.3444 134.6602]
    comlls: [-1.0026e+004 -5.5967e+003 -1.1177e+004 -4.4988e+003]
      dfm0: [100 100 100 100 100 100 100 100]
     dfcs0: [30 30 30 30]
```

```
[x,y,z] = mixgrid(Mix); contour(x,y,z,15);
xlabel('1st channel'); ylabel('2nd channel');
title('Direct estimation'); grid on
```

```
plt('zmixcons');
EDIT
```

back Home


Direct estimation
1.5
1
0.5
0
−0.5
−1
−1.5
2nd channel
−1.5
−1
−0.5
0
0.5
1st channel

;

Figure 2.2:

```
zmixinit
prodini


echo off
% mixture initialization : tutorial example
%
% Design : P. Nedoma
% Updated: March 2003
% Project: ProDaCTools
% zmixinit.m
% data sample
randn('seed', 1357);


ndat = 500;                              % size of data sample
cove = ltdl([0.1 0.01; 0.01 0.1]);     % common component noise variance
ncom = 4;                                % number of components
Sim  = statsim(ndat, ncom, cove);      % build simulator and data sample


sub  = 130;
sub  = sub+1; subplot(sub);
[x,y,z] = mixgrid(Sim); contour(x,y,z,15); grid on
M = legend('Simulator', 0);
drawnow


% build initial mixture
Mix0 = genmixe;                          % build initial mixture


% make mixture initialization
Mix   = mixinit(Mix0, 1, ndat);  % mixture initialization


sub = sub+1; subplot(sub);
[x,y,z] = mixgrid(Mix); contour(x,y,z,15); grid on
M=legend('Initialized mixture',0);
drawnow


% iterative estimation after initialization
Mix0  = mixflat(Mix);                    % mixture flattening
niter = 10;                              % number of iterations
```

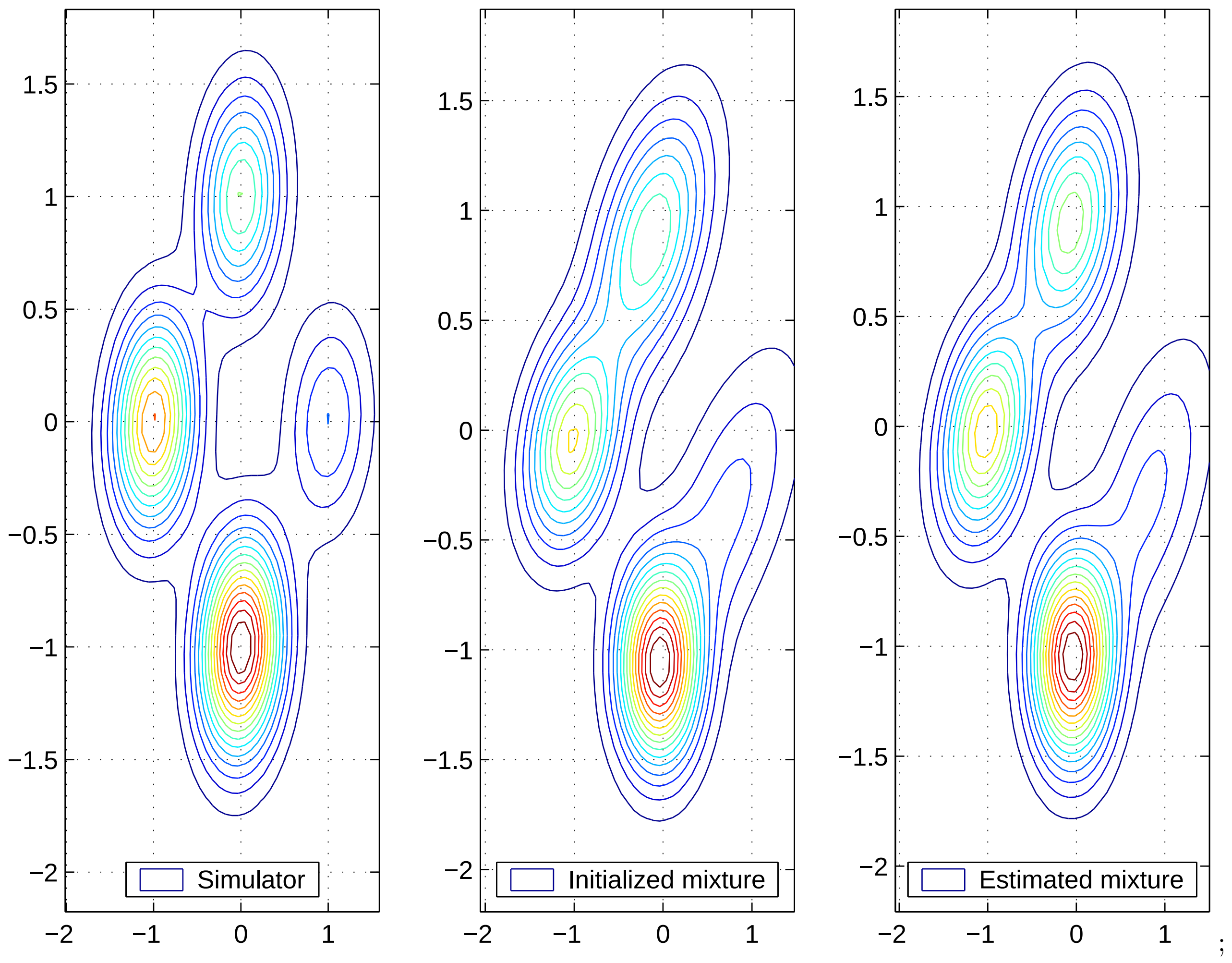


Figure 2.3:

```
Mix   = mixest(Mix0,1,ndat,niter);   % iterative mixture estimation


sub = sub+1; subplot(sub);
[x,y,z] = mixgrid(Mix); contour(x,y,z,15); grid on
H = legend('Estimated mixture', 0);


plt('zmixinit')
EDIT
```

back Home

```
zmixpre
prodini


echo off
% mixture projection  with scaled data
%
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zmixpre.m
randn('seed', 21); %21
ndat = 500;                          % length of data
ncom = 3;                             % number of components
cove = ltdl([0.1 0.01; 0.01 0.1]);    % noise covariance
Mix  = statsim(ndat, ncom, cove);     % get data sample
preproc({'scale', [5 7; 2 3]});       % transform data


sub = 220;
%sub = sub + 1; subplot(sub);
%datascan([], 1); title('Data clusters'); drawnow
Mix0  = genmixe(ncom);                % initial mixture for estimation
frg   = defaults('frg');              % forgetting rate
Mix1  = mixestim(Mix0,frg,ndat);      % estimate mixture


sub = sub + 1; subplot(sub);
[x,y,xy] = mixgrid(Mix1); contour(x,y,xy,20); grid on;
title('No scaling'); drawnow


% projection
pchns = 1;                            % predicted channel
cchns = 2;                            % channel in condition
pMix  = mix2pro(Mix1,pchns,cchns);    % mixture projector


psi0  = 2; str = [2;0];               % zero-delayed regression vector
pM    = profix(pMix, psi0, str);      % mixture projection


sub = sub + 1; subplot(sub);
[x,y] = mixgrid(pM); plot(x,y); grid on;
```

```
title('Prediction psi0=20'); drawnow


% define filters
pre   = preinit({'scale',[]});          % scaling list
Mix0  = fltpre(Mix0, pre);              % copy to processing channels
Mix2  = mixestim(Mix0, frg, ndat);      % estimate mixture


sub = sub + 1; subplot(sub);
[x,y,xy] = mixgrid(Mix2); contour(x,y,xy,20); grid on;
title('With scaling'); drawnow


pMix = mix2pro(Mix2, pchns, cchns);    % mixture projection
psi0 = 20;                              % not scalled psi0
pM   = profix(pMix, psi0, str);        % mixture prediction


sub = sub + 1; subplot(sub);
[x,y] = mixgrid(pM); plot(x,y); grid on;
title('Prediction psi0=20');


Mix11 = mix2mix(Mix1, 24);
Mix21 = mix2mix(Mix2, 24);
Eth1  = getflds(Mix11.Coms, 'Eth');
Eth2  = getflds(Mix21.Coms, 'Eth');
Eth1{:}
ans =
    8.9953
   23.5382
ans =
   12.0382
   20.9616
ans =
    9.0124
   18.4650
Eth2{:}
ans =
   -0.3651
   -0.8742
ans =
    1.9600
    0.1423
```

```
ans =
   -0.3802
    1.1685
```

back Home

```
zmixpre1
prodini


echo off
% mixture projection  with scaled data
%
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zmixpre1.m
randn('seed', 21); %21
ndat = 500;                             % data length
ncom = 3;                               % number of components
cove = ltdl([0.1 0.01; 0.01 0.1]);      % noise covariance
Mix  = statsim(ndat, ncom, cove);       % get data sample
data = DATA;
preproc({'scale', [5 7; 2 3]});         % transform data


sub = 220;
Mix0  = genmixe(ncom);                  % initial mixture for estimation
frg   = 1;                              % no forgetting
Mix1  = mixestim(Mix0,frg,ndat);        % estimate mixture


sub = sub + 1; subplot(sub);
datascan;


sub = sub + 1; subplot(sub);
[x,y,xy] = mixgrid(Mix1); contour(x,y,xy,20); grid on;
title('data scaled'); drawnow


% define filters
DATA  = data;
pre   = {'scale', [5 7; 2 3]};          % scaling list
Mix0  = fltpre(Mix0, pre);              % copy to processing channels


sub = sub + 1; subplot(sub);
datascan;
```

```
Mix2  = mixestim(Mix0,frg,ndat);       % estimate mixture
sub = sub + 1; subplot(sub);
[x,y,xy] = mixgrid(Mix2); contour(x,y,xy,20); grid on;
title('scaling'); drawnow
```

back Home

```
zmixpro
prodini


echo off
% mixture projection and prediction
%
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zmixpro.m
% generate mixture
cove = [0.1 0.01; 0.01 0.1];          % common component noise variance
ncom = 4;                             % number of components
ndat = 100;                           % data length
Mix  = statsim(ndat, ncom, cove);     % build ARX LS mixture estimator


