ABET98

Adaptive Bayesian Estimation Toolbox

User's Guide

Contents

1 Introduction

The ABET toolbox for MATLAB contains a selection of techniques primarily related to off-line design of adaptive controllers but, its functions are not limited to the task - at the same time, the toolbox is an experimental environment oriented to Bayesian estimation and decision making.

The toolbox is also a design base for project "Designer 2000" [1] that aims at interactive design of adaptive controllers.

The design of adaptive controllers under ABET is based on real dynamic system measurements and prior knowledge about the system. The design result is an C-encoded program module that is employed in the real time control computer and controls the system.

The design of adaptive controllers covers the design steps listed below together with the ABET function that solves the step.

Data preprocessing	`datahand`
Control period estimation	`peride`
Model structure estimation	`eststr`
Prior knowledge processing	`prior`
Forgetting estimation	`estfrg`
Parameter estimation	`estpar`
Control design	`cdesign`
Control loop performance estimation	`requsat`
Controller code generation	`encreg`
Control verification	`versim`

1.1 Abbreviations

pdf.	probability density function
std.	standard deviation
SI	single input model (system)
MI	multiple input model
SO	single output model
MO	multiple output model
SISO	SI, SO model
MISO	MI, SO model
MIMO	MI, MO model
ARX	autoregression with exogenous variables

1.2 Controlled system

Common terms used in technical practice will be used without detailed description.

The system to be controlled is measured at several places - channels. The measured signals are controlled outputs, inputs and auxiliary outputs.

The signals are recorded and enter ABET processing as a MATLAB matrix referred to as <u>signals</u>. The signal recorded on a channel is stored as the i-th row of the matrix; the number i is referred to as <u>channel</u>.

By <u>signal description</u> we refer to a character string expressing that corresponding signal is:

'o'	controlled output
'i'	input
'a'	auxiliary output
'-'	(temporary) not used in processing

Example:

```
des = 'oi';    % SISO
des = 'oiaa';  % MISO
des = 'oioi';  % MIMO
```

1.3 Computational means

ABET97 uses two tools introduced by MATLAB version 5 :
- cell arrays;
- structures.

Both tools offer the possibility that several matrices are handled as one matrix.

ABET uses for the same purpose similar approach supported by a set of functions. It makes it possible to work with the tools under any MATLAB version.

1.3.1 Cell vectors

A cell vector contains matrices as elements. The vector elements are referred to as cells. The function `cell4` builds the cell vector. The functions `getcell` and `setcell` get and set individual cells respectively.

Warning: the cell vector is not MATLAB 5 cell array but, it can be easily converted into it (a conversion function is available). The `cell4` *function differs from the MATLAB 5 function* `cell`*; the suffix character '4' is used to distinguish them.*

The following MATLAB session demonstrates how to build a cell vector and access its cells. The arguments of the `cell4` function are the matrices that form the cell vector. The function `disp4` displays the cell vector contents. Example:

```
A = cell4([1 2 3], 'abcd');
getcell(A,1)
ans =
      1     2     3
B = setcell(A, 2, [3 4]);
disp4(A);
ABET cell vector with 2 cells
1) m1 :double  (3) = 1  2  3
2) m2 :char    (4) = abcd

disp4(B);
ABET cell vector with 2 cells
1) m1 :double  (3) = 1  2  3
2) m2 :double  (2) = 3  4
```

Note: the cell vector is displayed with fictive names `m*` that label the individual cells.

1.3.2 Cell list

In ABET, variable list of requirements for a processing is allways expressed in the form of a cell vector. For example, the following cell vector defines a requirement for signal preprocessing:

```
pre = cell4('channel', 2, 'scale', [ 1 5 ]);
```

This is encoded requirement that specifies that each channel 2 signal value will be preprocessed before used - 1 will be added and the result multiplied by 5.

We shall refer to this type of the cell vector as cell list. It consist of pair of arguments - processing options - the first one is a qualitative description (character string specifying which option if referred to) and the second one carries a quantitative information about the option. For the option string, the first character is decisive, as a rule.

The function `getlist` is designed to extract the information encoded in the list, e.g.:

```
getlist(pre, 2, 's')  % gets scale information for 2nd channel
ans =
     1     5
getlist(pre, 's')       % gets first cell item coded as 's...'
ans =
     1     5
```

Instead of pair of arguments, the list can contain one cell list, e.g.

```
pre = cell4(pre, cell4('c', 3, 's', [ 10  50 ]));
getlist(pre, 3, 's')
ans =
    10    50
```

1.3.3 Structures

A structure is a vector containing named matrices - fields - as elements. It is created by function `struct4` and the fields are accessed by `getfld` and `setfld`.

Warning: the ABET structure is not a MATLAB 5 structure. A conversion to the MATLAB 5 structure is straight–forward (by a conversion function).

Example:

```
A = struct4('a', [1 2 3],'b', 'abcd');
getfld(A,'a')
ans =
      1     2     3
B = setfld(A, 'b', [3 4]);
disp4(A);
ABET structure with 2 fields
1) a :double  (3) = 1  2  3
2) b :char    (4) = abcd
disp4(B);
ABET structure with 2 fields
1) a :double  (3) = 1  2  3
2) b :double  (2) = 3  4
```

1.4 Models

1.4.1 General considerations

A model is description of relationships between inputs and outputs. ABET works with autoregressive models with exogenous variables (ARX), Markov models and multiple ARX models. They are shortly described in the following subsections. The general features are discussed here. They refer to SO models - the case of MO models is handled as a set of models for individual outputs.

The description that model provides is a pdf of an output at the time t conditioned by delayed (older) inputs and outputs and the current inputs at time t. The pdf is parameterised by unknown model parameters.

Using signals and prior knowledge available, the pdf is calculated of the parameters conditioned by the signals. By likelihood (a number) at a time t we mean the value of the predictive pdf with the inputs and outputs up to t substituted into it.

The statistics describing this pdf contain several model attributes represented as matrices. They are updated by signals during recursive parameter estimation.

The model attributes grow with number of data items processed. During recursive parameter estimation reach soon values that cannot be influenced by new signals. The forgetting allows to forget obsolete information and build the statistics from newly entered data. The rate of forgetting is controlled by forgetting factor (positive scalar < 1).

In ABET we mean by model a structure with the specific fields (diferrent for different model types). A model can be used for different purposes (explained later) , e.g.

initial model	model before parameter estimation is done;
prior model	model that comprises processed prior knowledge;
identified model	model obtained by parameter estimation
default model	default initial model used by functions

A model is usually created as output from a ABET function. The initial and alternative models appear among input arguments of the functions; as a general recommendation for ABET processing is to substitute a prior model for both of them.

The model fields are:

output channel `ychn`	says which output is described by the model;
model structure `str`	specifies which delayed outputs and inputs enters in the model;
model attributes	model type dependent matrices.

In MO cases, the model is a cell vector containing models for individual outputs.

The model structure is encoded as a matrix with two rows. The first row contains a list of channels, the second one a list of corresponding delays. A model offset (an additive constant in model) may extend the model structure by inserting 0 in the first row and the offset value in the second one. In the case of MO models, the model structure is a cell vector of model structures of individual outputs.

Example: the model structures used later in examples are:

```
str = [1 1  2 2; 1 2  0 1];                 % SO case
str = [1 1  2 2  0; 1 2  0 1  1];           % SO with offset
str = cell4(str, [3 3  4 4; 1 2  0 1]);     % MO case
str = rgrorder('oi', 3)                     % model of the order 3
ans =
     1     1     1     2     2     2     2
     1     2     3     0     1     2     3
```

The function `rgrorder` generates the model structure of an order, this is common term used in the field.

The ABET processing is generally recursive - the model attributes are updated with each processed data item. All recursive model attributes can be recorded in a matrix. We refer to the matrix trajectory of the model attribute.

1.4.2 ARX model

The most elaborated model used for identification and control is an autoregression with exogenous (external) variables (ARX). The model definition can be found in standard textbooks. We discuss here an example only.

The ARX model with output y and input u can be e.g. (the time and dimension indexes are distinguished):

$$y_t = a(1) * y_{t-1} + a(2) * y_{t-2} + b(0) * u_{t-td} + b(1) * u_{t-td-1} + o + e_t$$

where

a, b	are model coefficients
$y_{t-1}, y_{t-2}, u_t, u_{t-1}$	are inputs and outputs delayed with respect to the time t
o	is model offset
td	is transport delay
e_t	is the output noise with normal pdf, 0 mean, variace σ^2 ($\mathcal{N}(0, \sigma^2)$)

The model paramaters are the model coefficients and σ.

The function `arxmodel` genetes the default model:

```
des   = 'oi';                    % signal description
str   = [1 1  2 2; 1 2 0 1];     % model structure
Model = arxmodel(des,str);       % default model
disp4(Model);
ABET structure with 6 fields
 1) ychn :double   = 1
 2) str :double  (2,4) = 1  1  1  2  2  ...
 3) covy :double   = 0.0001
 4) Eth :double  (4) = 0  0  0  0
 5) Cth :double  (4,4) = 10000  0  0  0  0  ...
 6) dfm :double   = 1
```

The meaning of indidual model fields are:

ychn	output channel
str	model structure
	model attributes
`covy`	output noise standard deviation (σ)
`Eth`	mean of coefficients distribution
`Cth`	encoded covariance of the coefficients
`dfm`	the number of degrees of freedom

2 SISO control design

The control design is a complex task. The design steps are discussed in the next section. The ABET users, that do not want to enter processing details, have a possibility to design adaptive controller by the function `acdesign` that covers all design steps. The function is called as:

```
cresult = acdesign(data, des, maxstr, cons, pri, maxper, cmodule)
```

There are two results of the design:

1. the structure `cresult` that carries all information about the control design. The results can be displayed by the function `dispreg`;
2. C-encoded controller generated in the current working directory (name is specified by `cmodule`) that can be used in any C-encoded control program.

The input arguments specify:

`data`	signals
`des`	signal description
`maxstr`	maximum prior model structure
`cons`	control design constraints
`pri`	prior knowledge
`maxper`	maximum prior control period
`cmodule`	name of C-encoded module

The last three arguments are optional (if not used, no prior model is considered, the control period is one and no code is generated).

The meaning of the arguments is:

`data` specifies a name of a global matrix with signals.

`des` is signal description, e.g. `des = 'oi';`

`maxstr` specifies a maximum possible prior model structure, where a MAP structure estimates is searched for, e.g.:

```
maxstr = rgrodr(des,3)
```

specifies that the structure of the system to be controlled is maximum of the third order;

`cons` are constraints for controller design. The constrains are input ranges, input increment ranges and overshoot ranges. The constrains are encoded in an cell list, e.g.

```
cons = cell4('u', [- 10 10], 'du', [-5 5], 'ovr', 0.2 );
```

specify that the controller is to be designed in such a way that the input range and input difference range and overshoot fulfil the constraints specified. The overshoot can be specified as a scalar (in absolute signal values).

`pri` contains prior knowledge, e.g. the information about static gain between $< 0.9, 1.1 >$ of the output channel 1 due to unit step of input channel 2 is encoded as:

```
pri = cell4('gain',[ 1 2 0.99 1.1])
```

Refer to the function `prior` for more information;

`maxper` specifies maximum period (in multiples of sampling period) considered internally for selection of optimum control period, e.g. `maxper = 3;`.

`cmodule` is name (character string) of C-encoded controller located in the current working directory that contains C-encoded controller. It can be used in any C-encoded control program or as a MATLAB MEX-file.

2.1 Test ARX system

In tutorial examples of this guide, SISO simulated signals are used for an insight into processing. The first processing channel is output, the second one input. The ARX model used for processing has the following attributes:

```
str   = [1 1  2 2; 1 2  0 1 ];              % test model structure
Eth   = [1.81 -0.8187 0.00468 0.00438];     % test model parameters
covy  = 0.01;                  % test model output standard deviation
```

The open loop signals (input is standard white noise) are generated by the function `testdata`:

```
data = testdata(len, covy, seed, offset, delay)
```

The meaning of the arguments is:

		defaults
`len`	number of signal items	—
`sigma`	standard deviation (σ, `covy`) of output noise	0.01
`seed`	seed of random numbers generator	current seed
`offset`	model offset	0
`delay`	transport delay	0

The function `testsys` is designed for close loop generation of the test signals by the same model as `testdata`. In the example that follow, the test system unit step response is analysed. The trajectory of output (`Y`) is computed, averaged over 100 simulation runs and displayed together with standard deviation of the trajectories (we shall refer to this form as trajectory range). This is methodology of carrying out the simulation studies with ABET - it is preferable than show only one trajectory.

