Additions to Mixtools and Designer

Contents

1 Prior knowledge in ARX models

The prior knowledge is a useful tool in demanding learning tasks, e.g. in the task of ARX model structure estimation. As a rule, data available are poorly informative in this case. Underlying philosophy and processing algorithms are described in [1, 2, 3, 4, 5, 6, 7].

1.1 Prior knowledge coding

The prior knowledge is coded in the form of a *cell list* that means as a cell vector of pairs of cells. The first one of each pair specifies the type of the prior knowledge, the second one (often again cell vector) contains numerical characteristics.

Individual prior knowledge types considered are exposed. The first two types refers to all outputs. The function `prior` does the prior knowledge processing - see later on.

Data sample

can contain historical data not fully consistent with the current data sample, e.g. not sufficiently excited or recorded in another working point. The data collected on simulated system has this character, too.

The data are processed with a forgetting rate or with a grid of forgetting rates. If the forgetting rates are not specified, the `prior` function adds a default grid of rates (from "defaults.m"). For the data sample recorded in the matrix `data`, the respective coding is:

```
{'data' {data frgs} }  or
     {'data' data}
```

Data envelope

are two data matrices representing data ranges *datalow* and *datahigh*. The coding is:

```
{'fdata' {datalow datahigh}}
```

The prior knowledge pieces that follow are responses observed on an *output channel* caused by a special signal applied to an *input channel*. All frequencies specified are in Hertz, the phases are in degrees. With the exception of system static gain, it is necessary to specify also the *sampling time* (in seconds) in any position in the prior knowledge list:

```
{'stime' sampling_time }
```

The identifiers used are:

`ychn`	output channel
`uchn`	input channel
`st`	sampling time

The relevant prior knowledge pieces are discussed.

System static gain
is change of an output in steady state due to unit change of an input. It is specified in range of values:

```
{'gain' [uchn gainlow gainhigh] }
```

Frequency response
means amplitude and phase of output when a sinusoidal signal of a frequency and amplitude one is applied to an input. The amplitude is expected in a range of values (`alow, ahigh`). The coding is:

```
{'ampl' [uchn frequency alow ahigh phase]}
```

The phase can be specified as a vector of values. If it is empty, the `prior` function adds a default grid of phases.

Cut–off frequency
is the frequency of the signal applied to an input that is not reflected by the output:

```
{'cut'  [uchn frequency]}
```

The frequencies allow multiple specification. If not specified, the function `prior` adds a default grid of values (multiples of the frequency specified).

Dominant time constants
are implemented by modelling lower and upper envelope of the impulse response generated by the first or second order model with time constants equal to the specified bounds $(tclow, tchigh)$ on the time constant. It is specified in range of values:

```
{'tc' [uchn tclow tchigh]}
```

The prior knowledge list can contain specification related to several output channels. In the case, the sub-lists must be separated by output specification that is valid till a new specification:

```
{'ychn' ychn }
```

1.2 Conversion of the prior knowledge into fictitious factors

Generally, it is strongly recommended to scale data sample. At the case, the prior knowledge must be scaled, too. The scaling of data supplies scale of individual channels. It is used in scaling of prior knowledge pieces:

```
pre  = preproc( {'scale' [ ]} );          % scale data
pri  = scalepri( pri, pre);               % scale prior knowledge
```

The prior knowledge is converted into fictitious factor(s) by:

```
Facs = prior(Facs0, pri)
```

The arguments are:

`Facs`	fictitious factor or array of factors for individual outputs
`Facs0`	initial ARX factor or array of factors for individual outputs
`pri`	prior knowledge list

The specification of grids of values is added if not specified.

1.2.1 Example of prior knowledge coding and processing in MISO case

In single-output model we work with initial ARX factor `Fac0`. The result is an ARX fictitious factor or a cell vector of fictitious factors.

The prior knowledge consist of gain and data sample in the matrix `data`. The prior information is coded and converted into fictitious factors.

```
uchn = 2;                                % input channel
ga1.51 [0.99 1.01];                      % range for gain
pri  = {'gain' [uchn gain], 'data' data }; % prior knowledge list
Facs = prior(Fac0, pri)                  % conversion prior to fictitious factor
Facs =
    [1x1 struct]    {1x4 cell}
```

The first cell of the result is fictitious factor with gain information. The second one contains four factors processed with different forgetting rates internally included.