% mixture projection
pchns = 1;
cchns = 2;
pMix  = mix2pro(Mix,pchns,cchns);     % predictor - marginal of 1st channel
str   = [cchns 0]';


echo off


pause


y = vals;
clf; hold off
meshc(x,y,Y);
pause


% substituting real data
DATA = zeros(2,50);
mixsimul(Mix, 50);
echo off
```

back   Home

```
zmixpro1
prodini


echo off
% mixture prediction
%         static mixture, data dependent estimation
%
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zmixpro1.m
% generate mixture
   cove = [0.1 0.01; 0.01 0.1];        % common component noise variance
   ncom = 4;                           % number of components
   Mix  = statsim(0, ncom, cove);      % build ARX LS mixture estimator
   pMix = mix2pro(Mix,1,2);            % predictor - marginal of 1st channel


DATA = zeros(2,1);


echo off
y = -2:0.05:2;
meshc(x,y,Y);
pause


% substituting real data
DATA = zeros(2,50);
mixsimul(Mix, 50);
echo off
```

back Home

```
znocth
prodini


echo off
% znocth: components conversion
%
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% znoctr.m
% build matrix ARX LS component
   ychns = [3 1 2];
   str   = [1 2 3 0; 2 3 4 1];
   Com14 = matarxls(ychns, str);
   row   = [1 2 3 4];
   Com14.Eth  = [10+row;20+row;30+row];
   Com14.cove = ltdl(rand(3,3) + diag([11 22 33]));
   Com14.Cth  = ltdl(rand(4,4) + diag([10 20 30 40]));
   Com14.dfm  = 123;


% -> matrix ARX component
   Com12  = mat2arx(Com14);
   Com14a = arx2mat(Com12);


Com14.Cth
ans =
   10.2712         0         0         0
    0.0248   20.3470         0         0
    0.0016    0.0319   30.2593         0
    0.0218    0.0032    0.0154   40.3606
Com14a.Cth
ans =
     1000000           0           0           0
           0     1000000           0           0
           0           0     1000000           0
           0           0           0     1000000
x=getflds(Com12.Facs,'Cth')
x =
    [6x6 double]    [5x5 double]    [4x4 double]
x{:}
ans =
    0.0004         0         0         0         0         0
   -0.0909    0.0002         0         0         0         0
   -0.0006   -0.0006   10.5979         0         0         0
   -0.0004   -0.0003    0.0417   20.7170         0         0
   -0.0003   -0.0002    0.0142    0.0449   30.6800         0
   -0.0002   -0.0002    0.0319    0.0140    0.0264   40.8322
ans =
    0.0002         0         0         0         0
   -0.0007   10.4740         0         0         0
   -0.0004    0.0352   20.5721         0         0
   -0.0003    0.0093    0.0398   30.5108         0
   -0.0002    0.0279    0.0096    0.0219   40.6382
ans =
   10.2712         0         0         0
    0.0248   20.3470         0         0
    0.0016    0.0319   30.2593         0
    0.0218    0.0032    0.0154   40.3606
```

```
zpro2pre
prodini


echo off
% case study: component prediction
%
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zpro2pre.m
% build matrix ARX component
ychns = 1:7;
str   = [1 2 3 4 5 6 7   8  0; 1 1 1 1 1 1 1  0   1];
Com   = matarxls(ychns, str);
Eth1  = [1 2 3 4 5 6 7]';
Eth   = [];
for i=1:8, Eth = [Eth,i*Eth1]; end
Com.Eth  = [Eth, ones(7,1)];
Com.cove = ltdl(0.01*rand(7,7) + diag([11 22 33 44 55 66 77]));
Com.dfm  = 123;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  create predictor
pchns = [3 4 5];                        % marginal channels
cchns = [6 7];                          % channels in condition
Com   = com2pro(Com, pchns, cchns);
TIME  = 2;                     % time instant
DATA  = [ [1:8]', [11:18]'];  % previous and current data vector


% ==================================
val = [666 999 777 888];
str = [6   11  7   8  ];


Com = comdata(Com, val, str);
[Com1, scale] = pro2pre(Com)
Com1 =
     ychns: [1 2 3]
       str: 4
       dfm: 123
      type: 114
      cove: [3x3 double]
       Eth: [3x1 double]
       Cth: []
    comstr: [2x4 double]
scale =
 -3.0066e+007


Com1
Com1 =
     ychns: [1 2 3]
       str: 4
       dfm: 123
      type: 114
      cove: [3x3 double]
       Eth: [3x1 double]
       Cth: []
    comstr: [2x4 double]
comprt(Com1);
... component type 114 ...
comstr
    3  4  5  0
    0  0  0  1
ychns    1  2  3
str    4
Eth    21727.2  28971.6  36211.3
cove
    33.0013  0  0
    1.42478e-005  44.0036  0
    3.42652e-005  0.000155166  55.0056
...........................


Com = comdata(Com, [111 444], [1 4], 'no filter')
Com =
        Facs: {1x7 cell}
      comstr: [2x16 double]
        Flts: {1x16 cell}
        type: 112
     rawdata: [11 12 13 14 15 16 17 1 2 3 4 5 6 7 18 1]
    datavect: [111 12 13 444 15 666 777 1 2 3 4 5 6 7 888 1]
      states: [1x1 struct]
Com.comstr
ans =
  Columns 1 through 13
     1     2     3     4     5     6     7     1     2     3     4     5     6
     0     0     0     0     0     0     0     1     1     1     1     1     1
```

```
Columns 14 through 16
    7     8     0
    1     0     1
```

back Home

```
zprodep
prodini


echo off
% DATA DEPENDENT PREDICTION, MARKOV COPONENT JUMPS
%
% Design : P. Nedoma
% Updated: October 2002
% Project: ProDaCTools
% zprodep.m

randn('seed', 246);                    % define trajectory for randn


% Build simulated data sample:
% simulated static sample is used with Markov jumps between component.
% in the case of "true" mixture, nothing can be gained in prediction.
ndat = 1000;                           % \ndat
ncom = 4;                              % \ncom
cove = ltdl([0.1 0.01; 0.01 0.1]);     % \cove
Sim  = statsim(0, ncom, cove);         % define simulator (no data sample)


global CUMTAB ACTIVE                   % Markov jumps among components
CUMTAB = [0.97 0.01 0.01 0.01          % probability of transitions
          0.01 0.97 0.01 0.01          % between components
          0.01 0.01 0.97 0.01
          0.01 0.01 0.01 0.97 ];
DATA   = zeros(2, ndat);               % pre-allocate data
mixsimul(Sim, ndat);                   % \mixsimul



sub = 220;                             % composed plot will be generated
sub = sub + 1; subplot(sub);
[x,y,xy] = mixgrid(Sim);               % plot of simulator
 contour(x,y,xy,15); grid on
legend('Simulator');
drawnow


sub = sub + 1; subplot(sub);
plot(DATA(1,:),'.', 'MarkerSize',3);   % plot of 1st data channel
grid on; legend('Data 1st channel');
```

```
drawnow
% note the jumps between components
% mixture estimation
randn('seed', 4321);                    % fix seed for genmixe
Mix0 = genmixe(ncom);                   % \genmixe
frg  = 1;                               % \frg
Mix  = mixestim(Mix0, 1, ndat);         % \mixestim


sub = sub + 1; subplot(sub);
[x,y,xy] = mixgrid(Mix);                % plot of estimated mixture
contour(x,y,xy,15); grid on
legend('Estimated mixture');
drawnow


% Recursive prediction; profix is called with 4 output arguments
% the last argument is the weight dependent on data, see Theory ....
% Prediction error of a different data sample is computed
randn('seed', 9812);                    % fix seed for 2nd data sample
mixsimul(Sim, ndat);                    % \mixsimul
pchns = 1;                              % \pchns
pMix  = mix2pro(Mix,pchns);             % \mix2pro
for TIME = 1:ndat                       % prediction trajectory
   [Eths, coves, alphas, weights] = profix(pMix); % \profix
   yp(TIME) = Eths * weights';          % prediction error
echo off