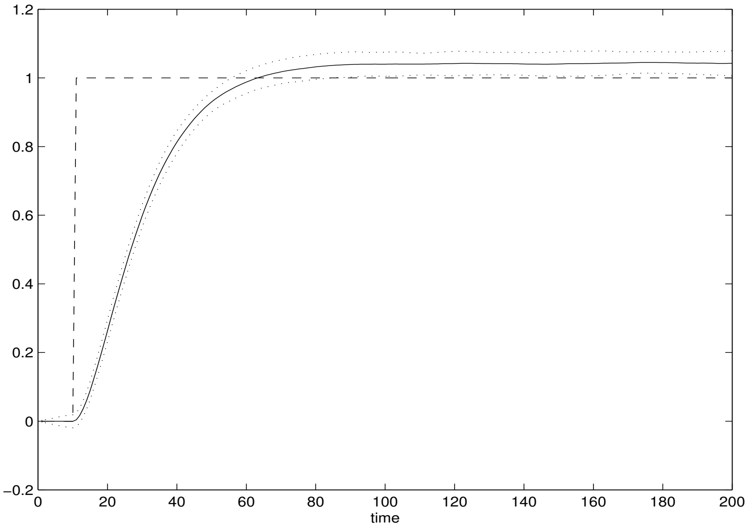
Session diary follows:

```
randn('seed', 123);      % fixes random trajectory
sigma = 0.001;           % standard deviation of output noise
u     = [zeros(1,10), ones(1,190)]; % input unit step
nrun  = 100;             % number of simulation runs
Y     = zeros(1,200);    % used for results of one run
S     = Y;               % used for sum of individual runs
SS    = Y;               % used for sum of squares "

for run = 1: nrun
    sys = testsys(sigma);  % initialise test system
    for time = 1: 200      % one trajectory
        [y, sys] = testsys(u(time), sys); % generate output
        Y(time)=y;         % store output
    end
    S = S + Y; SS = SS + Y.*Y;  % store results
end

meanY = S/nrun;  % mean trajectory of output
% standard deviation of trajectory
stdY = sqrt( (SS - 2*S.*meanY + nrun*meanY.*meanY)/(nrun-1) );
clf; hold off; plot(meanY); hold on; plot(u,'--');
plot(meanY + 2 * stdY,':');  plot(meanY - 2 * stdY,':');
xlabel('time');
```

Trajectory range and input are:



mean unit step response of the test system

The averaged trajectory of output can be used as a prior knowledge.

Another prior knowledge is gain of the system that can be computed e.g. as

```
gain_value = meanY(200);
gain_std   = stdY(200);
gain_range = [gain_value - 3 * gain_std, gain_value + 3*gain_std]
gain_range =
    0.9877    1.0977
```

In next sections, we shall use for value range the interval $< 0.99, 1.1 >$.

2.2 Design example

This section contains a tutorial example of adaptive controller design. The design diary follows.

Signals are generated by the test system:

```
global signals
sigma   = 0.01;
signals = testdata(600, sigma);  % signals
```

The design inputs are:

```
data   = 'signals';                 % signals
des    = 'oi';                      % signal description
maxstr = rgrord(des , 3)            % maximum prior model structure
maxstr =
     1     1     1     2     2     2     2
     1     2     3     0     1     2     3
    % design constraints
cons   = cell4('u', [- 10 10], 'du', [-5 5], 'ovr', 0.2 );
pri    = cell4('gain',[ 1 2 0.99 1.1]); % prior knowledge
maxper = 3;                         % maximum prior control period
```

The name of the resulting C-encoded controller:

```
cmodule = 'ctrl';
```

The control design is carried out by:

```
cresult = acdesign(data, des, maxstr, cons, pri, maxper, cmodule);
```

The design result is the C-encoded controller of the name `ctrl.c` located the current working directory and the structure `cresult` that carries all information about the design.

Let us display the processing results

```
dispreg(cresult);           % display of results
---------------
Project summary
---------------
data processed:      signals
model description:   oi
maximum prior period: 3
maximum prior model structure:
     1     1     1     2     2     2     2
     1     2     3     0     1     2     3
prior knowledge:
cell4( 'gain', [1 2 0.99 1.1 ])
design constrains:
cell4( 'u', [-10 10 ], 'du', [-5 5 ], 'ovr', 0.2 )
---------------
Project results
---------------
estimated control period: 1
estimated model structure:
     1     1     2     2
     1     2     0     1
estimated forgetting: 0.99
controller penalization:  1   0.000820373   0.00483143
-------------------------------------
Close loop performance evaluation
-------------------------------------
(*) means relation of the samples in user defined limits
    to the total number of samples

mean of the input          0.0530541
std.  of input                 1.37447
(*) for inputs             1
(*) for input difference   1
(*) for step responce      0.8648
mean of the output         -0.00208179
std of output                 0.00354277
steady state error         0.0607092
(*) for output             1
overshoot                  0.166953
-------------------------------------
```

2.3 Controller implementation

2.3.1 Controller implementation in control system

Implementation in a C encoded control program is expected. The controller code is `ctrl.c`.

The controller is implemented and called from the control program as follows:

```
#include "ctrl.c"
void main(void)
{  float  signal [3];
    float  u,y,setp;
// control loop
   // get system output y and setpoint setp
   signal[0]=y; signal[1]=u; signal[2]=setp; // fill signal
   ctrl(signal);      // compute input
   u = signal[1];     // get input from signal
// end of control loop
}
```

There is a function `getptrs` that allows access to internal complex controller structure. It is called as follows:

```
double* ptrs[3];
getptrs(ptrs);
```

The function returns 3 pointers to:

ptrs[0]	penalisations
ptrs[1]	setpoint filter constant
ptrs[2]	forgetting factor

2.3.2 Implementation in MATLAB

The C-encoded controller can be compiled into MEX-file and used in a MATLAB run. With the controller, a run-time library must be included when compiling with the MATLAB supplied batch `cmex`. The library is available on request (for MATLAB 5.2 and WATCOM 11).

The use of the MEX-file is:

```
% control loop
y = ...; setpoint = ...;                         % get output
signal(1)=y; signal(2)=y; signal(3)=setpoint; % fill signal
signal = ctrl(signal);                           % compute input
```

Using the results of design, the controller can be build in workspace by the function `encreg`. The controller is referred to as encapsulated controller. The is used as follow:

```
% control loop
y = ...; setpoint = ...;                         % get output
signal(1)=y; signal(2)=y; signal(3)=setpoint; % fill signal
signal = enccall(signal);                        % compute input
```

The function `setreg` is used to access penalizations (code 'p'), reference filter constant (code 'a') and forgetting factor (code 'f'). It is used as follow:

```
% encapsulated controller
frg   = setreg('frg', 0.97)  % set forgetting
frg =
    0.9900
frg   = setreg('frg')        % get forgetting
frg =
    0.9700
% MEX - file ctrl.dll
frg   = setreg('ctrl','frg', 0.97)  % set forgetting
frg =
    0.9900
frg   = setreg('ctrl','frg')        % get forgetting
frg =
    0.9700
```

2.3.3 Example of implementation

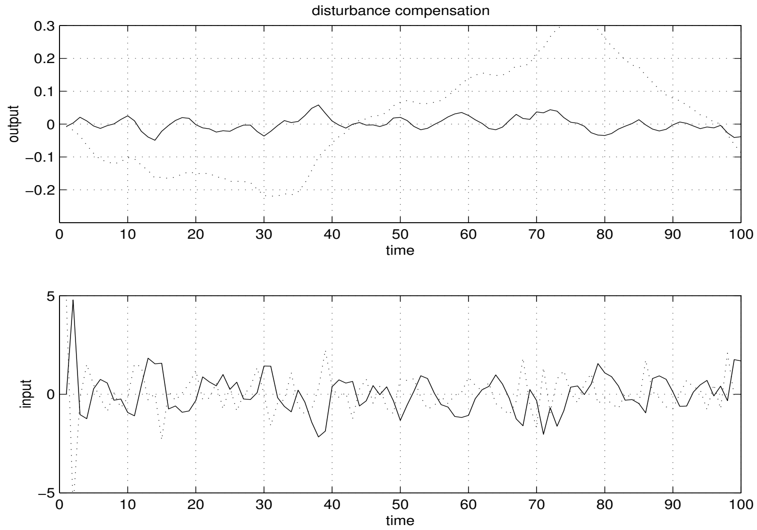
Let us show how to use the controller compiled in the previous section. This example is no verification of the controller but, it shows just one trajectory of the control loop in the case that signals are generated by the "true" system.

The setpoint below is a step function defined in such a way that disturbance compensation (first part) and step response can be displayed (separately):

```
sigma    = 0.01;
sys      = testsys(sigma);  % build test system
u        = 0;               % initial condition for input
setpoint = [zeros(1,100), ones(1,50) ]; % step function
Signal   = [];              % matrix to record signal
randn('seed', 123);         % fix random generator

for t=1:length(setpoint);
       [y, sys] = testsys(u, sys);     % generate output
       signal   = [y u setpoint(t)];  % signal
       Signal   = [Signal,signal'];   % record signal
       signal   = ctrl(signal);       % controller
       u        = signal(2);          % get input
end
```

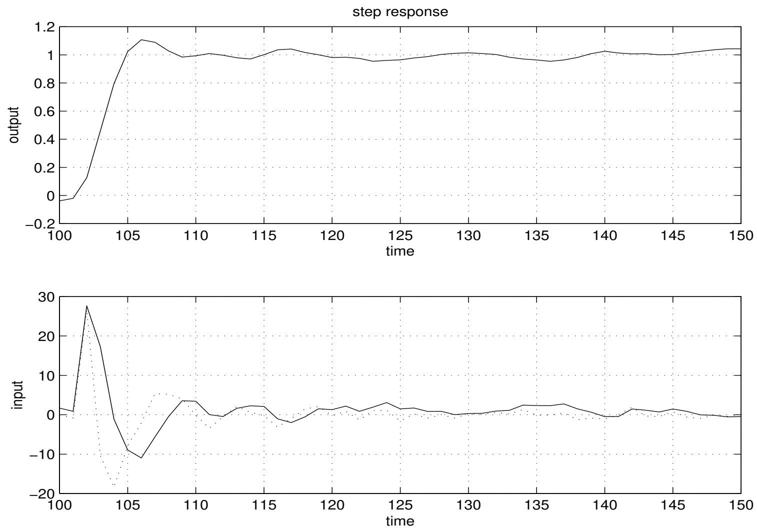
The results are displayed:



Disturbance compensation

The upper plot shows output a non-controlled output (dotted line). The lower one shows input and input difference (dotted line).

The step response is displayed:



Step response and disturbance compensation

3 SISO control design steps

This section summarizes individual steps of control design in SISO case and ends by an example of overall control design.

3.1 Data preprocessing

Signal values can be preprocessed (transformed and/or sampled and held) in ABET design.

There are two main reason for preprocessing in adaptive control design:

1. data normalisation: substantially different signal levels lead to numerical problems. It is recom-

mended to normalise data into mean 0 and standard deviation 1;

2. data sampling: if the control period is not 1, signals must be sampled. Between two samples, the signal level is held on the average value of previous sampling period (other strategies are available, too).

The information about the processing is encoded in a cell list. The following options are available:

option	meaning	qualitative description
'd'	time delay	value
'l'	signal limits	[minimum, maximum]
's'	scaling	see (*)
'i'	incremental mode	'on'
'f'	filtering	[parameters; states]
'H'	sample and hold	frequency
'c'	channel specification	channel number see (**)

(*) the specification consists of a vector of 2 elements. The 1st is a number that is added to the current signal value and result is multiplied by the 2nd vector element.

(**) The channel specification separates specification for individual channels.

The options in the first part of the table above are applied in the order how they are listed.

Example 1: data normalisation for channel 2 is encoded as:

```
channel = 2;
pre = cell4('channel', channel, 'scale',
      [-mean(data(channel,:)), 1/std(data(channel,:))]);
```

Example 2: sampling of all channels is expressed as:

```
pre = cell4('Hold', 10);
```

specifies that the signals will be sampled each 10 simulation steps.

The signals can be preprocessed by the function `datahandle`,e.g.

```
pre = cell4('c', 2, 'scale', [0 10]);
data = datahand(data, pre);
```

transforms data - the data second row is multiplied by 10.

The data normalization is done by:

```
[data, list] = datahand(data,'normalize')
```

The cell list used internally for preprocessing is returned.