The factors can be ordered into one cell vector. Then, the dfm of each factors is displayed (only as an example):

```
Facs = cellcat(Facs)                     % convert to simple cell
Facs =
  Columns 1 through 4
    [1x1 struct]    [1x1 struct]    [1x1 struct]    [1x1 struct]
  Column 5
    [1x1 struct]
dfms = getflds(Facs, 'dfm')              % get dfm of factors (an example)
dfms =
    1.6189   75.7711   86.5991   91.0913   90.6703
```

1.2.2 Example of prior knowledge coding and processing in MIMO case

In the MIMO case, the result is cell vector of results for individual outputs. Each of the partial results is again a cell vector of results generated by individual pieces of the prior knowledge. An example follows.

We have two initial factors `Fac01` and `Fac03` for output channels 1 and 3. The inputs are channels 2 and 4. Prior knowledge are system static gains and a data sample from simulation model. The prior knowledge is coded as:

```
pri ={'ychn' 1 'gain' [2 0.99 1.01]...  % gain output 1, input 2
      'ychn' 3 'gain' [4 2.13 2.95]...  % gain output 3, input 4
               'data' data  };          % data sample
```

The processing is done:

```
Facs0 = {Fac01 Fac03};                   % initial factors
Facs  = prior(Facs0, pri)                % coded fictitious factors
Facs =
    {1x2 cell}    {1x2 cell}
```

The result is cell vector of results for individual outputs. The coded results for the output 1 is accessed as:

```
Facs1 = Facs{1}
Facs1 =
    {1x1 cell}    {1x4 cell}
```

Now, the prior knowledge on gain occupies the 1st cell. The prior data are processed with different forgetting rates (added by the `prior` function) and are placed in the 2nd cell of the result.

The function `cellcat` converts the complex cell vector into the single one:

```
Facs1 = cellcat(Facs1)                    % concatenate cell vectors
Facs1 =
  Columns 1 through 4
    [1x1 struct]    [1x1 struct]    [1x1 struct]    [1x1 struct]
  Column 5
    [1x1 struct]
```

1.2.3 Merging of fictitious factors with data sample

The results of prior data processing must be weighted and merged with data sample. The merger is used in structure estimation and as the alternative model for stabilized forgetting. The merging is done by:

```
[Fac, vll, Fac0] = primerge(FacD, Fac0, Facs)
```

The arguments are:

Fac	merged posterior factor with prior knowledge
vll	v-log-likelihood
Fac0	merge prior factors
FacD	estimated factor with very flat initial factor
Fac0	initial fictitious factor
Facs	fictitious factor or cell vector of the factors reflection prior knowledge

1.3 Model structure estimation with prior knowledge

The function `pristr` is designed for estimation of structure of factors. It searches for the factor structure that has the highest posterior probability in a space of competitive factor structures.

The user specifies the space of competitors in the form of the richest (maximum possible) factor structure.

The function `pristr` is called as:

```
[Facs, vlls] = pristr(Facs0, pri, beliefs, nbest, nrep)
```

The function arguments are:

`Facs` is resulting estimated factor or cell vector of factors in MIMO case

`vlls` information about v-log-likelihood (see example in the next subsection):

vlls1	v-log-likelihood, nested prior
vlls2	v-log-likelihood with prior
vlls3	"best" structures in cell array
vlls4	structures in a block

`Facs0` is the richest initial factor or cell vector of initial factors

`pri` is the prior knowledge list

`beliefs` specifies user's *belief* in the maximum richest structure. The belief is a vector of the same length as the richest factor structure. Its elements specify that the corresponding items (the pairs of channel and delay) of the richest structure are in the estimated model structure:

1	surely present	2	probably present
3	probably not present	4	surely not present

In the MIMO case, the `beliefs` is cell vector of user belief for individual outputs. The empty belief can be specified.

`nbest` specifies the number of "best" structures encountered during repetitive estimations.

`nrep` specifies number of repetitive searches with random starts. Two runs are done automatically - start from empty and full regression vector. If results of the runs differ, a warning is displayed - the meaning is to discover local maxima in the space of competitive structures.