% std of data is higher than the std of  prediction.
% It means that the prediction (the linear model behind)
% can explain part of variance of data sample
dd  = DATA(pchns,:);                    % data row
ee  = dd-yp;                            % trajectory of prediction error
s2d = std(dd')                          % std of data row
s2d =
    0.7441
mean(ee')                               % mean must be added
ans =
   -0.0269
s2e = std(ee')                          % std of prediction errpr
s2e =
    0.7441
```

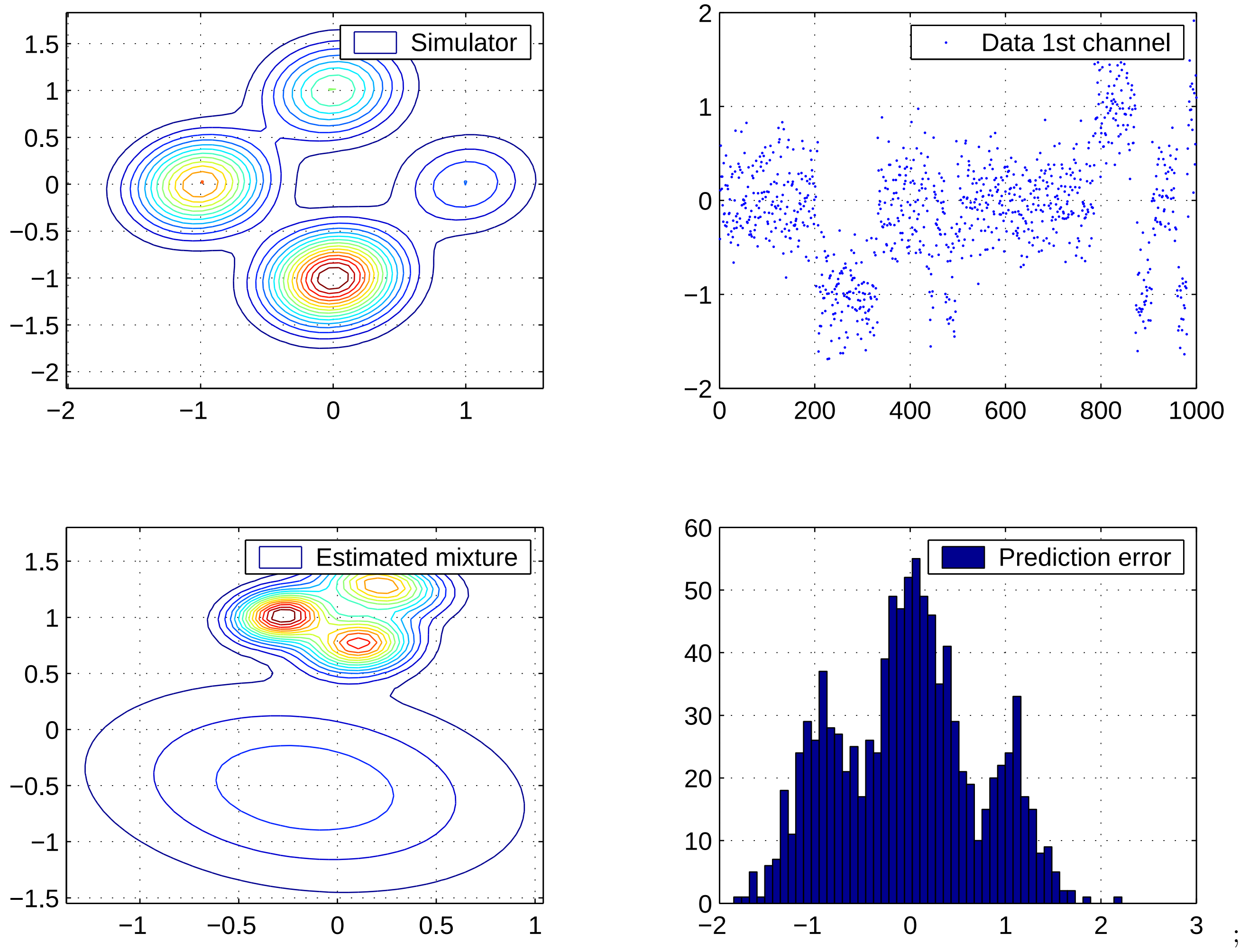


Figure 2.4:

```
sub = sub + 1; subplot(sub);
hist(ee,50); grid on                    % histogram of prediction error
legend('Prediction error');
plt('zprodep');
EDIT
```

back   Home

```
zpropair
prodini


echo off
% estimation of groups of data
%             prediction with groups of data
%
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zpropair.m
ndat = 100;                          % length of data
ncom = 2;                             % number of components
cove = ltdl([0.5 0.05; 0.05 0.5]);    % noise covariance
Sim  = statsim(0, ncom, cove);        % build simulator


global CUMTAB ACTIVE
CUMTAB = [0.97 0.03; 0.03 0.97];      % Markov component jumps


DATA   = zeros(2, 2*ndat);            % pre-allocate data
mixsimul(Sim, 2*ndat);                % get data sample


sub = 130;
sub = sub + 1; subplot(sub);
[x,y,z] = mixgrid(Sim); contour(x,y,z,15); grid on
legend('Simulator');


% split data to test and learning part
Data = DATA(:,ndat+1:2*ndat);         % data for prediction
DATA = DATA(:,1:ndat);                % data for learning


nsk  = 2;                             % number of data skipped
preproc({'group', nsk});              % reorganize data
[nchn, ndat] = size(DATA)             % dimensions
nchn =
     4
ndat =
    99
```

```
% estimation
Mix0   = genmixe(2);                    % initial mixture for estimation
niter  = 10;                            % maximum number of iterations
frg    = 1;                             % no forgetting
method = 'q';                           % iterative estimation method
Mix = mixest(Mix0, frg, ndat, niter, method); % iterative estimation


sub = sub + 1; subplot(sub);
[x,y,z] = mixgrid(Mix); contour(x,y,z,15); grid on
legend('Est. mixture');


% prediction
DATA = Data;
preproc({'group', nsk});                % reorganize data
pchns = 1;                              % predicted channel
cchns = [3 4];                          % channels in condition
pMix  = mix2pro(Mix,pchns,cchns);       % predictor
for TIME = 1:ndat                       % getting prediction trajectory
   [Eths, coves, alphas] = profix(pMix);  % prediction
   yp(TIME) = Eths * alphas';
echo off


% results
  dd  = DATA(pchns,:);
  ee  = dd-yp;
  s2d = std(dd')
s2d =
    0.9476
  s2e = std(ee')
s2e =
    0.9261


sub = sub + 1; subplot(sub);
hist(ee,50); grid on                     % histogram of prediction error
legend('Pred. error');
```

back Home

```
zprotut
prodini


echo off
% mixture prediction
%             tutorial example
%
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zprotut.m
cove = ltdl([0.1 0.01; 0.01 0.1]);     % common component noise variance
ncom = 4;                              % number of components
Mix  = statsim(10, ncom, cove);        % build static ARX LS mixture
Mix.dfcs                               % degree of freedom of components
ans =
    0.1000    0.2000    0.3000    0.4000


sub = 221; subplot(sub);
[x,y,z] = mixgrid(Mix); contour(x,y,z,15); grid on
legend('Static mixture');


pchns = 1;                             % predicted channel
cchns = 2;                             % channel in condition
pMix  = mix2pro(Mix, pchns, cchns);    % build mixture predictor
pMix.states
ans =
      predicted: 1
    incondition: 2
         mixstr: [2x3 double]


% any of the datavectors below
val  = 0.5;                            % rawdata values
str  = [2;0];


pMix1 = profix(pMix, val, str);        % get prediction
pMix1.states                           % predictor states
ans =
      predicted: 1
    incondition: 2
         mixstr: [2x3 double]


sub = sub + 1; subplot(sub);
[x,y] = mixgrid(pMix1); plot(x,y); grid on
legend(['val=', num2str(val)]);


val  = 0;                               % zero-delayed data vector
pMix1 = profix(pMix, val, str);         % get prediction


sub = sub + 1; subplot(sub);
[x,y] = mixgrid(pMix1); plot(x,y); grid on
legend(['val=', num2str(val)]);


val  = 2;                               % zero-delayed data vector
% build mixture prediction
[Eths, coves, weights] = mixpro(Mix, pchns, cchns, val, str);
weights
weights =
    0.0000    1.0000    0.0000    0.0000
[x,y] = statgrid(Eths,coves,weights);   % plot coordinates


sub = sub + 1; subplot(sub);
plot(x,y); grid on
legend(['val=', num2str(val)]);
setaxis([222:224]);


plt('zprotut');
EDIT
```

back Home

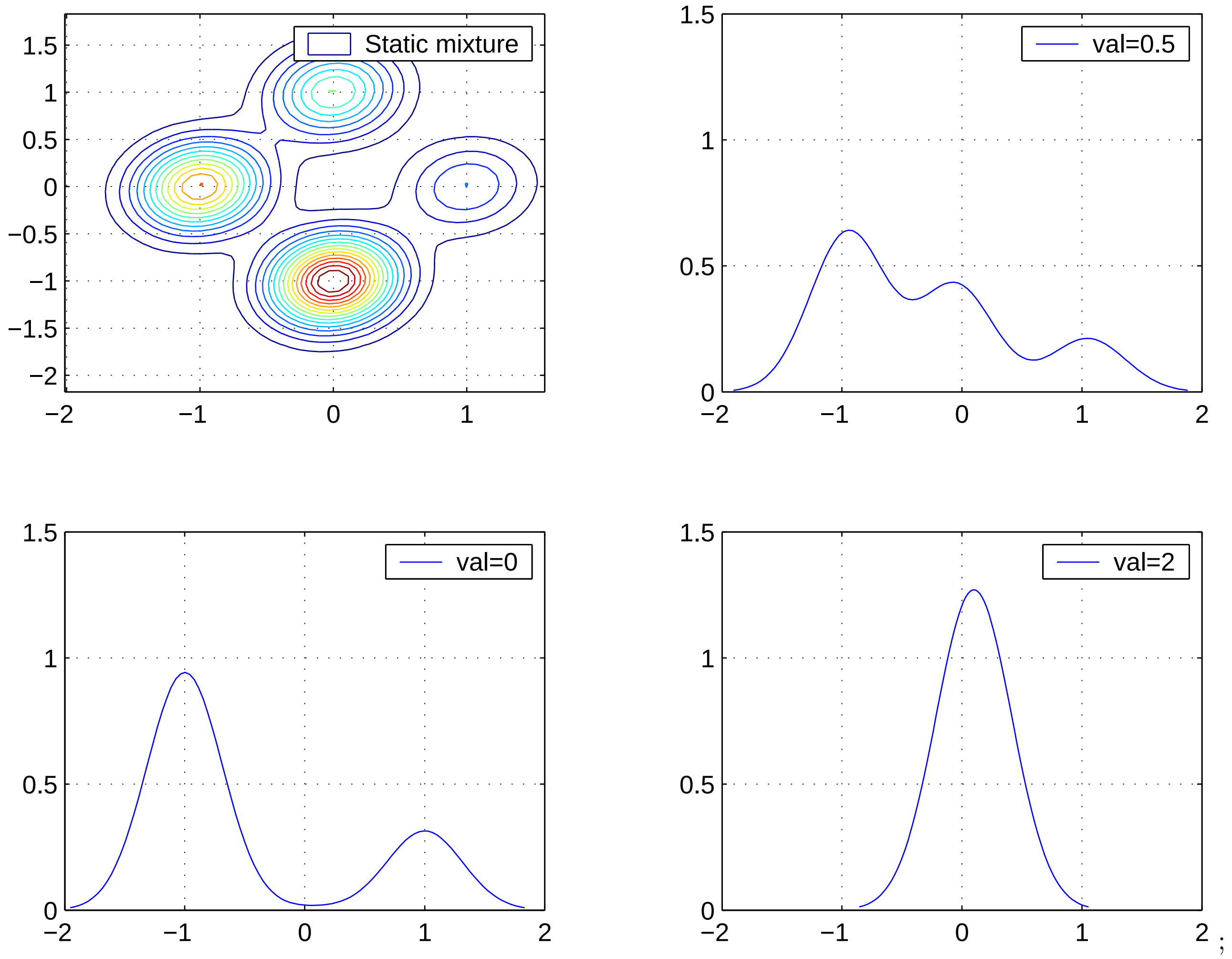


Figure 2.5:

```
zprotut1
prodini


echo off
% mixture prediction
%              tutorial example
%
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zprotut1.m
randn('seed', 3);
cove = ltdl([0.1 0.01; 0.01 0.1]);     % common component noise variance
ncom = 4;                              % number of components
ndat = 500;
statsim(ndat, ncom, cove);             % build static ARX LS mixture
preproc({'scale', [5 7; 2 3]});        % transform data


sub = 220;
sub = sub+1; subplot(sub);
datascan;
legend('Data clusters')
grid on; drawnow


Mix0  = genmixe(ncom);                 % initial mixture
pre   = preinit({'scale',[]});         % scaling list
Mix0  = fltpre(Mix0, pre);             % set scalling filters
Mix   = mixestim(Mix0, 1, ndat);       % estimate mixture
sub = sub + 1; subplot(sub);
[x,y,xy] = mixgrid(Mix); contour(x,y,xy,15); grid on
legend('estimated mixture');


pchns = 1;                             % predicted channel
cchns = 2;                             % channel in condition
pMix  = mix2pro(Mix, pchns, cchns);    % build mixture predictor
str   = [cchns;0];
val   = 22;                            % data vector
pMix1 = profix(pMix, val, str);        % get prediction
```

```
sub = sub + 1; subplot(sub);
[x,y] = mixgrid(pMix1); plot(x,y); grid on
legend(['val=', num2str(val )]);


val  = 24;                              % zero-delayed data vector
pMix1 = profix(pMix, val, str);         % get prediction


sub = sub + 1; subplot(sub);
[x,y] = mixgrid(pMix1); plot(x,y); grid on
legend(['val=', num2str(val)]);
```

back   Home

```
zshrink
prodini


echo off
% reducing component parameters
%
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zshrink.m
str   = [1    2    1    2    3    0;
         1    1    2    2    2    1];
Eth   = [1.1:0.1:1.6; 2.1:0.1:2.6; 3.1:0.1:3.6];
[nychn, npsi] = size(Eth);
cove  = ltdl(  10*eye(nychn) + rand(nychn,nychn) );
Cth   = ltdl(  10*eye(npsi)  + rand(npsi,npsi)   );
ychns = 1:nychn;
Com   = matarxls(ychns, str );


Eth = [1.1  0    1.3  0    1.5  0
       0    2.2  2.3  2.4  0    2.6
       3.1  3.2  0    3.4  3.5  0   ];
cove = diag([11 22 33]);


Com.Eth = Eth; Com.cove=cove; Com.Cth = Cth;
prt('ARX component',Com);
ARX component
ychns       :    1  2  3
str         :    4  5  6  7  8  9
dfm         :    1
Eth         :
    1.1  0  1.3  0  1.5  0
    0  2.2  2.3  2.4  0  2.6
    3.1  3.2  0  3.4  3.5  0
diag(Cth)   :    10.1886  10.0877  10.5763  10.7492  10.4047  10.8079
cove        :
    11  0  0
    0  22  0
    0  0  33
```

```
Com1    = mat2arx(Com);
comprt(Com1)
... component type 12 ...
comstr
    1   2   3   1   2   1   2   3   0
    0   0   0   1   1   2   2   2   1
ychns    1   2   3
str
--- cell 1 ---    2   3   4   5   6   7   8   9
--- cell 2 ---    3   4   5   6   7   8   9
--- cell 3 ---    4   5   6   7   8   9
Eth
--- cell 1 ---    0   0   1.1   0   1.3   0   1.5   2.77556e-017
--- cell 2 ---    0   0   2.2   2.3   2.4   0   2.6
--- cell 3 ---    3.1   3.2   -6.93889e-018   3.4   3.5   0
cove    11   22   33


Com2 = comshrink(Com1)
Com2 =
        Facs: {[1x1 struct]  [1x1 struct]  [1x1 struct]}
      comstr: [2x9 double]
        Flts: {[0]   [0]   [0]   [0]   [0]   [0]   [0]   [0]   [0]}
        type: 12
     rawdata: [0  0  0  0  0  0  0  0  0]
    datavect: [0  0  0  0  0  0  0  0  0]
prt(Com2)
... component type 12 ...
comstr
    1   2   3   1   2   1   2   3   0
    0   0   0   1   1   2   2   2   1
ychns    1   2   3
str
--- cell 1 ---    4   6   8
--- cell 2 ---    5   6   7   9
--- cell 3 ---    4   5   7   8
Eth
--- cell 1 ---    1.1   1.3   1.5
--- cell 2 ---    2.2   2.3   2.4   2.6
--- cell 3 ---    3.1   3.2   3.4   3.5
cove    11   22   33
```

back   Home

```
zsimbldl
prodini


echo off
% Simulation
%
% Design : P. Nedoma
% Updated: February 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% zsimbldl.m
randn('seed', 987);
ychn = 1;                              % output channel
uchn = 2;                              % input  channel
comstr = [1 2  1 1 1 1 2 2;  0 0 1 2 3 4 3 4]; % component structure
str = [1 1 1 1 2 2;  1 2 3 4 3 4];     % common dynamic factor structure


Fac = facarxls(ychn, str, comstr);     % model of output channel


% 1st dynamic system
Fac.Eth  = [1.41833  -1.5894  1.3161  -0.88642  0.2826  0.50666];
Fac.cove = 0.39463^2;                  % variance of output noise
Facs{1}  = Fac;


Fac = facarxls(uchn, [], comstr);      % model of input channel for simulation
Fac.cove = 0.01;                       % variance of output noise
Facs{2}  = Fac;


Coms{1} = comarx(comstr, Facs);


% 2nd dynamic system
Fac      = facarxls(ychn,str,comstr);  % model of output channel
Fac.Eth  = [2.0968  -2.3196  1.9335  -0.8713  0.641  0.1041];
Fac.cove = 0.37255^2;                  % variance of output noise
Facs{1}  = Fac;


Fac = facarxls(uchn, [], comstr);      % model of input channel for simulation
Fac.cove = 0.01;                       % variance of output noise
```

```
Facs{2}  = Fac;
Coms{2}  = comarx(comstr, Facs);


% mixture simulator
Sim      = mixconst(Coms, [0.3 0.7]);     % mixture simulator


Mix = simbldl(Sim)
Mix =
      Coms: {[1x1 struct]  [1x1 struct]}
      dfcs: [0.3000 0.7000]
      mmod: 2
      type: 22
    states: [1x1 struct]



Mix
Mix =
      Coms: {[1x1 struct]  [1x1 struct]}
      dfcs: [0.3000 0.7000]
      mmod: 2
      type: 22
    states: [1x1 struct]
Mix.states
ans =
    cumprob: [0.3000 1]
C=Mix.Coms{2}
C =
        Facs: {[1x1 struct]  [1x1 struct]}
      comstr: [2x8 double]
        Flts: {[0]  [0]  [0]  [0]  [0]  [0]  [0]  [0]}
        type: 12
     rawdata: [0 0 0 0 0 0 0 0]
    datavect: [0 0 0 0 0 0 0 0]
      states: [1x1 struct]
C.states
ans =
    facs: [1 2]
F=C.Facs{1}
F =
      ychn: 1
       str: [3 4 5 6 7 8]
       dfm: 1
      type: 2
      cove: 0.1388
       Eth: [2.0968 -2.3196 1.9335 -0.8713 0.6410 0.1041]
       Cth: [6x6 double]
    states: [1x1 struct]
F.states
ans =
            fac: 1
           ychn: 1
            str: [3 4 5 6 7 8]
            Eth: [2.0968 -2.3196 1.9335 -0.8713 0.6410 0.1041]
           stde: 0.3725
         useCth: 0
           CthL: [6x6 double]
    sqrtDiagCth: [1000 1000 1000 1000 1000 1000]
          etype: 1
```

```
ztutord
prodini


echo off
% Tutorial on dynamic mixtures learning
% Design : P. Nedoma
% Updated: March 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% ztutord.m
randn('seed', 987);
ychn = 1;                               % output channel
uchn = 2;                               % input  channel
comstr = [1 2  1 1 1 1 2 2;  0 0 1 2 3 4 3 4]; % component structure
str    = [     1 1 1 1 2 2;      1 2 3 4 3 4]; % common dynamic factor structure


Fac = facarxls(ychn, str, comstr);      % model of output channel


% 1st dynamic system
Fac.Eth  = [1.41833  -1.5894  1.3161  -0.88642  0.2826  0.50666];
Fac.cove = 0.39463^2;                   % variance of output noise
Facs{1}  = Fac;


Fac = facarxls(uchn, [], comstr);       % model of input channel for simulation
Fac.cove = 0.01;                        % variance of output noise
Facs{2}  = Fac;


Coms{1} = comarx(comstr, Facs);         % ARX LS component


% 2nd dynamic system
Fac      = facarxls(ychn, str,comstr); % model of output channel
Fac.Eth  = [2.0968  -2.3196  1.9335  -0.8713  0.641  0.1041];
Fac.cove = 0.37255^2;                   % variance of output noise
Facs{1}  = Fac;


Fac = facarxls(uchn, [], comstr);       % model of input channel for simulation
Fac.cove = 0.01;                        % variance of output noise
Facs{2}  = Fac;
```

```
Coms{2}  = comarx(comstr, Facs);


% mixture simulator
Sim      = mixconst(Coms, [0.3 0.7]);     % mixture simulator


% get data sample
ndat = 300;                               % sample size
DATA = zeros(2, ndat);                    % pre-allocated DATA
Sim = mixsimul(Sim, ndat);                % build data


sub = 220;
sub = sub + 1; subplot(sub);
[x,y,z] = mixgrid(Sim); contour(x,y,z,15); grid on
H = legend('Simulator', 0);


sub = sub + 1; subplot(sub);
datascan([2;1], 1);                       % scatergram of data
H = legend('Data clusters', 0);
xlabel('1st channel');
ylabel('2nd channel')