The data sampling and holding with the frequency 10 is done by:

```
pre = cell4('Hold',10):
data = datahand(data, pre);
```

3.1.1 Scaling of requirements

In the controller design, the following transformations of signals are usual (`per` is an control period greater that 1):

```
[data, pre] = datahand(data, 'normalize');  % data normalization
holder = cell4('Hold', per);         % requirement for sample and hold
data   = datahand(data, holder);     % sample and averaged data
pre    = cell4(pre,holder);          % preprocessing requrements
```

The cell vector `pre` is requirement for preprocessing of signals. The requirements for prior knowledge processing `pri` and the design constraints `cons` must be transformed, too. The function `scaleit` is designed for the purpose:

```
pri    = scaleit(pri , des, 'pri',  pre); % prior knowledge
cons   = scaleit(cons, des, 'cons', pre); % design constraints
```

3.1.2 Global data option

A global matrix containing data can be referred to by the matrix name in a function call. It means the commands

```
data = ...
fun(data,....)
```

can be substituted by:

```
global data
data = ....
fun('data',....)
```

This is recommended procedure that accelerates processing.

3.2 Prior knowledge processing

By prior knowledge we mean user supplied knowledge about various controlled system features such as system static gain or step responses.

Under prior knowledge we mean also sample dynamic signals or steady-state system measurements.

The types of prior knowledge and their coding are given in description of the function `prior`.

The toolbox design makes it possible to exploit available prior knowledge in whole cycle of adaptive control design.

The prior knowledge is specified by a cell list.

Example: several steady-state signals are assumed to be stored in a matrix `stddata`. The system static gain of the output 1 due to unit change of the input 2 is in the range $< 0.9, 1.1 >$.

The encoded prior knowledge is the matrix `pri`:

```
pri = cell4('measurements', stddata, 'gain', [1 2 0.9 1.1]);
```

The prior knowledge is processed by the function `prior`. The function needs the prior knowledge, the signal description `des` and the model structure `str`. The result of prior knowledge is a model referred to as prior model:

```
Pri = prior(des, str, pri);
```

3.3 Control period estimation

Data are sampled with a sampling frequency. The control period is expressed in multiples of the sampling frequency. ABET timing is expressed in multiples of the sampling frequency, it means 1,2,3,...

For the task of control period estimation the following prior knowledge must be available:

- maximum prior period to be considered in the search for optimum control period;
- maximum possible model structure where the MAP structure estimate is searched for.

The function `peride` searches for a model structure with the highest posterior probability in a space of competitive control-period-dependent model structures and returns the optimum control period.

Processing example:

```
data   = testdata(300, 0.01);          % signals
des    = 'oi';                         % signal description
maxstr = rgrord(des, 3)                % maximum prior model structure
maxstr =
     1     1     1     2     2     2     2
     1     2     3     0     1     2     3
maxper = 3;                              % maximum prior control period
per    = peride(data,des,maxper,maxstr)  % control period estimation
per =
     1
```

The result is usual, the control period is one if the true model structure lies in the set of competitive model structures.

3.4 Model structure estimation

The task of model structure estimation searches for the model structure with highest posterior probability in a space of competitive model structures.

The space of competitors is expressed as a a maximum possible model structure.

Example of processing:

```
data = testdata(120, 0.01);       % signals
des` = 'oi';                      % signal description
% maximum prior model structure
maxstr = [ 1 1 1  2 2 2 2; 1 2 3  0 1 2 3];
pri    = cell4('gain',[ 1 2 0.99 1.1]); % prior knowledge
str    = eststr(data, des, str) % structure estimation
str =
     1     1     2     2
     1     2     0     1
```

The true model structure has been estimated.

3.5 Forgetting estimation

The forgetting is a technique that allows to forget obsolete model attributes and build them from new signals in recursive parameter estimation.

The task of forgetting estimation selects forgetting factor (positive scalar ≤ 1) that determine tracking rate of the used forgetting technique. With the forgetting factor 1, no forgetting of obsolete information is done.

Two types of forgetting techniques are implemented in ABET – exponential and generalised exponential [2]. Each function that uses forgetting allows specification of the technique to be used.

The generalised exponential forgetting requires an alternative ARX model. The model attributes are forgotten by the forgetting factor and corrected by the alternative model. As a rule, a prior model is used as the alternative ARX model with each signal item processed.

The function `estfrg` is designed for forgetting estimation:

```
data = testdata(200, 0.01, 111);     % signals
des = 'oi';                          % signal description
str  = [ 1 1 2 2; 1 2  0 1 ];        % model structure
frg  = estfrg(data, des , str)       % forgetting factor
frg =
     1
```

The result shows that no forgetting is required. This is a common result in the case of simulated data with model structure that correspond to the model structure used in simulation.

Use of prior knowledge will be documented. The prior model is build by:

```
pri    = cell4('gain',[ 1 2 0.99 1.1]); % prior knowledge
Prior = prior(des, str, pri);    % prior model
```

The function `estfrg` allows to specify an initial (4th argument) and an alternative (5th argument) model. We recommend to use the prior model for both of them as follows:

```
frg  = estfrg(data, des , str, Prior, Prior)
frg =
     1
```

3.6 Parameter estimation

The parameter estimation means to update the model attributes by signals. The model before parameter estimation is referred to as initial model, after processing as an identified model (if necessary).

The function `estpar` carries out the parameter estimation.

```
data  = testdata(200, 0.01, 111);       % signals
des   = 'oi';                           % signal description
str   = [ 1 1 2 2; 1 2  0 1 ];          % model structure
frg   = 0.98;                           % forgetting factor
Model = estpar(data, des , str, frg); % estimation

disp(  getfld(Model,'Eth')  ); % mean of model coefficients
     1.8282   -0.8409    0.0042    0.0027
```

The function `estpar` allows us to specify an initial (5th argument) as well as an alternative (6th argument) model. Specification of the alternative model implies use of generalised exponential forgetting. We recommend to use a prior model for both of them as follows:

```
pri   = cell4('gain',[ 1 2 0.99 1.1]); % prior knowledge
Pri   = prior(des, str, pri);          % prior model
Model = estpar(data, des , str, frg, Pri, Pri); %estimation
disp( getfld(Model,'Eth') );
1.8230   -0.8361    0.0043    0.0026
```

3.7 Control design

LQ control synthesis is based on the linear ARX model and quadratic criterion. By control design we mean determination of the weights (penalisations) in the criterion.

The function `cdesing` tailors the adaptive controller according to previous design steps (resulting in a model) and the constraints placed on the closed loop behaviour:

```
cfinal = cdesign(Model, des, cons)
```

The function returns the structure `cfinal` that is used to generate C-encoded controller and to predict control loop performance.

The function inputs are `Model` as the result of previous design steps, `des` signal description and `cons` design constraints (a cell list).

The constrains are ranges for input, input increment and overshoot, e.g.

```
cons = cell4('u', [- 10 10], 'du', [-5 5], 'ovr', 0.2 );
```

specify that the controller is to be designed in such a way that the input range input difference range and overshoot fulfil the constraints specified. Example of the control design follows.

```
cons   = cell4('u', [- 10 10], 'du', [-5 5] ); % design constraints
cfinal = cdesign(Model, des, cons);            % control design
```

3.8 Control loop performance prediction

The function `requsat` calculates the statistics of the basic signals determining the closed loop performance. The evaluation is based on results of previous design steps.

The mean values and variances of the input and output are calculated for uncertain system represented by the mean value and covariance matrix of its parameters obtained in parameter estimation design step. The evaluation is based on Monte Carlo simulation. All results are grouped in an output cell vector `perf`.

Processing can be done as follows:

```
perf = requsat(Model, des, cfinal, pre);
```

The structure `perf` can be displayed by the function `dispperf`:

```
dispperf(perf);
-------------------------------------
Close loop performance evaluation
-------------------------------------
(*) means relation of the samples in user defined limits
    to the total number of samples (in percent)

mean of the input          0.0530541
std of input               1.37447
(*) for inputs             1
(*) for step responce      0.8648
(*) for input difference   1
mean of the output         -0.00208179
std of output               0.00354277
steady state error         0.0607092
(*) for output             1
overshoot                  0.166953
-------------------------------------
```

The results displayed enable to compare initial user's requirements and wishes.

3.9 Controller code generation

The last stage of design of an adaptive controller is generation of an C-encoded controller that makes the same processing as the controller designed under MATLAB. The code is independent to MATLAB processing and can be used in a real-time control program as well as the MEX-file in MATLAB.

The function `encreg` is used to generate the C code:

```
module = 'ctrl';
Reg = encreg(Model, cfinal, frg, pre, module);
```

The input arguments specify:

`Model`	the identified model
`cfinal`	the result of control design
`frg`	forgetting factor
`pre`	preprocessing
`cmodule`	name of the module (.c is appended)

The name of the C-encoded module is optional. The function `encreg` builds in workspace an "encapsulated controller" that can be used by the function `enccall`.

The output structure `Reg` contains all information needed for building the encapsulated controller. This is the second use of the function `encreg`:

```
encreg(Reg);
```

See chapter 2 how the controllers are interfaced.

3.10 Summary example

An example of overall adaptive controller desing follows. To show all processing options, after the control period estimation, the control period is set to 2.

The data are generated by the test system with output std.

```
sigma  = 0.01;                         % std. deviation of output noise
```

The design input arguments are:

```
data   = testdata(600, sigma);         % signals
des    = 'oi';                         % signal description
maxper = 3;                            % maximum prior control period
maxstr = rgrord(des , 3);              % maximum prior model structure
pri    = cell4('gain',[ 1 2 0.99 1.1]); % prior knowledge
                                       % design constraints
cons   = cell4('u', [- 10 10], 'du', [-5 5], 'ovr', 0.2 );
```

Control period selection and data preprocessing follow.

```
% data normalization
[data, scale] = datahand(data, 'normalize');
% control period estimation
per = peride(data,des,maxper,maxstr)
per =
     1
per = 2;  % simulated control period
% sample data
holder = cell4('Hold', per);       % requirement to sample
data   = datahand(data, holder);   % reduced data
pre    = cell4(scale,holder);      % preprocessing
disp( size(data) );
     2   300
```

Data processing design steps follow.

```
% scale prior knowledge
pri    = scaleit(pri , des, 'pri',  pre); % prior knowledge
% processing steps
str    = eststr(data, des , maxstr, pri)   % model structure estimation
str =
     1     1     2     2
     1     2     0     1
Pri    = prior(des, str, pri);             % prior model
frg    = estfrg(data, des, str, Pri, Pri)  % forgetting factor
frg =
    0.9900
Model  = estpar(data, des , str, frg, Pri, Pri); % parameter estimation
```

Control design step with results displayed:

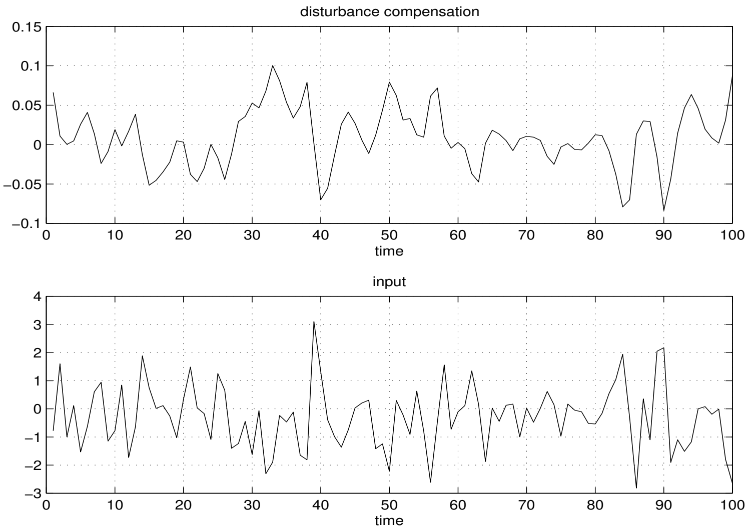
```
% scale constraints
cons   = scaleit(cons, des, 'cons', pre); % design constraints
% control design
cfinal = cdesign(Model, des, cons);        % control design
dispreg(cfinal);
controller penalization:  1   0   0.00423788
time constant of setpoint filter: 0.33
```

Control loop performance evaluation follows with results displayed.