1.3.1 Example of structure estimation, SISO case

Example of structure estimation follows. Dynamic SISO model is considered. Data are simulated for insight into processing. The dynamic model is:

```
ychn = 1;                                  % output channel
uchn = 2;                                  % input channel
str  = [ychn ychn uchn uchn; 1 2 0 1];     % factor structure
Eth  = [1.81 -0.8187 0.00468 0.00438];     % regression coefficients
Fac  = facarxls(ychn, str);                % ARX LS factor
Fac.Eth  = Eth;
Fac.cove = 0.0001;                         % noise covariance
```

The data sample is generated:

```
ndat = 300;                                % sample size
randn('seed', 7);                          % fix seed
DATA = [zeros(1, ndat)                      % pre-allocated data sample
        0.4*randn(1, ndat)];                % output std = 0.4
Sim  = mixconst(Fac, 1, 1);                % build simulator
mixsimul(Sim, ndat);                       % get data sample
```

The task of structure estimation follows. The richest structure is expressed as the model of 6th order and the initial factor is build:

```
maxstr = [ones(1,6), 1+ones(1,7), 0        % richest structure
            1:6,       0:6,       1];
Fac0   = facarx(ychn, maxstr);             % richest initial factor
```

The prior knowledge items considered are:

```
gain = [uchn 0.99 1.01];                   % static gain
tc   = [uchn 0.82 0.84 ];                  % dominant time constant
load ... dlow dhigh                        % load data envelope
load ... data                              % load data sample
```

The data sample of the length 100 is generated by the model above with the regression coefficient $b_0 = 0$. The data envelope is average of 20 realizations of step response specified in two matrices `dlow, dhigh`. The length considered was 150.
The basic prior knowledge items are:

```
pri  = {'gain' gain};
pri  = {'data' data};
pri  = {'fdata' {dlow dhigh} };
pri  = {'st' 0.1 'tcons' tc};
```

Data to be processed by the structure estimation algorithm should be scaled. Accordingly, the prior knowledge must be scaled. This is done:

```
pre  = preproc( {'scale' []} );            % scale data
pri  = scalepri( pri, pre, ychn);          % scale prior knowledge
```

Now, the number of "best" regressors considered and number of repetitive random starts is to be specified:

```
nbest = 10;                                % number of "best" regressors
nruns = 50;                                % number of estimation runs
```

Note: the values of 30 for nbest and 50 for nruns are current default values.
The structure estimation is done (the belief is omitted):

```
[Fac, vll] = pristr(Fac0, pri, [ ], nbest, nruns);
```

The optional second output argument is designed for detailed analysis. The possible processing is as follows:

```
wgs  = vll{1};                             % v-log-likelihood with nested prior
wgs1 = wgs/sum(wgs);                       % normalization
```

```
wgs  = vll{2};                          % v-log-likelihood with prior
wgs2 = wgs/sum(wgs);

plot(wgs1,'o'); hold on;                % plot of v-log-likelihood
plot(wgs2,'*');

ilh = vll{3};                           % simplified plot of best structures
for i=1:nbest
    fprintf('%2i',ilh(i,:)); disp(' ');
end
```

The display of "best" regressors shows that the "true" regressor is the 4th one:

```
 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  0  0  0  0  0  0  0  0  0  0  0  0   0.82800
 1  1  0  0  0  0  0  1  0  0  0  0  0  0   0.15400
 1  1  0  0  0  0  1  0  0  0  0  0  0  0   0.00685
 1  1  0  0  0  0  1  1  0  0  0  0  0  0   0.00656
 1  1  1  0  0  0  0  0  0  0  0  0  0  0   0.00184
 1  1  0  0  0  0  0  0  0  0  0  0  1  0   0.00171
 1  1  0  1  0  0  0  0  0  0  0  0  0  0   0.00042
 1  1  0  0  1  0  0  0  0  0  0  0  0  0   0.00029
 1  1  0  0  0  1  0  0  0  0  0  0  0  0   0.00022
 1  1  1  0  0  0  0  1  0  0  0  0  0  0   0.00020
```

The plots of v-log-likelihood is in Fig. 1 and 2. The numbers on x-axis coincide with rows of the table above. The normalized v-log-likelihood is displayed on y-axis marked by character 'o' for nested prior knowledge. The probabilities marked by '*' resulted from the use of prior knowledge described in heading.

The simple conclusions can be drawn from the plots:

- static gain, data envelope and data increase probability of the true structure significantly
- time constant brings nothing in this case
- combination of knowledge items does not guarantee significant improvement
- combination of bad and good knowledge do not spoil the result.