% mixture initialization
% richest mixture
maxstr   = [1 1 1 1 1 1  2 2 2 2  0
            1 2 3 4 5 6  3 4 5 6  1];
Mix0     = genmixe(1,[ychn uchn], maxstr);
                                          % number of iterations
frg   =  1;
Mix   = mixinit(Mix0, frg, ndat, 2); % mixture initialization


ncom = length(Mix.dfcs)                   % number of components
ncom =
     4
Mix.dfcs/sum(Mix.dfcs)                    % component weights
ans =
    0.2184    0.1666    0.4676    0.1474


M=mix2mix(Mix, 22);
```

```
prt(M.Coms{1});
... component type 12 ...
comstr
    1  2  1  1  1  1  1  1  2  2  2  2  0
    0  0  1  2  3  4  5  6  3  4  5  6  1
ychns    1  2
str
--- cell 1 ---    2  3  4  5  6  7  8  9  10  11  12  13
--- cell 2 ---    3  4  5  6  7  8  9  10  11  12  13
Eth
--- cell 1 ---    0.739243  1.89716  -2.08468  1.7255  -0.836947  -0.0484779  0.0679865  0
--- cell 2 ---    -0.00196641  -0.0186497  0.0383873  -0.0465348  0.038754  -0.0244584  0.
cove    0.492125  0.00933523


sub=sub+1; subplot(sub);
[x,y,z] = mixgrid(Mix); contour(x,y,z,15); grid on
H = legend('Initialized mixture, time 500');


% iterative estimation after initialization
Mix0  = mixflat(Mix);                  % mixture flattenning
niter = 5;                             % number of iterations
Mix   = mixest(Mix0, frg, ndat, 10);   % iterative mixture estimation


sub=sub+1; subplot(sub);
[x,y,z] = mixgrid(Mix); contour(x,y,z,15); grid on
H = legend('Estimated mixture, time 1500');
```

back Home

```
ztutors
prodini


echo off
% tutorial on static mixture simulation and learning
%
% Design : P. Nedoma
% Updated: March 2005
% Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
%          MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
% ztutors.m
randn('seed',135531);


%  data
ndat = 500;                             % size of data sample
cove = [0.1 0.01; 0.01 0.1];            % component noise variance
ncom = 4;                               % number of components


% build ARX LS component
ychns    = [1 2];                       % output channels
str      = [0;1];                       % common static structure
comstr   = [ [1 2; 0 0], str ] ;
Com      = matarxls(ychns, str, comstr); % ARX component
Com.cove = ltdl(cove);                  % common cove


% array of components for simulation
Eths     = [1 0 -1 0; 0 1 0 -1];        % mean of components (columns-wise)
for i=1:ncom                            % build array of components
    Com.Eth = Eths(:,i);
    Coms{i} = Com;
    Com.Eth = Eths(:,i);
    Coms{i} = Com;
    Com.Eth = Eths(:,i);
    Coms{i} = Com;
    Com.Eth = Eths(:,i);
    Coms{i} = Com;
end
% build mixture simulator
dfcs = 1:ncom;                          % degree of freedom of components
Sim  = mixconst(Coms, dfcs);            % build simulator
```

```
% generate data
DATA    = zeros(2, ndat);              % pre-allocate data
mixsimul(Sim, ndat);                   % get data sample


% visualization
[x,y,z] = mixgrid(Sim);                % get coordinates
clf; hold off; meshc(x,y,z);
xlabel('1st channel'); ylabel('2nd channel');
H = legend('Simulated mixture (pdf)',0);
pause


sub = 230;
sub=sub+1; subplot(sub);
contour(x,y,z,15); grid on
xlabel('1st channel'); ylabel('2nd channel');
H=legend('Simulator',0);
drawnow


sub = sub+1; subplot(sub);
datascan([],1);
H=legend('Data scattergram');
drawnow


% build initial mixture
Mix0 = genmixe;


% make mixture initialization
niter = 3;                             % maximum number of components (iteratios)
frg   = defaults('frg');               % default forgetting rate
Mix   = mixinit(Mix0, frg, ndat, niter); % mixture initialization
mixlls(1) = Mix.states.mixll;


sub=sub+1; subplot(sub);
[x,y,z] = mixgrid(Mix); contour(x,y,z,15);
xlabel('1st channel'); ylabel('2nd channel');
title('Initialized mixture'); grid on; drawnow
```

```
% iterative estimation after initialization
Mix0 = mixflat(Mix);                     % mixture flattenning
Mix  = mixest(Mix0, frg, ndat, 5);       % iterative mixture estimation
mixlls(2) = Mix.states.mixll;


sub = sub+1; subplot(sub);
[x,y,z] = mixgrid(Mix); contour(x,y,z,15);
xlabel('1st channel'); ylabel('2nd channel');
title('Estimated mixture'); grid on
drawnow


% standalone estimations
Mix0  = genmixe(4);
niter = 10;                              % maximum number of iterations
frg   = defaults('frg');                 % default forgetting rate
Mix   = mixest(Mix0,frg,ndat,niter);     % iterative mixture estimation
mixlls(3) = Mix.states.mixll;


sub=sub+1; subplot(sub);
[x,y,z] = mixgrid(Mix); contour(x,y,z,15);
xlabel('1st channel'); ylabel('2nd channel');
title('Direct estimation'); grid on


sub = sub+1; subplot(sub);
plot(mixlls, 'o'); grid on
hold on; plot(mixlls, ':');
title('v-log-likelihood');
```

back Home

diary off

Chapter 3

Termbase

TIME processing time

DATA data sample

---

ndat length of data

Ndat specification for buffered processing

---

psi regressor vector

Psi data vector

---

npsi length of regression vector

---

nPsi length of data vector

---

str structure of regression vector

Fac factor

Facs array of factors

---

fac position of a factor in an array of factors

---

ychn modeled channel

dfm degrees of freedom of a factor

LD L'DL decomposition of the extended information matrix

L triangular part of L'DL decomposition

D diagonal part of L'DL decomposition of extended information matrix

Eth point estimate of regression coefficients

Cth covariance of regression coefficients

cove point estimate of noise covariance

FacA alternative factor used for stabilizing forgetting com component

coms array of components

dfcs vector of degrees of freedom of components

dfcs0 initial degrees of freedom of components

alphas normalized vector of degrees of freedom of components

Com component

chns list of channels

Coms array of matrix ARX or ARX LS components ychns modeled channels in component

nychn number of modeled channels

comstr component structure

datavect component filterred data

rawdata data extracted via comstr

weight probabilistic weight

Mix mixture estimate

Sim mixture simulator

pMix mixture predictor

pMixfix mixture prediction

facs list of factors

nfac number of active factors

ncom number of components

nchn number of modeled channels

frg forgetting rate

frgd default forgetting rate

---

rate mixture flattening rate

---

maxtd maximum time delay of factors in a mixture

---

nruns number of runs in iterative mixture estimation

---

relerr relative error

---

maxerr maximum possible error

---

pchns predicted channels

---

cchns channels in condition

---

psi0 value of zero-delayed regressor

---

nsk extent of data grouping

---

faclls virtual factor predictions

---

comlls component predictions

---

mixll posterior data likelihood (mixture prediction)

---

comwgs component weights

---

facwgs factor weights

---

maxstr guess of the richest structure

---

maxFac richest factor

maxMix richest mixture

belief belief on a guess of richest structure

chbelief belief on factors of a channel

nbest number of "best" MAP structures stored

nrep number of random starts

irep iteration

MAPstr MAP estimate of the factor structure

pri list of prior knowledge

---

pre preprocessing requirements

---

aMixc advised mixture of the type ARX LS + control states

---

aMixu desired mixture of the type ARX LS + control states

---

strc common control structure

---

ufc normalised vector qualifying components

---

kc lift of quadratic forms

UDc cell vector of u'du decompositions of KLD kernels

---

udca u'du decomposition of average KLD kernel in UDc kca average lift of quadratic forms kc

uchn list of channels with recognisable actions

pochn list of channels with o-innovations

outs list of channels with innovations

npochn number of channels with o-innovations

chis strategy of control design

DEBUG global debugging flag

PLOTNO control results of iterative computation

chn	channel (data row)
std	standard deviation
pdf	probability density function
ll	log of posterior likelihood on data: v-log-likelihood
kld	Kullback-Leibler distance
niter	number of iterations
options	computational options
seed	seed of random generator

---

sig standard deviation of output noise

Can component in matrix factorized ARX LS form

Cans array of components in matrix factorized ARX LS form

CUMTAB transition table of components

ACTIVE active component

Chns channel descriptions

---

type object (structure) type

---

typ structure type

---

dvect data vector

---

jacob jacobian of transformation

---

args additional arguments

---

mode mode of processin

---

arx2arx conversion between ARX and ARX LS representations

```
X = arx2arx(X)
X: factor, component, mixture: recognised by the associated type
                               (types are described in the Guide)
Design : P. Nedoma
Updated: June 2001
Project: ProDaCTools
Project: GA CR 102/03/0049, AV CR S1075351, S1075102, DESIGNER
```

---

arx2can convert ARX LS component into matrix factorized ARX LS

```
Facs : factorized component type 12 or 112
Can  : matrix factorized component
ok   : 1: the factors were already in the form required
     : 0: transformation was made because not "
Design : P. Nedoma
Updated: September 2003
Project: GA CR 102/03/0049, AV CR S1075351, S1075102, DESIGNER
```

---

arx2mat convert ARX or ARX LS component into matrix ARX LS

```
Com1 = arx2mat(Com)
Com1 : matrix ARX LS component
Com  : array of ARX or ARX LS factors
```

---

can2marg re-organize matrix factorized component

```
Can = can2marg(Can, pchns)
Can   : matrix factorized ARX LS component
pchns : marginal (predicted) channels
Design : P. Nedoma
Updated: March 2001
Project: GA CR 102/03/0049, AV CR S1075351, S1075102, DESIGNER
```

---

can2mat Mixtools can2arxc