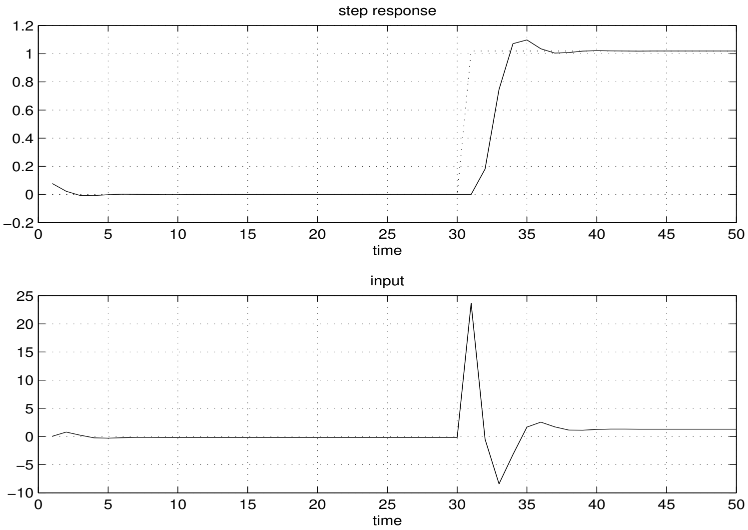
```
[perf,C,C1] = requsat(Model, des, cfinal, pre); % control loop performance
dispperf(perf);
-------------------------------------
Close loop performance evaluation
-------------------------------------
(*) means relation of the samples in user defined limits
    to the total number of samples (in percent)

mean of the input          -0.190558
std of input                1.40142
(*) for inputs             1
(*) for step responce      0.976
(*) for input difference   0.998586
mean of the output         0.00159441
std of output              0.00862223
steady state error         0.00820015
(*) for output             1
overshoot                  0.190048
-------------------------------------
```

The optional output arguments `C` and `C1` contain sample trajectories:



Disturbance compensation (C)



Step response (C1)

The last optional step is generation of C-encoded controller and build the encapsulated controller in the workspace.

```
cmodule = 'ctrl'; % name of the module
Reg     = encreg(Model,des,cfinal,frg,pre,cmodule,Pri); % build controller
```

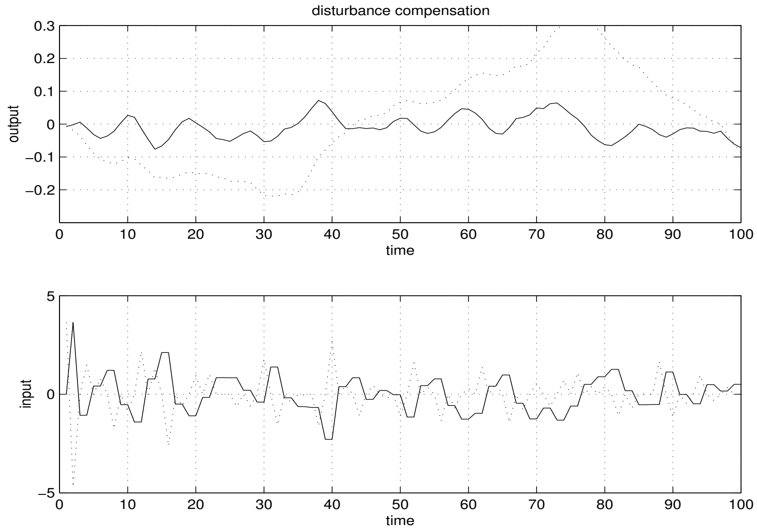
Let is show use of encapsulated controller. The example is no verification of the controller but it shows one trajectory with the "true" system.

The setpoint below is a step function defined in such a way that disturbance compensation (first part) and step response can be displayed (separately):

```
sigma    = 0.01;
sys      = testsys(sigma);  % build test system
u        = 0;               % initial condition for input
setpoint = [zeros(1,100), ones(1,50) ]; % step function
Signal   = [];              % matrix to record signal
randn('seed', 123);         % fix random generator

for t=1:length(setpoint);
       [y, sys] = testsys(u, sys);     % generate output
       signal   = [y u setpoint(t)];   % signal
       Signal   = [Signal,signal'];    % record signal
       signal   = enccall(signal);     % controller
       u        = signal(2);           % get input
end
```

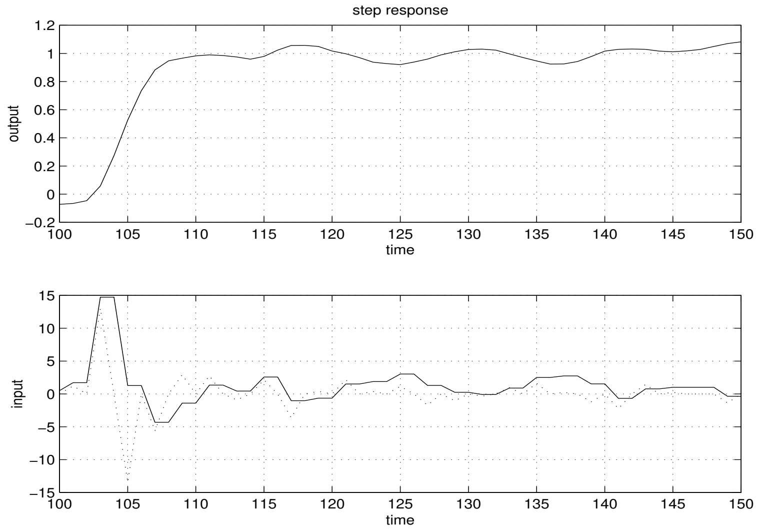
The results are displayed:



Disturbance compensation

The upper plot shows output a non-controlled output (dotted line). The lower one shows input and input difference (dotted line).

The step response is displayed:



Step response and disturbance compensation

3.11 Controller verification

The controller can be verified in the real control system (see chapter 2) or in MATLAB a SIMULINK environment. A SIMULINK model of the controlled system must be available.

The function `versim` verifies the behaviour of the pretuned LQ adaptive controller. The closed loop experiment is set up imitating the situation as closed as possible to the reality. The SIMULINK environment was used for these experiment as:

- it makes possible to connect the computer to a real controlled system in a simple way (e.g. by using a Real time toolbox);
- it allows a modelling of complex nonlinear systems in a relatively simple way.

User has to program its specific SIMULINK scheme for the connection to a real system or must provide the system model as a Simulink block together with the generators of disturbances and setpoint relevant to the system. Necessary information is encoded in the structure `System`. Function `versim` merges user defined `System` with internal Simulink scheme where are realised necessary blocks for verification. The scheme consists of:

- parameter estimation block
- controller synthesis (design) block
- controller block
- various blocks analysing the control process and calculating values consired in control specification response.

These blocks are set up according to the previous ABET steps. The minimum length of the experiment(its duration) is also set by ABET as a default if no user value is given in `System`.

The verification performs the function `versim`:

```
Ver = versim(system,scales,Model,csetup,cons,frg,Alt)
```

4 Experiments

This section summarizes functions that are designed for experimentation in individual design steps. The summary is related to technical topics, no attempt is made to teach the methods available

The experimental features are designed for SISO or MISO case - instead the signal description, the output channel `ychn` is used.

4.1 Model structure estimation

The structure estimation is the most demanding step of a controller design. The used of prior information in design is demostrated:

```
data = testdata(120, 0.01, 85); % signals
des‘ = ’oi’;                    % signal description
% prior model structure
maxstr = [ 1 1 1  2 2 2 2; 1 2 3  0 1 2 3];
str    = eststr(data, des, str) % structure estimation
str =
     1     1     2     2
     1     2     0     1
```

The true model structure has been estimated. Let us now omit the 20 measurements and repeat the estimation:

```
data = data(:,1:length(data)-20);
disp( eststr(data, des, str) );
      1     1
      1     2
```

The result claims that the output does not depend on input. Let us show that our prior knowledge about the system gain may substitute the lack of signal-supported information:

```
pri    = cell4('gain',[ 1 2 0.99 1.1]); % prior knowledge
disp( eststr(data, des, str, pri) );
      1     1     2     2
      1     2     0     1
```

This simple example indicates that use of prior knowledge may be vital for the task of structure estimation.

There is another possibility to influence the structure estimation. The user can express his believe to the prior model structure. It specifies for each possible structure item (the pair of channel and delay) that it is:

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

in the resulting model structure.

The believe is added to the maximum prior model structure as an additional row. In the example below, we specify that the output surely depends on the input with delay 0 and probably on other items:

```
believe = [ 2 2 2  1 2 2 2];
disp( eststr(data, des, [str; believe]) );
      1     1     2
      1     2     0
```

Two functions has been designed to make experiments with structure estimation.

4.1.1 Repeated estimation

The function `estrrep` makes repeated structure estimation with random start and records the "best structures" in the sense of MAP estimate. The function `estrrep1` is an analogy but, it does not process data - the statistics is entered as the 1st and 2nd argument.

The input arguments are:

data	signals
ychn	output channel
str	maximum prior model structure with possible believe added as an additional 3th row
nbest	number of best structures recorded
nrun	number of repetitions

The function returns:

LH	log-likelihood of the best structures
strs	best structures recorded
LHs	log-likelihood of individual runs
nomax	number of local maximums

Commented example follows:

```
data   = testdata(100, 0.01, 111);              % data
ychn   = 1;                                     % output channel
maxstr= [ 1 1 1  2 2 2 2; 1 2 3  0 1 2 3];     % prior structure
mask   = [ 2 2 2  1 2 2 2];                     % believe
nbest  = 10;                                    % number of best structures
nrun   = 10;                                    % number of estimations

[LH,strs,LHs,nomax] = estrrep(data,ychn,[str; mask], nbest, nrun);
strs    % best structures
strs =
     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     1     1     1     1
     0     0     0     0     1     1     0     0     0     0
     0     1     0     1     0     0     0     0     1     0
     0     0     1     1     0     1     0     0     1     1
     0     0     0     0     0     0     1     0     1     1
     0     0     0     0     0     0     0     1     0     1
```

Each line of the matrix correspond to one item in the prior model structure. One means that the item was present, 0 that was not present in the prior model structure.

```
LH      % likelihoods of the structures
LH =
  Columns 1 through 7
  404.1019  403.9507  399.4838  399.2916  399.0773  395.4407  391.9000
  Columns 8 through 10
  390.8123  387.8383  373.7792
LHs     % likelihoods of individual runs
LHs =
  Columns 1 through 7
  404.1019  404.1019  404.1019  404.1019  404.1019  404.1019  404.1019
  Columns 8 through 10
  404.1019  404.1019  404.1019
nomax  % number of local likelihood maximums discovered
nomax =
     1
```

The estimation above can be interpreted as distance of the models with different structure. The number of local maximums discovered shows the quality of information hidden in data.

4.1.2 Global search

The function makes global search of the prior structure in subspaces limited by the believe. Note that the search is very time consuming. The "best structures" are recorded.

Example follows.

```
data  = testdata(100, 0.01, 111);              % data
ychn  = 1;                                     % output channel
str   = [ 1 1 1  2 2 2 2; 1 2 3  0 1 2 3];     % prior structure
mask  = [ 2 2 2  1 2 2 2];                     % believe
nbest = 10;                                    % number of best structures

[LH,strs]=estrglob(data,ychn,[str; mask], nbest);
strs    % best structures
strs =
     1     1     1     1     1     1     1     1     1     1
     1     1     1     1     1     1     0     1     1     1
     0     0     1     1     0     0     1     0     1     0
     1     1     1     1     1     1     1     1     1     1
     0     1     0     1     0     0     1     1     0     1
     0     0     0     0     1     0     0     1     1     0
     0     0     0     0     0     1     0     0     0     1
LH      % likelihoods of the structures
LH =
  Columns 1 through 7
  403.9507  399.2916  399.0305  396.0248  392.3398  390.2639  388.1186
  Columns 8 through 10
  387.8383  387.7119  385.6721
```

4.2 Multiple ARX models

A multiple ARX model is based on a collection of n ARX models for the same output. The models are referred to as partial models. The multiple model output at a time t is generated by a partial model. The (partial) model probabilities $p(i), i = 1, ..., n$ define which partial model "works" at the time t.

The estimates of these probabilities are updated by signals according to likelihood of partial models. A second level forgetting factor is used for forgetting of the model probabilities.

The multiple ARX model structure is the cell vector of structures for each partial model. The model attributes are attributes of the partial ARX models and the model probabilities (p).

Note: the multiple ARX model is the cell vector of partial models for the same output while a MIMO model is the cell vector of models for each output.