1.3.2 Example of structure estimation, MIMO case

The example is extension of the example of previous subsection - two independent systems are build on four channels. For simplicity, the data are: DATA = [DATA; DATA];

```
DATA = [DATA; DATA];                    % data sample, 4 dimensions
```

The richest initial factors are build:

```
ychn1 = 1;                              % output channel
uchn1 = 2;                              % input channel
maxstr1 = [ones(1,6), 1+ones(1,7), 0    % richest structure, channel1
           1:6,       0:6,         1];
Fac01    = facarx(ychn, maxstr1);       % richest initial factor channel1
```

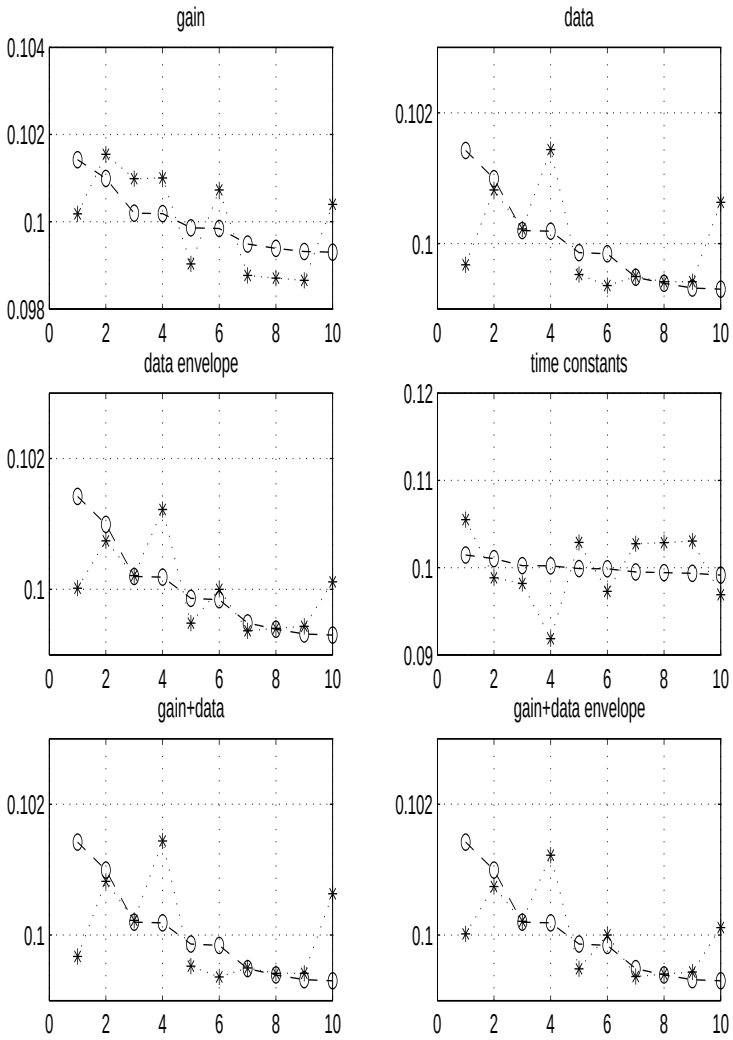

Figure 1: Stucture estimation with prior knowledge

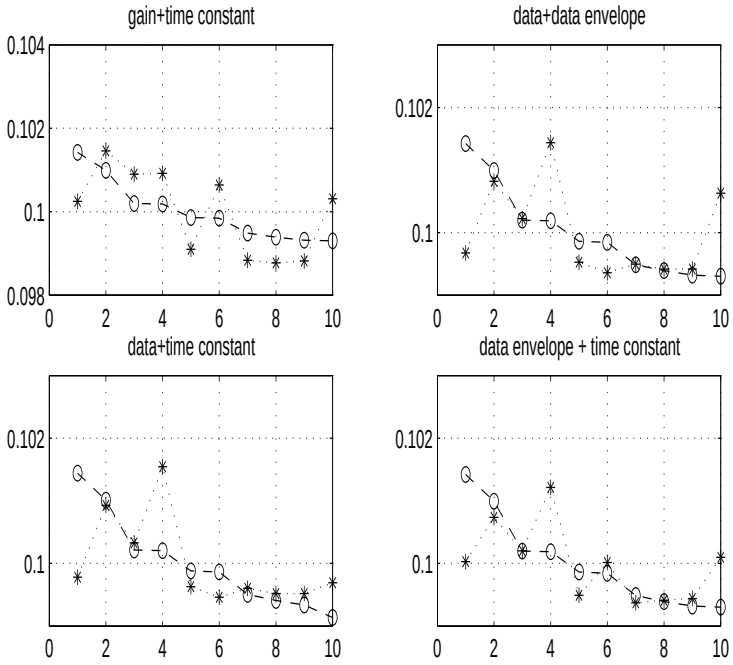

Figure 2: Stucture estimation with prior knowledge

```
ychn2   = 3;
uchn2   = 4;
maxstr2 = [2+ones(1,6), 3+ones(1,7), 0  % richest structure, channel 3
           1:6,       0:6,         1];
Fac02    = facarx(ychn2, maxstr2);      % richest initial factor, channel 3
Facs0 = {Fac01 Fac02};                  % initial factors
```