```
convert matrix factorized ARX LS component into matrix ARX LS component
Com = can2arxc(Can)
Com : matrix factorize ARX or ARX LS component
Design : P. Nedoma
Updated: July 2003
Project: GA CR 102/03/0049, AV CR S1075351, S1075102, DESIGNER
```

---

com2com convert component to a specified form

```
Com1 = com2com(Com, form)
Com  : component to be converted
form : coded component form:
       11 - ARX
       12 - ARX LS
       13 - matrix ARX
       14 - matrix ARX LS
       + 100 for predictor component
Com1  : resulting component
Design : P. Nedoma
Updated: July 2003
Project: GA CR 102/03/0049, AV CR S1075351, S1075102, DESIGNER
```

---

com2fcom convert Mixtools ARX LS component to filter versi

```
 fCom = com2fcom(Facs)
Facs   : Mixtools ARX or ARX LS component
fCom   : filtered version component
Design : P. Nedoma
Updated: June 2003
Project: ProDaCTools remake
```

---

com2marg data-marginal component projection

```
Com1 = com2comm(Com, pchns)
Com1 = com2comm(Com) predicted channels coincide with modelled ones
Com1     : component predictor
Com      : component of any form, preferably 12, 112
pchns    : marginal channels
Design   : P. Nedoma
Updated : November 2003
Project: GA CR 102/03/0049, AV CR S1075351, S1075102, DESIGNER
```

---

com2pro convert ARX LS component into predictor (11

```
 Com1 = com2pro(Com, pchns, cchns)
 Com1 = com2pro(Com, pchns)  no channels in condition
 Com1 = com2pro(Com)         all modelled channels are predicted
 Com1  : predictor component, type 112
 Com   : component, preferably ARX LS
 pchns : predicted channels
 cchns : channels in condition
```

---

comarx build ARX component

```
Com = comarx(comstr, Facs)
Com     : ARX or ARX LS component
comstr  : component structure
Facs    : component factors
Design  : P. Nedoma
Updated : August 2003
Project: GA CR 102/03/0049, AV CR S1075351, S1075102, DESIGNER
```

---

comdata Com = comdata(Com, Psi, st

```
Com: updated component
Psi: changes of Com.rawdata
str: structure describin the Psi elements
Design : P. Nedoma
Updated: February 2005
Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
         MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
```

---

commerge optimum merging of mixture components

```
  or possibly with enforced range
[Mix, Mix0, changed] = commerge(Mix, Mix0, nlow, nhigh)
[Mix, Mix0, changed] = commerge(Mix, Mix0, nlow) nhigh = nlow
[Mix, Mix0, changed] = commerge(Mix, Mix0)       range unspecified
Mix     : merged estimated mixture
Mix0    : merged initial mixture
changed : 0 - mixture not changed, otherwise 1
Mix     : estimated mixture
Mix0    : initial mixture
nlow    : lower bound on number of components to be preserved
nhigh   : upper bound on number of components to be preserved
Note    : merging is made when there is a chance for increasing of
          v-likelihood; it acts as counterpart of mixsplit
Design  : M. Karny, P. Nedoma
Updated : September 2002
Project : ProDaCTools
See also:  mixinit , mixsplit , mixcut
```

---

comprt help comments found in comprt.

---

comshrink rebuild component

```
regression coefficients below "eps" are cut
Com = comshrink(Com)
Com ; updated ARX LS component
Design : P. Nedoma
Updated: February 2005
Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
         MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
```

---

comupdt componet update

```
Com = comupdt(Com, weight)
Com   : updated component
weight: component weight
Design : P. Nedoma
Updated: February 2005
Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
         MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
this function has no mex equivalent
```

---

estbldl normal mixture constructor

```
Mix = estbldl(Mix)
Mix     : updated estimator
Author  : P. Nedoma
Updated : August 2003
Project: GA CR 102/03/0049, AV CR S1075351, S1075102, DESIGNER
```

---

facarx build ARX-factor

```
Fac = facarx(ychn, str, comstr, default)
Fac = facarx(ychn, str, comstr) 'A'
Fac     : ARX factor, type = 1
ychn    : modeled channel
str     : factor structure
comstr : component structure
default: defaults
Design  : P. Nedoma
Updated : July 2003
See also:  facarxls
```

---

facarxls build ARX LS factor

```
Fac = facarxls(ychn, str, comstr, default)
Fac = facarxls(ychn, str, comstr) 'A'
Fac     : ARX factor, type = 2
ychn    : modeled channel
str     : factor structure
comstr : component structure
default: defaults
Design  : P. Nedoma
Updated : July 2003
See also:  facarxls
```

---

facflat factor flattening

```
Fac0 = facflat(Fac, rate, FacA)
Fac0 = facflat(Fac, rate)         without stabilizing forgetting
Fac0    : flattened factor
Fac     : estimated factor
rate    : factor flattening rate
FacA    : alternative flattening mixture
Design  : M. Karny, P. Nedoma
Updated : January 2003
Project : ProDaCTools cnt'd
See also:  mixflat , mixflatv
Note    : function of the construction base, the default rate is
          computed in the caller function
```

---

facfrg factor forgetting

```
Fac = facfrg(Fac, rate, Faca)
Fac = facfrg(Fac, rate) no stabilized forgetting
Fac  : updated factor
rate : forgetting rate
FacA : stabilized forgetting
Design : P. Nedoma
Updated: April 2003
Project: ProDaCTools remake
```

---

facstr factor-structure estimation

```
[optstr, lhs] = facstr(Fac, Fac0, belief, nbest, max-nrep, uchns, lambda, order-k)
[optstr, lhs] = facstr(Fac, Fac0, belief, nbest, max-nrep, uchns, lambda) order-k=2
[optstr, lhs] = facstr(Fac, Fac0, belief, nbest, max-nrep) uchns) lambda = 0.9
[optstr, lhs] = facstr(Fac, Fac0, belief, nbest, max-nrep) uchns = []
[optstr, lhs] = facstr(Fac, Fac0, belief, nbest) max-nrep  = 100 !!!!! change this to 500
[optstr, lhs] = facstr(Fac, Fac0, belief) nbest = 1
[optstr, lhs] = facstr(Fac, Fac0) belief = 2
                              defaults generated by function defaults with the option 's'
optstr: maximum a posteriori probability estimate of factor structure
% lhs :  cell vector of the best regressors -- not true now

Fac0     : initial factor  type = 1
belief   : user's belief on maximum structure items
          (1 items must     be present, 2 items are probably     present
           4 items must not be present, 3 items are probably not present)
nbest    : how many "best" regressors are maintained
max-nrep : maximal number of random starts in search for the best
            structure
uchns    : list of input channels - if specified, the resulting structure
          contains at least one of inputs (suboptimal solution)
           Note: channel description can be used instead of "uchns"
lambda   : stooping rule threshold
order-k  : order of k
Design  : L. Tesar. Interface Based on P. Nedoma's previous version of facstrid.
Updated : 14.4.2003 - 10.9.2003, MK July 2004
Project : post-ProDaCTool
Reference: straux1
```

---

facupdt update factor statistics

```
Fac = facupdt(Fac, dvect)
Fac = facupdt(Fac, dvect, weight)
Design  : P. Nedoma
Updated : April 2003
Project : ProDaCTools remake
```

---

facvll comput data log-likeliho

```
 vll = facvll(Fac, Fac0)
 Fac : estimated factor
 Fac0: initial factor - if not specified, a relative vll is computed
Design : P. Nedoma
Updated: June 2003
Project: ProDaCTools remake
```

---

filters data filt

```
 [Com, out] = filters(Com, fac, mode)

 mode==1:  - update Com.datavect
           - out: logarithm of jacobian
 mode==2:  - update filter states
 mode==3:  - update Com.datavect
           - out: jakobian of logarithm of  jakobian of inverze transformation

 Individual filters
 1     scaling    Flt = fltconst( add, mul ) set by
                  Mix = fltpre(Mix, pre)
```

---

fltconst filter constructor

```
Flt = fltconst(typ, varargin)
typ      : filter type
varargin : constructor arguments
0       : no action
1: --- scaling ---
add     : added value
mul     : multiplied by value
99: --- exp ---
```

---

fltpre add scaling facto

```
 Mix = fltpre(Mix0, pre)
 Mix0 : any mixture
 pre  : preprocessing list
Design : P. Nedoma
Updated: February 2005
Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
         MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
```

fltset set mixture filterri

```
 Mix = fltset(Mix0, operation. arguments)
 Mix  : updated mixture (predictor-
 Flt  : filter
 typ  : type of operation
        1 - set filter to a channel for all components
        a1: channel
Design : P. Nedoma
Updated: February 2005
Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
         MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
```

genmixe generates mixture for identification (type 24)

```
Mix = genmixe(ncom, ychns, str,  ndat, diagCth, diagcove, dfm)
Mix = genmixe(ncom, ychns, str,  ndat, diagCth, diagcove) dfm=100
Mix = genmixe(ncom, ychns, str,  ndat, diagCth)     diagcove=0.01
Mix = genmixe(ncom, ychns, str,  ndat)              diagCth=1
Mix = genmixe(ncom, ychns, str)                     ndat=size(DATA,1)
Mix = genmixe(ncom, ychns)                          str=[0;1]
Mix = genmixe(ncom)                                 ychns=1:size(DATA,1)
Mix = genmixe;                                      ncom = 1;
ncom   number of components
ychns  modeled channels
str    component structure
nchn   number of channels
ndat   size of data sample
diagCth, diagcove  setting of Cth and cove
Design : P. Nedoma
Updated: January 2002
Project: ProDaCTools
```

---

getdvect get raw data vect

```
 dvect  = getdvect(Fac)
 dvect : component data vector
 str   : component structure in the form channels;delays
 Design  : P. Nedoma
 Updated : June 2003
 Project : ProDaCTools remake
```

---

getpsi0 get values from ps

```
 val = getpsi(psi0, chtd)
 val  : extracted value
 psi0 : coded values of any
 chtd : structure item
```

---

isstatic check whether the mixture is static one