Multiple model estimation is solved by the function `estmul`. The estimation of forgetting (see section "Forgetting") is based on this model and in this section, the same problem is solved by multiple model estimation. Data are simulated and general framework of the task is:

```
data  = testdata(200, 0.01, 111);      %  signals
des   = 'oi';                          %  signal description
frgs  = 0.8:0.01:1;                    %  forgetting factors
nmod  = length(frgs);                  %  number of models
frgup = 0.99;                          %  second level forgetting
```

The model structure as well as prior model are specified for each partial model as a cell vector e.g.:

```
str   = [1 1  2 2; 1 2  0 1];         % partial model structure
STR  = cell4(str);                    % multiple model structure
for i=2:nmod, STR=setcell(STR,i,str); end
pri   = cell4('gain',[ 1 2 1 ]);      % partial model knowledge
Pri   = prior(des, str, pri);         % partial prior model
Prior = cell4(Pri);                   % multiple prior model
for i=2:nmod, Prior=setcell(Prior,i,Pri); end
```

The prior model will be used as initial (6th argument) as well as the alternative one (7th argument). The estimation follows:

```
[Models, loglh, Eth, ep] = ...
     estmul(data, des, STR, frgs, frgup, Prior, Prior);
```

The function returns cell vector of partial models `Models` and (optionally) trajectories of log-likelihoods of partial models `loglh`, cell vector of trajectories of partial models parameters estimates `Eth` and trajectories of prediction errors `ep`.

The MAP estimate of forgetting follows:

```
size(loglh)
ans =
21   200
datalen = size(data,2);
lh = loglh(:,datalen);      % last step log-likelihood
[m,iMAP]=max(lh);           % find iMAP estimate
frgs(iMAP)
ans =
1
```

The MAP model is extracted from `Models` and displayed:

```
size4(Models)
ans =
21
modelMAP = getcell(Models,iMAP); % MAP model
disp4(modelMAP);
ABET structure with 6 fields
1) ychn :double   = 1
2) str :double  (2,4) = 1  1  1  2  2  ...
3) covy :double   = 0.000107993
4) Eth :double  (4) = 1.8487  -0.859681  ...
5) Cth :double  (4,4) = 7.68142  -0.994143  ...
6) dfm :double   = 204
```

The trajectories of parameters are returned as a cell vector. The MAP model trajectory of parameters `EthMAP` are extracted by:

```
EthMAP = getcell(Eth,iMAP); %  parameters of iMAP model
size(EthMAP)
ans =
4   200
mean(EthMAP')
ans =
1.7889   -0.7990    0.0036    0.0048
```

The trajectory of prediction error of the MAP model `epMAP`:

```
epMAP = ep(iMAP,:);          %  prediction error of iMAP model
size(epMAP)
ans =
1   200
mean(epMAP)
ans =
-0.0031
```

If the values of log-likelihood are small enough (< 720), the trajectory of log-likelihods can be converted into model probabilities `pmod`, e.g.:

```
pmod = exp(loglh); sumpmod = sum(pmod);
for i=1:nmod; pmod(i,:) = pmod(i,:)./sumpmod; end
```

With the trajectories of the partial models probability we can compute e.g. mean model probability `cum` and mean forgetting:

```
cum = sum(pmod');            %  sum of partial model probability
cum = cum/sum(cum);          %  normalized
frgs*cum'                    %  "mean forgetting"
ans =
0.9551
```

MO cases are not supported.

4.3 Forgetting estimation

Effects of estimation of forgetting will be mentioned. Below, signals are simulated, the signal description and model structure encoded and forgetting factor estimated by the function `estfrg`:

```
data = testdata(200, 0.01, 111);     % signals
des = 'oi';                          % signal description
str  = [ 1 1 2 2; 1 2  0 1 ];        % model structure
frg  = estfrg(data, des , str)       %  estimate forgetting
frg =
     1
```

The result shows that no forgetting is required. This is a common result in the case of simulated data with model structure that correspond to the model structure used in simulation.

Now an under-estimated structure will be used for estimation of forgetting:

```
str  = [ 1 2; 1 0];          %  under-estimated structure
frg  = estfrg(data, des, str)
frg =
    0.8500
```

The result shows that a high forgetting (low forgetting factor) is to be used.

The estimation of forgetting is based on a multiple model. The partial models have the same model structure but differs in forgetting factors (e.g. recorded in a vector `frgs`). The second level forgetting factor is needed for forgetting of the model probabilities estimates (e.g. encoded in `frgu`). These quantities can be specified as arguments of `estfrg`. Not that this experimental use of `estfrg` differs from the one previously explained. Exampla follows.

```
frgs = 0.8:0.01:1;       % forgetting factors of individual models
frgu = 0.98;             % second level forgetting factor
frg  = estfrg(data, des, str, frgs, frgu)
frg =
     0.85
```

Note: the values of `frgs` and `frgu` used here are the function defaults.

Use of prior knowledge will be documented. The prior model is build by:

```
pri    = cell4('gain',[ 1 2 0.99 1.1]); % prior knowledge
Prior = prior(des, str, pri);     % prior model
```

The function `estfrg` allows to specify an initial (5th argument) and an alternative (6th argument) model. We recommend to use the prior model for both of them as follows:

```
frg  = estfrg(data, des , str, [], [], Prior, Prior)
frg =
     0.8500
```

Now, the estimation of forgetting factors will be demonstrated for MO case. The result is vector of forgetting factors for individual outputs:

```
data1= testdata(200,0.01,111);
data2= testdata(200,0.01,111);
data = [data1;data2];                %  signals
des   = 'oioi';                      % model description
str1 = [1 1  2 2 ; 1 2  0 1 ];       % structure, 1st model
str3 = [3    4   ; 1    0   ];       % structure, 2nd model
str  = cell4(str1,str3);             % model structure
frg  = estfrg(data, des , str)
frg =
    1.0000    0.8500
```

Note that the structure for the 2nd model is under-estimated.

The use of prior knowledge in MO cases follows.

```
pri  = cell4('gain',[1 2 1], 'g',[3 4 1]);  % encoded prior
Prior= prior(des , str, pri);              %  prior model

frg  = estfrg(data, des , str, [], [], Prior, Prior)
frg =
1.0000    0.8500
```

The function `estfrg1` can be used for SO model. It returns optional second output argument – (log) likelihood of individual models:

```
output = 1;              % which channel is output
str    = [ 1 2; 1 0];    % under-estimated model structure
frgs   = 0.8:0.01:1;     % default value of forgetting
[frg,loglh]  = estfrg1(data, y, str,frgs);
frg
frg =
    0.8500
```

The loglikelihoods can be converted to model probabilities if the value is below 710. Then, mean forgetting can be computed:

```
pmod = exp(loglh);
pmod=pmod/sum(pmod);      % model probabilities
frg  = frgs*pmod'         % mean forgetting
frg =
    0.8707
```

4.4 Parameter estimation

The `estpar1` function can be used for SO case. It returns all trajectories of estimated coefficients:

```
data = testdata(200, 0.01, 111);
ychn = 1;
str  = [ 1 1 2 2; 1 2  0 1 ];
[Model, Eth] = estpar1(data, y, str);
size(Eth)
ans =
      4   201
```

Mean parameters can be calculated as:

```
sum(Eth')/200
ans =
     1.7584   -0.7699    0.0035    0.0030
```

5 References

References

[1] J. Bůcha and M. Kárný and P. Nedoma and J. Böhm and J. Rojíček, "Designer 2000 project", in *Preprints Control 98*. 1998, accepted.

[2] R. Kulhavý, "Restricted exponential forgetting in real-time identification", *Automatica*, vol. 23, no. 5, pp. 589–600, 1987.

[3] M. Kárný and P. Nedoma, "Automatic processing of prior information with application to identification of regression model", *Automatica*, 1997, submitted.

[4] M. Kárný, "Algorithms for determining the model structure of a controlled system", *Kybernetica*, vol. 19, no. 2, pp. 164–178, 1983.

[5] Rojíček Jiří, "Preliminary control design under constrains and imcomplete knowledge of the system", Tech. Rep., ÚTIA AV ČR, Praha, 1999, Phd. Thesis.

6 ABET98 functions

6.1 Coding summary

This subsection summarises common rules, identifiers and synopsis of selected ABET functions.

The following rules are common and not mentioned in descriptions:

- global data option can be used for all signals;
- the requirement for an argument default value is expressed by empty matrix. Defaults are specified in the function descriptions.
- SO and MO cases are not explicitly distinguished. In the MO case, the model structure, model, etc. are cell vectors of the specification of individual outputs.

The identifiers used in function descriptions and examples with data type and dimensions follow. This list covers SISO control design steps.

ABET SISO control design coding convention

arguments

	inputs to control design	
`data`	signals	matrix(2, :)
`des`	signal description	char(2)
`maxstr`	maximum prior model structure	matrix(2, :)
`maxper`	maximum prior control period	scalar
`pri`	prior knowledge	cell list
`cons`	control design constraints	cell list
	matrices in design steps	
`per`	control period	scalar
`pre`	preprocessing handle	cell list
`str`	model structure	matrix(2, :)
`frg`	forgetting factor	scalar
`Pri`	prior model	structure
`Model`	identified model	structure
`Ini`	initial model	structure
`Alt`	alternative model	structure
`cfinal`	result of control design	structure
`perf`	control loop performance	structure
`cresult`	result of overall design	structure
	name of C-encoded controller	
`cmodule`	name of C-encoded controller	char(1:8)

functions

```
cresult = acdesign(data,   des,   maxstr, cons, pri, maxper, cmodule)

per     = peride  (data,   des,   maxper, maxstr);
str     = eststr  (data,   des,   maxstr, pri)
Pri     = prior   (des,    str,   pri,    Ini)
frg     = estfrg  (data,   des,   str,    Ini,  Alt)
Model   = estpar  (data,   des,   str,    frg,  Ini, Alt)
cfinal  = cdesign (Model,  des,   cons)
perf    = requsat (Model,  des,   cfinal, pre)

Reg     = encreg  (Model,  des,   cfinal, frg,  pre, cmodule, Pri)
Reg     = encreg  (Reg,    cmodule)
cresult = encreg  (cresult,cmodule)

data    = datahand(data,   pre)
cons    = scaleit (cons,   'u',  pre)
pri     = scaleit (cons,   'p',  pri)
```

6.2 acdesign

Purpose

Adaptive controller design.

Synopsis

`cresult = acdesign(data, des, str, cons, pri, per, module)`

Description

The function tailors an adaptive controller. It covers all controller design steps and returns the structure **`cresult`** that carries all relevant information about the design. The results can be listed using the function **`dispreg`**.

The input arguments specify:

`data`	signals
`des`	signal description
`str`	maximum prior model structure
`cons`	control design constraints
`pri`	prior knowledge (default: not used)
`per`	maximum control period (default: not used)
`module`	name of C-encoded module (default: not used)

Example

```
randn('seed', 1234);            % fix random generator

global data                     % global data
sigma  = 0.01;                  % standard deviation
data   = testdata(600, sigma);  % signals
des    = 'oi';                  % signal description
str    = 3;                     % maximum structure considerred
cons   = cell4('u', [- 10 10], 'du', [-5 5] ); % constraints
cresult = acdesign('data', des, str, cons); % controller design
dispreg(cresult);                % display of results
-------------------------------------
Results of adaptive controller design
-------------------------------------
data processed:       data
control period:       1
estimated model structure:
     1     1     2     2
     1     2     0     1
estimated forgetting: 0.99
estimated model parameters:
    1.8156   -0.8282    0.0214    0.0221
controller penalization:  1   2.94219e-005   0.0005933
-------------------------------------
```

6.3 arxmodel

Purpose

Create ARX model(s) using defaults.

Synopsis

```
Model = arxmodel( des, str, nmod)
```

Description

The function returns model or cell4 matrix of models.

The input arguments are :

des	model description
str	model structure
nmod	number of models (default is 1)

Example

```
des   = 'oi';
str   = [ 1 1  2 2; 1 2  0 1 ];
Model = arxmodel(des, str);
disp4(Model);
ABET structure with 6 fields
1) ychn :double   = 1
2) str :double  (2,4) = 1  1  1  2  2  ...
3) covy :double   = 0.0001
4) Eth :double  (4) = 0  0  0  0
5) Cth :double  (4,4) = 10000  0  0  0  0  ...
6) dfm :double   = 1
```

6.4 arxsys

Purpose

ARX system.

Synopsis

```
Sys = arxsys(Model)
[y, Sys]=arxsys(Sys,u)
```

Description

The function builds the ARX system based on the ARX `Model`.

In a control loop, the function `arxsys` computes output `y` given the system `Sys` and input `u`.

Examples: zarxsys

6.5 cdesign

Purpose

Control design.

Synopsis

```
cfinal = cdesign(Model, des, cons)
```

Description

The function tailors the adaptive controller according to constraints placed on the closed loop behaviour and the identified model.