The data sample is pre-processed:

```
pre  = preproc( {'scale' []} );         % data are preprocessed
```

The prior knowledge used is:

```
pri  = {'ychn' ychn1 'gain' [uchn1 gain] ...
        'ychn' ychn2 'gain' [uchn2 gain] ...
        'fdat' {dlow dhigh} };
```

The prior knowledge must be scaled. It is done simply by:

```
pri  = scalepri(pri, pre);              % preprocess prior knowledge
```

The structure is estimated and results displayed:

```
nbest = 10;                             % number of "best" regressors, default is 30
nruns = 50;                             % number of estimation runs
[Facs, vlls] = pristr(Facs0, pri, [], nbest, nruns);
```

```
Facs = arx2arx(Facs);
Fac1 = Facs{1};
Fac1.str, Fac1.Eth
ans =
     1     1     2     2
     1     2     0     1
ans =
    1.7624   -0.7731    0.0119    0.0146
Fac2 = Facs{2};
Fac2.str, Fac2.Eth
ans =
     3     3     4     4
     1     2     0     1
ans =
    1.7624   -0.7731    0.0119    0.0146
```

Note: in simple cases, the structure estimation can be done channel-wise. The advantage of processing outlined is that the data increment of initial factors is computed only ones.

1.4 Prior knowledge and channel description

A brief excursion into the channel description approach. The example of the previous section is continued.

Before the structure estimation starts, the channel description is build

```
Chns = chnconst(1:4);                    % channel description
```

The scaling of individual channels is done and the scaling of individual channels is recorded:

```
pre  = preproc( {'scale' []} );          % data are preprocessed
Chns = chnset(Chns, 'scale', pre);       % set channel scaling
```

Prior knowledge is scaled and recorded in the channels structure:

```
pri  = scalepri( pri, pre);              % scale prior knowledge
Chns = chnset(Chns, 'prior', pri);
Chns{1}.prior
ans =
    'ychn'    [1]    'gain'    [1x3 double]
Chns{3}.prior
ans =
    'ychn'    [3]    'gain'    [1x3 double]    'fdat'    {1x2 cell}
```

The structure estimation is done. The `pri` argument can be replaced by the channel description.

```
nbest = 10;                              % number of "best" regressors
nruns = 15;                              % number of estimation runs
[Facs, vlls] = pristr(Facs0, Chns, [], nbest, nruns);
Facs = arx2arx(Facs);
Facs{1}.str
ans =
     1     1     2     2
     1     2     0     1
Facs{1}.Eth
ans =
    1.7624   -0.7731    0.0119    0.0146
```

Note: in future design, the scaling of prior knowledge can be done automatically inside the function `chnset`*. It the* `pri` *argument is empty, it will be substituted by a global channel description.*

References

[1] P. Nedoma, M. Kárný, and J. Böhm, "Software tools for use of prior knowledge in design of LQG adaptive controllers", in *The preprints of IFAC workshop ACASP'98*, pp. 425–429. Glasgow, 1998.

[2] Kárný M. and Nedoma P., "Automatic processing of prior information with application to identification of regression model", *Kybernetika*, 1999, submitted.

[3] Kárný M., Khailova N., Nedoma P., and Bohm J., "Quantificaton of prior information revised", *Adaptive Control and Signal Processing*, vol. 15, no. 1, pp. 67–84, 1999.

[4] M. Kárný, N. Khailova, P. Nedoma, and J. Böhm, "Quantification of prior information revised", *International Journal of Adaptive Control and Signal Processing*, vol. 15, no. 1, pp. 65–84, 2001.

[5] Quinn A., Ettler P., Jirsa L., Nagy I., and Nedoma P., "Use of prior informartion in structure estimation", *International Journal of Adaptive Control and Signal Processing, special issue*, vol. 17, no. 3, 2003, accepted.

[6] N. Khailova, M. Kárný, P. Nedoma, and J. Bůcha, "Apriorní znalost pro počítačový návrh adaptivního řízení", *Automa*, vol. 8, no. 10, pp. 45–49, 2002.

[7] N. Khailova, *Exploitation of Prior Knowledge in Adaptive Control Design. Ph.D. Thesis*, PhD thesis, 2002.