```
is = isstatic(Mix)
Mix : inspected mixture
is  : 1 - mixture is static
      0 - mixture is dynamic
Design : P. Nedoma
Updated: April 2001
Project: ProDaCTools
verloaded methods
  help frd/isstatic.m
  help lti/isstatic.m
  help ss/isstatic.m
  help tf/isstatic.m
  help zpk/isstatic.m
```

---

iterplot help comments found in iterplot.

---

mat2arx convert matrix component into factorized component

```
Coma = mat2arx(Com, flag)
Com    : matrix component
Coma   : component (of the same ARX type)
flag   : if present,  zeros in Eth are not eliminated
Design : P. Nedoma
Updated: July 2003
Project: GA CR 102/03/0049, AV CR S1075351, S1075102, DESIGNER
```

---

matarx build matrix ARX component

```
Com = comarx(ychns, str, comstr, default)
Com     : matrix ARX component, type = 13
ychns   : modeled channels
str     : component regressor structure
Design  : P. Nedoma
Updated : July 2003
Project : ProDaCTools
See also:  comarxls , facarx , facarxls
```

---

matarxls build matrix ARX LS component

```
Com = comarxls(ychns, str, comstr, default)
Com     : matrix ARX LS component, type = 14
ychns   : modeled channels
str     : component structure
Design  : P. Nedoma
Updated : August 2003
Project: GA CR 102/03/0049, AV CR S1075351, S1075102, DESIGNER
```

---

mix2marg build data-marginal project

```
 pMix = mix2mixm(Mix, pchns)
 pMix = mix2mixm(Mix)   predicted channels coincide with modelled ones
 pMix    : mixture predictor
 Mix     : mixture, any form, preferably 22, 122
 pchns   : predicted channels
 Design  : P. Nedoma
 Updated : November  2003
 Project: GA CR 102/03/0049, AV CR S1075351, S1075102, DESIGNER
```

---

mix2mix convert mixture to a specified form

```
Mix = mix2mix(Mix, form)
Mix     : mixture of any form
form    : coded form 21 22 23 or 24; (+100 for estimator)
Author  : P. Nedoma
Updated : July 2003
Project: GA CR 102/03/0049, AV CR S1075351, S1075102, DESIGNER
```

---

mix2mix0 create initial mixture dimensioned as a pattern

```
Mix0 = mix2mix0(Mix, dfcs)
Mix0 = mix2mix0(Mix)       dfcs are set to default values
Designed: P. Nedoma
Updated : July 2000,  MK August 2001, August 2004
Project : post-ProDaCTools
Calls   : defaults, facarx, facarxl, mixconst
```

---

mix2pro make/re-build mixture projector

```
pMix = mix2pro(Mix, pchns, cchns)
pMix = mix2pro(Mix, pchns)          no channels in condition
pMix = mix2pro(Mix)        all modelled channels predicted and
                           no channels in conditions
pMix    : mixture predictor
Mix     : mixture or p-mixture, any form
pchns   : modelled channels
cchns   : channels in condition
Design  : P. Nedoma
Updated : June 2001
Project : ProDaCTools
```

---

mixconst normal mixture constructor

```
Mix = mixconst(Coms, dfcs)
Mix     : mixture or p-mixture
Coms    : array of components
dfcs    : degrees of freedom of components
Author  : P. Nedoma
Updated : July 2003
Project : GA CR, AV
```

---

mixdata 1. Com.rawdata are extracted from DATA according to Com.comstr

```
2. psi value is copied to Com.rawdata according to str
```

---

mixdfms sum of dfm and dfm0 of mixture factors

```
[s, s0] = mixdfms(Mix)
Mix : mixture
s   : sum of dfms
Design : P. Nedoma
Updated: August 2003
Project: GA CR 102/03/0049, AV CR S1075351, S1075102, DESIGNER
```

mixest iterative Bayesian ARX mixture estimation

```
LHS = mixest(Mix0, frg, ndat, niter, method)
LHS = mixest(Mix0, frg, ndat, niter)          method = quasi-Bayes (q)
LHS = mixest(Mix0, frg, ndat)                 niter  = 1 for q,f; 10 for b
LHS = mixest(Mix0, frg)                       ndat   = lenght(DATA)
LHS = mixest(Mix0)                            frg    = defaults
LHS    : Mix or [Mix, mixlls]
Mix    : estimated ARX mixture,           type = 21
Mix0   : initial estimate of ARX mixture of any type
mixlls : trajectory of mixll, i.e. v-log-likelihood of the mixture
         (used for experiments only)
frg    : forgetting rate (alternative forgetting for method (f))
ndat   : size of data sample or data sample (copied to global DATA)
niter  : number of iterations used in iterative methods
method : character string, the 1st significant character means:
         'q' : iterative       quasi-Bayes mixture estimation
         'b' : iterative batch quasi-Bayes mixture estimation
         'f' : iterative mixture estimation with forgetting branching
  if the string 'method' 2nd column is 's' : states are
     evaluated by mixstats
Design  : P. Nedoma
Updated : October 2001
Project : ProDaCTools
See also:  mixestqb , mixestq , mixestbb , mistats
Note: DEBUG mechanism is exploited for inspecting intermediate results
      DEBUG is set in Prodini
```

mixestbq iterative mixture estimate by batch quasi-Bayes estimation

```
function [Mix, mixlls] = mixestbq(Mix0, frg, ndat, niter)
LHS = mixestbq(Mix0, frg, ndat, niter)
LHS = mixestbq(Mix0, frg, ndat)          niter = 10
LHS = mixestbq(Mix0, frg)                ndat  = size of global DATA
LHS = mixestbq(Mix0)                     frg   = defaults('f')
LHS   : Mix or [Mix, mixlls]
Mix   : estimated mixture, type = 21
mixlls: trajectory of mixll, i.e. v-log-likelihood of mixture
Mix0  : initial mixture, any type
frg   : forgetting rate
ndat  : size of data sample or data sample (copied to global DATA)
niter : number of iterations
Note  : by setting global DEBUG>1 the initial and estimated mixtures
        are displayed in each iteration
Design  : P. Nedoma
Updated : October 2001
Project : ProDaCTools
See also:  mixestqb , mixest , mixestim , mixestbb
```

---

mixestee quasi-Bayes mixture estimation fitting two-component mixtures

```
to filtration errors of all factors
[Mix, Mixs, Mixs1] = mixestee(Mix0, frg, ndat)
[Mix, Mixs, Mixs1] = mixestee(Mix0, frg)       ndat = sample size
[Mix, Mixs, Mixs1] = mixestee(Mix0)            frg = defaults('frg');
Mix   : estimated mixture
Mixs  : cell vector of mixtures fitted to filtration errors,
       each is static, having 2 different components
Mixs1: cell vector of mixtures fitted to filtration errors,
       each is static, having 2 identical components
Mix0 : initial mixture
frg   : forgetting rate
ndat : size of data sample
Design   : P. Nedoma
Updated : May 2002
Project : ProDaCTools
See also:  mixinit , issplit , facsplit
```

---

mixestfe auxiliary estimation for factor splitting in initialization

```
[Mix, Mixs, Mixs1] = mixestfe(Mix0, frg, ndat, Mixs, Mixs1);
mixture estimate + filtration error for each factor
called in covering function mixestee.m
Mix   : estimated mixture
Mixs : cell vector of mixtures, 2 components
Mixs1: cell vector of mixtures, 1 component
Mix0 : initial mixture
frg   : forgetting rate      (default: default forgetting rate)
ndat : size of data sample  (default: whole data sample)
Design : P. Nedoma
Updated: May 2002
Project: ProDaCTools
==========================================================*/
```

---

mixestim quasi-Bayes estimation of ARX mixtu

```
 Mix = mixestim(Mix0, frg, ndat, Mixa)
 Mix = mixestim(Mix0, frg, ndat)     % stabilized forgetting is not used

Mix0    : initial mixture
frg     : forgetting rate
ndat    : sample size
Mixa    : alternative mixture for stabilized forgetting
Design  : P. Nedoma
Updated : January 2003
Project : ProDaCTools remake
```

---

mixestqb function [Mix, mixlls] = mixestqb(Mix0, frg, ndat, niter)

```
quasi-Bayes iterative estimation of normal mixture
  LHS = mixestqb(Mix0, frg, ndat, niter)
  LHS = mixestqb(Mix0, frg, ndat)    niter = 1
  LHS = mixestqb(Mix0, frg)          ndat  = length of global DATA
  LHS = mixestqb(Mix0)               frg   = defaults('frg')
LHS   : Mix or [Mix, mixlls]
Mix   : estimated mixture
Mix0  : initial mixture
mixlls: trajectory of mixll, i.e. log-likelihood of the mixture
frg   : forgetting rate
ndat  : size of data sample or data sample (copied to global DATA)
niter : number of iterations
Note  : by setting global DEBUG>1:
        in each iteration, start and end mixture is displayed
Design  : P. Nedoma
Updated : October 2001
Project : ProDaCTools
```

---

mixflat mixture flattening

```
   Mix0 = mixflat(Mix)
   Mix0 = mixflat(Mix, rate)
   Mix0 = mixflat(Mix, rate, rate1)
   Mix0 = mixflat(Mix, rate, rate1, MixA)
Mix     : input estimated mixture
Mix0    : flattened mixture
rate    : factor flattening rate
rate1   : component-weights flattening rate
MixA    : alternative flattening mixture
Design  : M. Karny, P. Nedoma
Updated : May 2001
Project : ProDaCTools
See also:  mixflatv
```

---

mixflatv mixture flattening with variable flattening rate

```
 [Mix0, handle] = mixflat(Mix, niter, ndat)    initialization
                                                (1st estimation step)
 [Mix0, handle] = mixflat(Mix, handle)         otherwise
Mix     : input estimated mixture
Mix0    : output flattened mixture
handle  : 1st step: No of iterations
          otherwise run time states (structure)
Design  : P. Nedoma
Updated : November 2001
Project : ProDaCTools
See also:  mixflat
```

---

mixfrg mixture forgetting