The constrains are ranges input range, input increment range and overshoot range.

Underlying philosophy and processing algorithms are described in [5]. The design is fully probabilistic, the constraints can be fulfilled only with a probability, see Advanced topics. The design is time consuming due to a great number of simulation runs.

The arguments specify:

`Model`	model
`des`	signal description
`cons`	design constraints
`cfinal`	results of control design

The constraints are encoded in any order in a cell list:

code	meaning
`'u'`	input range
`'d'`	input increment range
`'o'`	overshoot range

The overshoot range can be specified as a scalar value. Then the range of overshoot will be $[value - 0.05, value + 0.05]$. The value of overshoot is specified as signal value (not in percent).

The result of the controller design `cfinal` is used for controller verification or for generation of the C-encoded controller, see `encreg`.

Example

Example of the control design follows. The matrices `Model` and `des` enters from previous steps of the controller design:

```
cons = cell4(...
  'u',       [-10 10 ],...     % input range
  'd',       [ -5  5 ],...     % input increment range
  'o',        0.1 );           % overshoot range

cfinal = cdesign(Model, des, cons);
```

Advanced topics

Signal description: can extended by character 's' that specifies position of setpoint among the processing channels. For example `'osi'` specifies that the adaptive controller uses the channel 1 for output, the channel 2 for setpoint and the channel 3 for input. The default value for setpoint channel is third channel.

Probabilistic measures for the users constraints: each constraint range can be extended by a probabilistic measure placed on the constraint. It is specified by confidence (e.g. 0.9) and its tolerance (e.g. 0.05). Then, for example, the input range coded as `[-1 1 0.9 0.05]` means that the user wants to have input in the range $< -1, 1 >$ with the probability from the interval $< 0.9 - 0.05, 0.9 + 0.05 >$ (the interval must be from interval $< 0, 1 >$). The values above are the function defaults.

Step function: can be specified for the overshoot design. The design is done by selection of an output reference filter. As a default, this is analyzed using unit step function. However, the overshoot in the close loop can be dependent on the signal level and the user can then specify step function for design. It is done by an additional information located among the constraints. It specifies the minimum and maximum of the step function, e.g. :

```
... 'step', [0 1], ...
```

specifies the default step function.

6.6 datahand

Purpose

Signal preprocessing.

Synopsis

```
data = datahand(data, pre)
[data, pre] = datahand(data, pre)
```

Description

Coding of preprocessing

The information about the processing is encoded in the cell list `pre`. The following options are available:

option	meaning	qualitative description
'd'	time delay	value
'l'	signal limits	[minimum, maximum]
's'	scaling	see (*)
'i'	incremental mode	'on'
'f'	filtering	[parameters; states]
'H'	sample and hold	frequency
'c'	channel specification	channel number see (**)

(*) the specification consists of a vector of 2 elements. The 1st is a number that is added to the current signal value and result is multiplied by the 2nd vector element.

(**) The channel specification separates specification for individual channels.

The options in the first part of the table above are applied in the order how they are listed.

Example 1: data normalisation for channel 2 is encoded as:

```
channel = 2;
pre = cell4('channel', channel, 'scale',
      [-mean(data(channel,:)), 1/std(data(channel,:))]);
```

Example 2: sampling of all channels is expressed as:

```
pre = cell4('Hold', 10);
```

specifies that the signals will be sampled each 10 simulation steps.

Use of function `datahand`

Data transformation is coded e.g. as

```
pre = cell4('c', 2, 'scale', [0 10]);
data = datahand(data, pre);
```

The processing above transforms data - the data second row is multiplied by 10.

The data normalization is done by:

```
[data, list] = datahand(data,'normalize')
```

The cell list used internally for preprocessing is returned.

The data sampling and holding with the frequency 10 is done by:

```
pre = cell4('Hold',10):
data = datahand(data, pre);
```

6.7 dispreg, disp*

Purpose

Auxiliary display functions. The following functions are available:

`dispreg`	information about control design
`dispperf`	information about control loop performance
`displist`	displays list
`disppro`	internal processing
`disppre`	preprocessing

6.8 encreg, enccall

Purpose

The function `encreg` builds an adaptive controller in workspace according to results of previous design steps. Optionally, an equivalent C-encoded cmodule is generated.

The function `enccall` is designed for calling the adaptive controller in workspace.

Synopsis

```
cresult = encreg(Model, des, cfinal, frg, pre, cmodule, Pri)
cresult = encreg(cresult, cmodule)
signal = enccall(signal)
```

Description

The input arguments of `encreg` are:

`Model`	model
`des`	signal description *
`cfinal`	result of control design step
`frg`	forgetting factor
`pre`	preprocessing
`cmodule`	name of C-encoded cmodule
`cresult`	result of overall design

* description can be extended for reference channel.

Example

Let us the structure `cresult` has been designed by `acdesign` and std. of simulated output `sigma` has been selected.

```
cresult = encreg(cresult,'ctrl');      % build controller
sys      = testsys(sigma);  % build test system
u        = 0;               % initial condition for input
setpoint = zeros(1,200);    % setpoint - zeros
randn('seed', 123);         % fix random generator

for t=1:length(setpoint)
       [y, sys] = testsys(u, sys);
       signal   = [y u setpoint(t)];  % signal
       signal   = enccall(signal);    % controller
       u        = signal(2);          % get input
```

The enccapsulated cmodule makes the same processing. It is compiled by (an internal access to `cmex`):

```
makemex('ctrl');
```

The upper code can be now run as follows:

```
for t=1:length(setpoint)
.......
       signal   = ctrl(signal);     % controller
.......
end
```

6.9 estfrg

Purpose

Estimation of forgetting, ARX model.

Synopsis

```
frg = estfrg( data, des, str, Ini, Alt)
frg = estfrg( data, des, str, frgs, frgup, Ini, Alt)
```

Description

The estimation of forgetting is based on a multiple model. The partial models have the same model structure but differ in forgetting factors from a selection. The estimate of model probabilities is forgotten with a second level forgetting factor.

The partial models parameter estimation start from an initial model. Generalised exponential forgetting can be used for forgetting of partial models – in the case, an alternative model is needed. We recommend using a prior model both in role of the initial and alternative models.

The function returns mean forgetting factor reached in the last recursive parameter estimation step.

The input arguments are :

		defaults
`data`	signals	none
`des`	model description	none
`str`	model structure	none
`frgs`	partial model forgetting factors	0.8:0.01:1
`frgup`	second level forgetting factor	0.98
`Ini`	initial model	default
`Alt`	alternative model	not used

Examples

Example 1: SO processing based on the test system:

```
data = testdata(200, 0.01, 111); % signals
des  = 'oi';                     % signal description
str  = [ 1 1 2 2; 1 2  0 1 ];    % prior model structure
pri  = cell4('gain', [1, 2, 1]); % encoded prior knowledge
Pri  = prior(des, str, pri);     % prior model
%                models:          initial  alternative
frg  = estfrg(data,des,str,'def','def',Pri, Pri)
frg =
     0.9902
```

Example 2: MO processing based on the test system:

```
data12 = testdata(200,0.01,111);
data34 = testdata(200,0.01,222);
data   = [data12; data34];        %  signals
des  = 'oioi';                    % signal description
str1 = [1 1  2 2 ; 1 2  0 1 ];    %  model structure, 1st output
str3 = [3 3  4 4 ; 1 2  0 1 ];    %  model structure, 2nd output
str  = cell4(str1,str3);          %  model structure
pri  = cell4('g',[1 2 1], 'g',[3 4 1]); %  prior knowledge
Pri  = prior(des , str, pri);     %  prior model
frg  = estfrg(data, des , str, 'def','def', Pri, Pri)
frg =
0.9902    0.9895
```

6.10 estmul

Purpose

Parameter estimation, multiple ARX model.

Synopsis

```
[Model, loglh, parms, ep] = ...
           estmul(data, des, str, frgs, frgup, Ini, Alt)
```

Description

Recursive identification is carried out for a group of MISO ARX (partial) models related to one output.

Log-likelihood of partial models are calculated at each identification step. The likelihood are forgotten with a common second level forgetting factor.

The input arguments are :

		defaults
`data`	signals	none
`des`	signal description	none
`str`	multiple model structure	none
	cell vector of partial model structures	
`frgs`	partial model forgetting factors	see (*)
`frgup`	second level forgetting factor	0.99
`Ini`	initial multiple model	default model
	cell vector of initial partial models	
`Alt`	alternative multiple model	not used
	cell vector of alternative partial models	

The function returns:

`Model`	cell vector of identified partial models
`loglh`	trajectory of log-likelihood of individual models
`parms`	cell vector of recursive parameters of partial models
`ep`	prediction error of individual models

6.11 MISO estimation functions

Purpose

Estimation functions, ARX model.

Synopsis

```
str            = eststr1( data, ychn, str, frg, pri)
[frg, pmod]    = estfrg1( data, ychn, str, frgs, frgu, Ini, Alt)
[Model, parms] = estpar1( data, ychn, str, frg, Ini, Alt)
```

Description

Experimental versions of estimation functions. Refer to corresponding functions for details.

The input arguments are :

		defaults
`data`	signals	none
`ychn`	output channel	none
`str`	model structure	none
`frg`	exponential forgetting factor	1
`frgs`	forgetting factors	0.8:0.01:1
`frgup`	second level forgetting factor	0.98
`Ini`	initial model	defaut
`Alt`	alternative model	not used

The functions return:

`str`	model structure
`frg`	mean forgetting for the last recursion step
`pmod`	trajectory of model probabilities
`Model`	identified model
`parms`	trajectory of parameter estimates

6.12 estpar

Purpose

Parameter estimation, ARX models.

Synopsis

`Model = estpar( data, des, str, frg, Ini, Alt)`

Description

The model parameters are identified recursively.

The function returns the identified model. The input arguments are :

		defaults
`data`	signals	none
`des`	model description	none
`str`	model structure	none
`frg`	forgetting factor	1
`Ini`	initial model	default
`Alt`	alternative model	not used

If a prior model is available, we recommend to use it both for initial and alternative model.

If alternative model is not specified, exponential forgetting is applied. If it is specified, generalised exponential forgetting is used. In this case, the forgetting factor < 1 should be specified.

Examples

Example 1: SO processing based on the test system:

```
data  = testdata(200, 0.01, 111);     % signals
des   = 'oi';                         % signal description
str   = [ 1 1 2 2; 1 2  0 1 ];        % model structure
frg   = 0.98;                         % forgetting factor
pri   = cell4('gain', [1 2 1]);       % encoded prior knowledge
Pri   = prior(des, str, pri);         % prior model
Model = estpar(data, des , str, frg, Pri, Pri); % estimation
disp( getfld(Model,'Eth') );
     1.8230   -0.8361    0.0043    0.0026
```

Example 2: MO processing bases on the test system:

```
data1  = testdata(200,0.01,111);
data3  = testdata(200,0.01,222);
data   = [data1; data3];            % signals
des    = 'oioi';                    % signal description
str1   = [1 1  2 2 ; 1 2  0 1 ];    % model structure, 1st output
str3   = [3 3  4 4 ; 1 2  0 1 ];    % model structure, 2nd output
str    = cell4(str1,str3);          % model structure
pri    = cell4('gain',[1 2 1], 'g',[3 4 1]); % prior knowledge
Pri    = prior(des , str, pri);     % prior model
frgs   = [0.98, 0.98];              % forgetting factors
Model  = estpar(data, des , str, frgs, Pri, Pri);
Model1 = getcell(Model,1);
getfld(Model1,'Eth') % parameter estimate, 1st output
ans =
     1.8230   -0.8361    0.0043    0.0026
Model3 = getcell(Model,2);
getfld(Model3,'Eth') %  parameter estimate, 2nd output
ans =
     1.7131   -0.7174    0.0047    0.0049
```

6.13 estpar1

Purpose

Parameter estimation, SISO ARX models.

Synopsis

`[Model, Eth, ep] = estpar( data, ychn, str, frg, Ini, Alt)`

Description

The model parameters are identified recursively.

The function returns the identified model `Model`, trajectory of parameter estimates `Eth` and trajectory of prediction error `ep`.

The input arguments are :

		defaults
`data`	signals	none
`ychn`	output channel	none
`str`	model structure	none
`frg`	forgetting factor	1
`Ini`	initial model	default
`Alt`	alternative model	not used

If a prior model is available, we recommend to use it both for initial and alternative model.

If alternative model is not specified, exponential forgetting is applied. If it is specified, generalized exponential forgetting is used. In this case, the forgetting factor < 1 should be specified.