```
Mix = mixfrg(Mix, rate, rate1, Mixa)
Mix = mixfrg(Mix, rate, rate1)          flat "bar" alternative
Mix = mixfrg(Mix, rate)                 rate1 = rate
Mix   : forgotten mixture
Mix   : updated mixture
rate : factor flattening rate
rate1: component-weights forgetting rate
Mixa : stabilezed forgetting
Design  : P. Nedoma
Updated : March 2003
Project : ProDaCTools remake
```

---

mixgrid conversion of ARX mixture to pdf. coordinates

```
[x,y,z] = mixgrid(Mix, n, r)
[x,y,z] = mixgrid(Mix, n) r     = []
[x,y,z] = mixgrid(Mix)    n     = []
Mix     : mixture, any form
n       : grid density (densities)
          if empty: 100 for 1 dim, [50,50] for 2 dimensional case
r       : grid range (ranges)
          if empty: automatic value computed from means and noise covariances
Design  : P. Nedoma
Updated : August 2003
Project: GA CR 102/03/0049, AV CR S1075351, S1075102, DESIGNER
```

---

mixinit initialization of mixture estimation

```
[Mix, Mix0] = mixinit(Mix0,  frg,  ndat,  niter,  options,  belief)
[Mix, Mix0] = mixinit(Mix0,  frg,  ndat,  niter,  options)  belief is not used
[Mix, Mix0] = mixinit(Mix0,  frg,  ndat,  niter)  defaults are used
[Mix, Mix0] = mixinit(Mix0,  frg,  ndat)          niter=5
[Mix, Mix0] = mixinit(Mix0,  frg)                 size(DATA, 2)
[Mix, Mix0] = mixinit(Mix0)                       default forgetting rate
Mix    : estimated mixture, type 21
Mix0   : initial mixture, any type internally converted to 21
frg    : forgetting rate
ndat   : size of data sample
         it can be cell vector for buffered data processing
niter  : number of iterations
options: initialization options
         character string or character string followed an number
belief : cell vector of beliefs for individual channels
--- options for estimation ---
p : projection-based Bayes mixture estimation
q : iterative quasi-Bayes mixture estimation
b : iterative batch quasi-Bayes mixture estimation
f : iterative mixture estimation based on forgetting branching
m : iterative quasi-Bayes with fixed variances
n : number of iterations for iterative estimation, a number follows;
    default is 10 iterations
Note: if iterative estimation is not explicitly specified,
      mixestim is used
--- option for structure estimation ---
h : number of runs for structure estimation (integer follows)
--- options that modify processing ---
c : do not make the final housekeeping,                   default: make it
g : number of initial steps when all factors are split    default: 2
k : number of steps when components are not merged or erased, default: 1
See also:  commerge , mixerase , mixest
Design : M. Karny, P. Nedoma
Updated: September 2002
Project: ProDaCTools
```

---

mixmaxtd get maximum time delay in mixture

```
maxtd = mixmaxtd(Mix)
Design : P. Nedoma
Updated: January 2003
Project: ProDaCTools
```

---

mixmesh mixture contour plot

```
mixmesh(Mix, n, r)
mixmesh(Mix, n) r     = []
mixmesh(Mix)    n     = []
Mix     : mixture, any form
n       : grid density (densities)
          if empty: 100 for 1 dim, [50,50] for 2 dimensional case
r       : grid range (ranges)
          if empty: automatic value computed from means and noise covariances
Design  : P. Nedoma
Updated : August 2003
Project : GA CR 102/03/0049, AV CR S1075351, S1075102, DESIGNER
```

mixplot mixture contour plot

```
mixplot(Mix, n, r)
mixplot(Mix, n)   r     = []
mixplot(Mix)      n     = []
pMix    : mixture, any form, converted to 114 if necessary
n       : grid density (densities)
          if empty: 100 for 1 dim, [50,50] for 2 dimensional case
r       : grid range (ranges)
          if empty: automatic value computed from means and noise covariances
Design  : P. Nedoma
Updated : August 2003
Project: GA CR 102/03/0049, AV CR S1075351, S1075102, DESIGNER
```

mixpro get mixture projection (prediction on conditioned marginal pdf)

```
LHS  = mixpro(Mix, pchns, cchns, psi, str)
LHS  = mixpro(Mix, pchns, cchns)      no rawdata suplied
LHS  = mixpro(Mix, pchns)             cchns = []
LHS  = mixpro(Mix)                    pchns = modelled channels
LHS  = pMix : mixture prediction or LHS = [Eths, coves, dfcs]
       with mixture prediction sum dfcs * N(Eths,coves)
Mix   : mixture or p-mixture, any form
pchns : modelled channels
cchns : channels in condition
psi0  : values of channels in condition
Design  : P. Nedoma
Updated : August 2003
Project : ProDaCTools
```

---

mixscan scan mixture for 2 dim. clusters

```
mixscan(Mix,chns)
mixscan(Mix)  all predicted channels
Design : P. Nedoma
Updated: July 2002
Project: ProDaCTools
```

---

mixsimul mixture simulation

```
Sim = mixsimul(Sim)         - recursive
Sim = mixsimul(Sim, ndat)   - batch
Sim :  mixture simulator
ndat:  size of data sample
Design : P. Nedoma
Updated: August 2003
Project: GA CR 102/03/0049, AV CR S1075351, S1075102, DESIGNER
```

---

mixstats compute estimation statistics for a fixed mixture

```
Mix = mixstats(Mix, ndat)     batch
Mix = mixstats(Mix)           recursive
Design : P. Nedoma
Updated: March 2003
Project: ProDaCTools remake
```

---

mixstrid estimate mixture structure

```
[Mix, Mix0, flag] = mixstrid(Mix, Mix0, belief, nruns)
Mix   : estimated mixture
Mix0  : initial mixture
flag  : 0 if not changed, otherwise 1
belief: cell vector of believes for individual channels
nruns : number of runs for factor structure estimation
Design : P. Nedoma
Updated: April 2002
Project: ProDaCTools
```

---

mixupdt one step of mixture upda

```
 Mix = mixupdt(Mix0, flag)
 Mix = mixupdt(Mix0)
Mix     : estimated mixture
Mix0    : initial mixture
flag    : 1 only trial step
          0 states physical update
weight  : mixture weight (used in mixinit)
Design  : P. Nedoma
Updated : March 2003
Project : ProDaCTools remake
```

---

pro2pre convert predictor component to prediction o

```
 [Com1, scale] = pro2pre(Com)
Com1   : prediction component (114)
scale  : component scale
Com    : predictor component(112)
Note: Com.datavect contains prediction data
Design : P. Nedoma
Updated: February 2005
Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
         MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
```

---

profix get mixture projection from projector

```
OUTPUTS = profix(pMix)
OUTPUTS = profix(pMix, psi, str, noflt)
OUTPUTS
pMix1                 or [pMix1, scales]  or
[Eths, coves, dfcs]   or [Eths, coves, dfcs, scales]
pMix  : mixture predictor
psi   : a datavector
str   : description of psi
OUTPUTS
pMix1 : mixture prediction
Eths  : cell vector or vector of means of the mixture component
coves : cell vector or vector of noise covariances of the mixture component
dfcs  : transformed mixture dfcs respecting conditioning
scales: approximate data dependent component weights
Design : P. Nedoma
Updated: February 2005
Project: GACR 102/03/0049, AVCR S1075351, 1ET100750401, Edukalibre
         MINERVA 110330-CP-1-2003-1-ES-MINERVA-M
```

---

prt debugging prints

```
use : prt(mes , X)
      prt(mes)
mes : message
X   : cell vector, structure matrix, LS normal mixture, normal factor
Design : P. Nedoma
Updated: August 2001
Project: ProDaCTools
```

---

relep [se, yp] = relep(Mix, ndat, use-wgs)

```
se:     relative prediction error
yp:     prediction trajectory
Mix:    estimated mixture  or mixture predictor
ndat:   portion of the data sample to be processed
use-wgs: use additional components weights
Design: P. Nedoma
Project: GA CR 102/03/0049, AV CR S1075351, S1075102, DESIGNER
```

---

scalepsi scale psi, return scaling of predicted channels in "pchns" order

```
[psi0, M, S] = scalepsi(psi0, pre, pchns, cchns)
psi0 : not scaled values on zero-delayed output channels
pre  : overall scaling
Design : P. Nedoma
Updated: February 2002
Project: ProDaCTools
```

---

simbldl build simulation states

```
Sim = simbldl(Mix)
Sim : mixture simulator
Mix : any mixture
```

---

simopt Simulation options

```
Design : P. Nedoma
Updated: April 2004
Project: GA CR 102/03/0049, AV CR S1075351, S1075102, DESIGNER
```

---

statgrid generate coordinates of a stataic mixture

```
[x,y,z] = statgrid(Eths, coves, alphas, n, r)
[x,y,z] = statgrid(pMix, n, r)
x,y,z  : coordinates
Eths   : vector or cell-vector of means
coves  : vector or cell vector of noise variances
alphas : normalized degrees of freedom of components
n      : grid density - number of points (or densities)
r      : grid range (or ranges)
pMix   : mixture projection
Design : P. Nedoma
Updated: June 2001
Project: ProDaCTools
```

---

statplot plot centers and 2-sigma borders of components of a mixture

```
statplot(Mix, arg)
Mix : mixture type 114
arg : character string for plot (e.g.'r--');
      1  - standard
      2  - standard, instead of 'o' in center, component No is displayed
      missing: standard, before plot: clf; hold off
plot ends by hold on
Design : P. Nedoma
Updated: September 2001
Project: ProDaCTools
```

---

statsim build static mixture simulator and data sample, 2 channels

```
Sim = statsim(ndat, ncom, cove)
ndat:  size of data sample (located at global DATA)
ncom:  number of components located on unit circle
cove:  LD decomposition of the components common covariance (default eye(2))
Sim :  the mixture simulator
%
convert between relative and absolute structure form
str1 = str2str(comstr, str)
str1   : structure, type oposite to str
comstr : component structure
str    : absolute or relative structure
Design : P. Nedoma
Updated: August 2003
Project: GA CR 102/03/0049, AV CR S1075351, S1075102, DESIGNER
```