Examples: zestpar1, zestpar2

6.14 estrglob

Purpose

Experimental ARX model structure estimation - global search.

Synopsis

```
[vLH, iLH] = estrglob(data, ychn, str, nbest)
```

Description

The function makes global search of the prior structure subspaces. Note that the search is very time consuming. The "best structures" in the sense of MAP estimate are recorded.

The input arguments are:

data	signals
ych	output channel
str	prior model structure with possible believe added as additional 3th row
nbest	number of best structures recorded

The function returns:

vLH	likelihood of the best structures
iLH	best structures recorded

Examples: zestrglo

6.15 estrrep

Purpose

Experimental ARX model structure estimation.

Synopsis

```
[vLH, iLH, results, nomax] = estrrep(data, ychn, str, nbest, nrun)
[vLH, iLH, results, nomax] = ...
          estrrep1(V,n, ychn, str, nbest, nrun)
```

Description

The function makes repeated structure estimation with random start and record the "best structures" in the sense of MAP estimate. The function `estrrep1` is an analogy but, it does not process data - the statistics is entered as the 1st and 2nd argument.

The input arguments are:

data	signals
ych	output channel
str	prior model structure with possible believe added as additional 3th row
nbest	number of best structures recorded
nrun	number of repetitions

The function returns:

LH	likelihoods of the best structures
strs	best structures recorded
LHs	likelihoods of individual runs
nomax	number of local maximums

Examples: zestrrep, zestrrp1

6.16 eststr

Purpose

ARX model structure estimation.

Synopsis

`str = eststr( data, des, str, pri)`

Description

The task of model structure estimation searches for the model structure with highest posterior probability in a space of competitive model structures.

The user specifies the space of competitors in the form the richest (maximum possible) model structure. We shall refer to it as prior model structure.

The user can express his believe to the prior model structure. He specifies for each model structure item (the pair of channel and delay) that the item is in the resulting model structure:

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

The believe is specified as an additional row of the model structure for individual outputs.

The input arguments are:

		defaults
`data`	signals	none
`des`	signal description	none
`str`	prior model structure	none
	or prior model structure with believe added	
`pri`	encoded prior knowledge	none

The output is the estimated model structure.

Examples

Example 1: SO processing based on the test system:

```
data   = testdata(500, 0.01, 111);     % signals
des    = 'oi';                         % signal description
pri    = cell4('gain',[ 1 2 1]);       % encoded prior knowledge
str    = rgrord(des , 3)       % prior model structure (3rd order)
str =
      1     1     1     2     2     2     2
      1     2     3     0     1     2     3
str    = eststr(data, des , str, pri) % model structure
str =
      1     1     2     2
      1     2     0     1
```

Example 2: MO processing, test system:

```
data1 = testdata(200,0.01,111);
data2 = testdata(200,0.01,222);
data  = [data1;data2];              % signals
des   = 'oioi';                     % signal description
str   = rgrord(des, 3);             % 3th order model structure
str   = cell4(str, str);            % prior model structure
pri   = cell4('g',[1 2 1], 'g',[3 4 1]);% encoded prior knowledge
str   = eststr(data, des, str);     % model structure
disp( getcell(str,1) );             % structure related to 1th output
      1     1     2     2
      1     2     0     1
disp( getcell(str,2) );             % structure related to 2nd output
      3     3     4     4
      1     2     0     1
```

6.17 peride

Purpose

The joint control period and regression vector structure estimation for "generalized" ARX models. Works with SISO systems only.

Synopsis

```
[conper, optstr] = peride(data, des, maxper, maxstr)
```

Description

The joint control period and regression vector structure estimation procedure searches for a model structure with the highest posterior probability in a space of competitive control-period-dependent model structures.

The space of the model structure competitors is specified by the user in the form of a maximal integer control period (CP) to sampling period (SP) ratio and the richest potential regression vector structure, the latter being a two row matrix of channel–delay pairs as specified above. An additional row expressing a user's belief on importance of the particular regressors can be concatenated with it: for each regression vector element it specifies that this element is

1	certainly present	2	probably present
3	probably not present	4	certainly not present

in the estimated model structure. If it is not specified at all, it is composed of all 2's by default. Also, the possibility of an offset in the estimated model structure can be specified by adding a column [0; 1] to the `maxstr` variable.

The input arguments to the function have the following meaning:

`data`	signals
`des`	description of signals
`maxper`	maximum prior control period
`maxstr`	maximum prior model structure

The output argument `conper` is the estimated CP/SP ratio, `optstr` is the estimated regression vector structure.

Example

The true model structure is an element of the set of competitive model structures.

```
data   = testdata(500, 0.01, 111);
des    = 'oi';
maxper = 3;                % maximum prior control period
maxstr = rgrord(des, 3) %  3rd order model structure
[conper, optstr] = peride(data, des, maxper, maxstr)
...
conper =
1 optstr =
      1     1     2     2
      1     2     0     1
```

6.18 prior

Purpose

Prior knowledge processing for ARX models.

Synopsis

`Pri = prior(des, str, pri, Ini)`

Description

Prior knowledge is transformed into prior model - underlying philosophy and processing algorithms are described in [3].

The input arguments specify:

		defaults
`des`	signal description	none
`str`	model structure	none
`pri`	encoded prior knowledge	none
`Ini`	initial model	default

The function returns the prior model `Pri`. We recommend to use this model as initial and/or alternative model in subsequent processing.

The initial model makes it possible to split the prior knowledge processing into several steps - output from one step is used as initial model for a subsequent step.

Encoded prior knowledge

The encoded prior knowledge is a cell vector

```
pri = cell4(t1, d1, t2, d2, ...)
```

where each pair `t*, d*` describes a prior knowledge item.

Each `t*` is a character string. It describes the type of the prior knowledge item (the first character is decisive with exception of 'D'). The matrix `d*` that follows contains a quantitative description of the prior knowledge item.

The function processes the following types of the prior knowledge:

type (`t*`)	meaning
'd'or 'D'	dynamic signals
's'	sparse dynamic signals
'm'	steady state signals
'g'	system static gain
'r'	signal ranges
'c'	cut–off frequency
'l'	parameter range
'a'	frequency characteristic
't'	dominant time constant
'f'	functional dependence
'w'	*confidence factor*

Note: the prior model structure and user's believe to it are another important prior knowledge types. They are discussed in the section "Model structure estimation" and they are not processed by the `prior` function.

The quantitative description (value) of a prior knowledge is a case specific matrix. For most of the prior knowledge types, it can be recorded as a single value or as "value ranges". The value ranges (see paragraph below) reflect uncertainty of the prior knowledge (confidence that the user have to the prior knowledge item). When the value ranges are not applicable for a prior knowledge type, it is explicitly mentioned in the following description.

By value ranges we mean vector [minimum value, maximum value] for a scalar value. If the value is a vector, it is replaced by two rows containing the minimum and maximum values.

Some types of prior knowledge relates to frequency. It is expressed as a number of cycles that fall in one sampling period selected. All values that are represented by time are encoded in the multiples of the sampling period.

We discuss how individual prior knowledge types are encoded:

<u>dynamic signals</u> are signals recorded with the sampling frequency selected.

We speak about historical data when the dynamic signals are not consistent with the signals used for controller design e.g. they are not sufficiently excited or they have been recorded in a not typical working point.

The historical data should be encoded with the `'d'` type, otherwise (if a part of signals used for control design) they should be encoded with `'D'` specification.

In the later case, the signals are processed by least square algorithm with a forgetting (if knowledge type is is endcoded as `'D1'`, exponential forgetting is used, otherwise generalised exponential forgetting is applied). The processing takes part after all other prior knowledge items are processed. The forgetting factors (or vector forgetting factor in a MO case) and the signals are compressed in a cell vector.

If generalised forgetting is applied, the alternative model (or cell vector of models in MO case) can be specified as the 3rd cell of the encoded prior knowledge item. If not specified, the prior model(s) resulting from the previous (not 'D' or 'd') prior knowledge processing is used as the alternative one.

sparse dynamic signals are signals in an ordered time sequence different from the regular sampling. The sampling time is recorded as an additional (the last) row added to the signals.

The signals can be generated e.g. by sampling plots of step or impulse responses (even manually from plots).

steady state signals are steady state system measurements recorded in a matrix.

system static gain is change of an output in steady state due to unit change of an input. It is specified as vector:

[output, input, gain]

signal ranges specify minimum (first row) and maximum signal values (second row) in the order determined by the model description.

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

[output, input, frequency]

parameter range is entered as:

[parameter index, minimum parameter value, maximum value]

By parameters we mean here the field `'Eth'` of the ARX model.

frequency characteristic means amplitude and shift of output response when a sinusoidal signal of the transformed frequency and amplitude 1 is applied to an input. It is specified as:

[output, input, frequency, amplitude, shift]

dominant time constant is the constant that describe an output response when unit step is applied to an input. It is specified as:

[output, input, time constant]

functional dependence (e.g. a conservation law) are expressed by a character string. The string is evaluated in processing. The character 'x' prefixes the channels - see below.

Let us explain the coding by examples. Let us know that the 2nd channel is sum of the 1st and 7th channels. The dependency is encoded as:

```
dependency = 'x2=x1+x7'
```

In more complex cases, the dependency can be specified by a MATLAB function, e.g. `fun`:

```
dependency = '[x3,x5]=fun(x1,x7)'
```

confidence factor makes it possible to express user's confidence to the knowledge source that follows. This possibility is for expert usage of the function only, normally, the confidence is derived from value ranges.

The confidence factor w expresses ratio of the guess of model noise variance $w1$ and the guess of the prior knowledge variance $w2$ ($w = \sqrt{w1}/\sqrt{w2}$).

Examples

Example 1 : we consider a SISO model of the 2nd order, the first channel is the output. The system static gain is 1, the cut-off frequency is 1000.

We make definitions and encode the prior knowledge:

```
des    = 'oi'; % signals description
str    = [ 1 1  2 2 2; 1 2  0 1 2 ]; % prior model structure
output = 1;     % auxiliary - output channel
input  = 2;     % auxiliary - input channel
gain = 1;       % system static gain
cutoff=1000;    % cut-off frequency
pri = cell4('gain', [ output, input, gain] ...
            'cuto', [ output, input, cutoff]);
```

We compute the prior model:

```
Model = prior(str, des, pri);
```

Example 2: consider a model with 4 channels, the channels 1 and 3 are outputs, the channels 2 and 4 inputs with the model structure as below:

```
des = 'oioi'; % signal description
str = cell4([ 1  2 2  3; 1  0 1  1 ],...  % model structure
              [ 3  4 4  1; 1  0 1  1 ]);
```

Our prior knowledge consists of two steady states system measurements:

```
m1 = [  1   1   1  1 ];
m2 = [ 10  -1  -2  0 ];
```

and cut-off frequencies 1000 for the 1st output due to the 1st input as well as the 2nd output due to the 2nd input:

```
c1 = [ 1, 2, 1000 ];
c2 = [ 3, 4, 1000 ];
```

Moreover, we have historical data `data` to be processed by least squares algorithm with the generalised exponential forgetting with forgetting factors:

```
frgs = [0.92, 0.92];
d = cell4(frgs, data);
```

We encode prior knowledge:

```
pri = cell4('m',[m1;m2], 'c',c1, 'c',c2, 'D', d);
```

The, the prior model is obtained and individual prior model accessed by:

```
Model = prior(des, str, pri);
Model1 = getcell(Model,1);
Model3 = getcell(Model,2);
```

6.19 requsat

Purpose

Closed loop performace prediction.

Synopsis

```
[perf,C,C1] = requsat(Model,cfinal,wish,pre)
```

Description

This function calculates the statistics of the basic signals determining the closed loop performance and thus predict the closed loop behaviour. The mean values and variances of the input and output are calculated for uncertain system represented by the mean value and covariance matrix of its parameters obtained by procesing available data and corresponding optimal controller designed by the function `cdesign`. The evaluation is based on Monte Carlo simulation.

The probability of satisfaction of user's requirements and wishes are evaluated as well. All results are grouped in an output structure `perf`. Optionally, the output variable `C` contains trajectory of disturbance compenzation for the system with parameters having the mean value of the distribution. The variable `C1` contains a step responce if user has specified overshoot.

The input arguments are :

matrix	meaning
`Model`	result of parameted estimation (kwestpar)
`cfinal`	result of control desing (`cdesign`)
`wish`	requirements related to input range
`pre`	preprocessing (cell list)

The output parameter `perf` is a structure containing the simulation results. The structure fields are:

fields	meaning
`meanu`	mean of the input
`sig2u`	$\sum(u-u0)^2$
`ulim`	relation in % of the samples of the input in user defined limits to the total number of samples
`uslim`	simmilar as `ulim` but for the case of a step responce
`dulim`	simmilar as `ulim` but for the input differences
`meany`	mean of the output
`sig2y`	$\sum(y-y0)^2$
`einf`	steady state error
`ylim`	simmilar as `ulim` but for output
`overshtt`	overshoot (max(y)/y(inf))

6.20 scaleit

Purpose

Scalling of special cell lists

Synopsis

```
cons = scales( cons, des, 'c', pre)
wish = scales( wish, des, 'y', pre)
```

Description

The input arguments are :

`cons`	user's constraints
`wish`	user's wish about output
`pre`	preprocessing

The function returns transformed cell lists.

Examples

zscaleit

6.21 testsys

Purpose

Test system - close loop data.

Synopsis

```
sys=testsys(sigma)
y=testsys(u, sys)
```

Description

The first form build the test system, the second one generates output in open loop. The test system corresponds to the one used in `testdata`.

The arguments are :

`sigma`	output std.
`sys`	states
`y,u`	output and input

6.22 versim

Purpose

Verification of the behaviour of pre-tuned controller in an experiment simulating real environment.

Synopsis

```
ver = versim(system,prefilt,Model,csetup,requ,frg,Alt)
```

Description

The aim is to provide a verification of the results obtained from the preceding steps of the design. The experiment is set up using the designed controller in the loop with the real or simulated system. The experiment uses SIMULINK environment. Standard SIMULINK features are used for system simulation, real time toolbox makes possible the connection with real controlled system.

The input arguments are :

matrix	meaning	default
`system`	SIMULINK block of real plant	System0
`prefilt`	information about pre-filtering	prefilt0
`Model`	model used for the design	no
`csetup`	results of a pretuning step	csetup0
`requ`	user design requirements	requ0
`frg`	exponential forgetting factor	1
`Alt`	alternative model	no

The `system` is a structure containing the information about the SIMULINK system simulation. It consists of the items:

matrix	meaning	type of variable	default
`integr`	SIMULINK integration method	string	'euler'
`tol`	the tolerance	number	.001
`maxstep`	largest allowed integration step	number	.1
`explen`	recommended experiment length	number	no
`plant`	name of a M-file with SIMULINK scheme	string	system0.m

The scheme must have one input and three outputs. The internal structure of the scheme is not restricted. The input is system input, the outputs of the scheme are reserved for system output, measurable disturbance and set point variable. If some of these variables are not considered in the experiment, they must be set to zero.

The output parameter `ver` is a structure consisting of simulation results and represents Control Specification Response. It consists of

matrix	meaning
`sig2e`	$\sum(y-y0)^2$
`einf`	steady state error
`sig2u`	$\sum(u-u0)^2$
`ylim`	relation in % of the samples of the output out of user defined limits of outputs to the total number of samples
`dylim`	simmilar, but for the output differences
`ulim`	simmilar, but for input
`dulim`	simmilar, but for the input differences
`oversht`	relation in % of max(y)/yinf

6.23 getlist

Purpose

Get cell list options.

Synopsis

```
val = getlist(list, channel, option)
val = getlist(list, option)
```

Description

The cell list specifies e.g. a requirement for signal preprocessing:

```
pre = cell4('channel', 2, 'scale', [ 1 5 ]);
```

This is encoded requirement that specifies that each channel 2 signal value will be preprocessed before used - 1 will be added and the result multiplied by 5.

The function `getlist` is designed to extract the information encoded in the list, e.g.:

```
getlist(pre, 2, 's')   % gets scale information for 2nd channel
ans =
     1     5
getlist(pre, 's')      % gets first cell item coded as 's...'
ans =
     1     5
```

Instead of pair of arguments, the list can contain one cell list, e.g.

```
pre = cell4(pre, cell4('c', 3, 's', [ 10  50 ]));
getlist(pre, 3, 's')
ans =
    10    50
```

Example: zgetlist

6.24 Cell vectors and structures

Purpose

Interface to the equivalent MATLAB 5 cell arrays and structures. The functions are supported under MATLAB 4 so that ABET97 is not dependent on the MATLAB version.

Synopsis

```
S = struct4('f1', m1, 'f2', m2, ...)
field = getfld(S,'fieldname')
S = setfld(S,'fieldname', value)
names = fldnames(S)
i = isstruc4(S)
C = cell4(m1, m2, ...)
field = getcell(C,i)
C = setcell(C,i,value)
i = iscell4(C)
n = size4(SC)
M = conv5(SC)
disp4(SC);
```

Note: `S` is a structure, `C` is a cell vector, `SC` is a structure or a cell vector, `field` is any matrix, `i` is a logical scalar, `names` is a character matrix `n` a number, M is MATLAB 5 cell array or structure.

Description

The functions described in this section are:

function	equivalent to MATLAB 5	meaning
struct4	struct	create structure
getfld	getfield	get structure field
setfld	setfield	set structure field
fldnames	fieldnames	get field names
isstruc4	isstruct	test if structure
cell4	C{i}=...	create cell vector
getcell	...= C{i}	get cell array member
setcell	C{i}=...	set cell array member
iscell4	iscell	test if cell array
disp4	disp	display
size4	size	number of fields/cells
conv5	—	conversion to MATLAB 5

ABET cell vectors

The cell vector is vector whose elements are matrices. The following commented diary explains use of relevant functions.

```
C =   cell4( 1, [1 2; 3 4]);      % create cell vector
disp4(C);                         % display
ABET cell vector with 2 cells
1) m1 :double   = 1
2) m2 :double  (2,2) = 1  3  2  4
C =   setcell(C, 1, [1 2 3]);     % set a cell
disp( getcell(C, 1)  );
1     2     3
disp( iscell4(C) );               % test if a cell vector
1
disp( size4(C) );                 % size
2
B =   setcell(C, 5, 123);         % extend cell vector
disp4(B);
ABET cell vector with 5 cells
1) m1 :double  (3) = 1  2  3
2) m2 :double  (2,2) = 1  3  2  4
3) m3 :char    (5) = dummy
4) m4 :char    (5) = dummy
5) m5 :double   = 123
%        cell  offset replacement
cell4set(C, 2,  2,     [22, 33]);   % fast set
disp( getcell(C, 2)  );
1    33
22    4
```

ABET structures

The structures consist of named fields. The following commented diary explains use of relevant functions.

```
A = struct4( 'b',1,'c','....');  %  create structure
disp4(A);                        %  display
ABET structure with 2 fields
1) b :double   = 1
2) c :char    (4) = ....
disp(  getfld  (A,'c')  );       %  get field
....
A = setfld  (A,'b',[1 2 3]);     %  set field
disp4(A);
ABET structure with 2 fields
1) b :double  (3) = 1  2  3
2) c :char    (4) = ....
A = setfld  (A,'new',123);       %  extend structure
disp4(A);
ABET structure with 3 fields
1) b :double  (3) = 1  2  3
2) c :char    (4) = ....
3) new :double   = 123
Afields = fldnames(A)            %  get field names
Afields =
b
c
new
disp( isstruc4(A) );             %  test if structure
1
B = struct4('A',A);
disp(  getfld(B,'A.b')  );       %  dot notation
1     2     3
```

Conversion to MATLAB 5 tools

The function conv5 converts ABET cell vectors and structures to equivalent MATLAB 5 matrices. Use is clear from the next diary:

```
S = struct4('a',1,'b',2);
C = cell4(1,[1 2],S);
M=conv5(C)
M =
     [1]    [1x2 double]    [1x1 struct]
A = struct4('mx', 1,  'str', S);
M = conv5(A)
M =
     mx: 1
    str: [1x1 struct]
getfield(M,'str')
 ans =
     a: 1
     b: 2
```

6.25 rgrord

Purpose

Create model structure using orders.

Synopsis

```
str = rgrord( des, orders, delays, offset)
```

Description

The function returns model desription.

The input arguments are :

des	model description
orders	orders for individual channels
delays	delays for inputs
offset	model offset

Instead of orders for individual channels, a scalar value can be used representing the common order.

Absence of a channel in the structure is expressed by the character '-' in the model description.

Example

```
des = 'oi';
rgrord(des,3)
ans =
      1     1     1     2     2     2     2
      1     2     3     0     1     2     3
```

7 *Low level support (auxiliary)*

The functions described here are part of a private design base (ABET) for the toolbox. They are not intended for an ordinary user.

7.1 simulation control

Purpose

Control of a simulation run.

Synopsis

```
simstart
simstart(nt)
simstart([t1,t2])
simstart([t1,i,t2])
```

```
simulate
resuts = simulate(pointers)
simconti(nstep)
```

```
resuts = simconti(nstep, pointers)
err = simstep(t)
simstop
```

Description

The function kwsimstart starts a simulation run. The length is expressed by argument as number of steps (`nt`), or as range (`t1:t2`) or as range with increment (`i`). If no length is specified, unlimited simulation run is started.

The function `simulate` make the simulation run. Use of the function without arguments leads to a fast simulation run (MEX processing only).

The `pointers` is a vector of pointers to any object members or zero - it leads to trajectory of signals.

The function returns trajectory of selected member or the cell vector of trajectories if more than one trajectory is requested. Row vectors are stored transposed.

The function `simconti` makes continuation of a simulation run. The number of steps is expressed as the 1st argument, the pointers can be specified as in the case of the function `simulate`.

The function `simstep` makes 1 step of simulation run with at the time specified.

The function `simstop` stops the simulation run.

Example

```
   datalen = 200;
   data = testdata(datalen);
   sigconst(simcon6(data));            % signal

   chnconst(2);                        % processing
R = rgrconst([1 1 2 2], [1 2 0 1]); % model structure
M = modcon1(R,1);                    % model
I = ideconst(M); chntask(I,1);       % identification

   Eth  = mbr2ptr(M,'Eth');      % pointer to parameters
Cth  = mbr2ptr(M,'Cth');      % pointer to covariance
ptrs = [0,Eth,Cth];           % 0 means signals

sysstart([1,datalen]);        % start simulation
C = simulate(ptrs);           % simulate, get results
sysstop;                      % finish

sig = getcell(C,1); sig = sig(1:2,:);       % signals
max(max(sig-data))
ans =
7.9635e-008
plot(sig');

Eth = getcell(C,2);                         % parameters
plot(Eth');

Cth = getcell(C,3);                         % Cth(1,1)
ans =
4   800
Cth1=Cth(1,1:4:800);
plot(Cth1);
```

7.2 eststr2

Purpose

ARX model structure estimation.

Synopsis

```
str = eststr2( data, des, str, pri)
```

Description

The function is a variant of `eststr` with the exception that prior model structure is expressed as one single global prior model structure.

The input arguments are:

`data`	signals	none
`des`	signal description	none
`str`	prior model structure	none
`pri`	encoded prior knowledge	not used

The output is the estimated model structure.

The function `eststr2` makes detailed model analysis that relates to control design aspects. Results are displayed as above by `strsumm`.

The function `eststr2` makes the following analysis. The signal description is reduced by omitting the inputs that do not affect any output and uncontrolled outputs that do not affect (even indirectly) any output.

The outputs that do not depend on any input are changed to uncontrolled outputs if other output depends on them, otherwise they are left out.

The model structure is analysed and isolated groups of channels are determined (those that depend only on the group members).

Three types of channel groups are distinguished. The groups contain outputs:

- with at least one controlled output (output group);
- uncontrolled outputs not affected by inputs (disturbance);
- uncontrolled outputs (auxiliary group).

Example

The prior model structure is expressed as a model structure of the fifth order by the auxiliary function `rgrord`:

```
data1= testdata(200,0.01,111);
data2= testdata(200,0.01,222);
data = [data1;data2];
des = 'oioi';
str  = rgrord(des, 5);
eststr2(data,des,str); strsumm;
Processing summary
------------------
... output group  ...
1 output depends on    1 output( 1, 2)    2 input ( 0, 1)
... output group  ...
3 output depends on    3 output( 1, 2)    4 input ( 0, 1)
--------------------
```

7.3 encaps

Purpose

Build encapsulated module.

Synopsis

```
encaps
enccaps('name')
```

Description

The encapsulated modules The ABET object `X` is converted into ABET97 structure. It can be converted to MATLAB 5 structure by `conv5` function.

Example

The example explains how ABET objects are converted to ABET97 structures. Let us solve a parameter estimation task:

```
data  = testdata(200, 0.01, 111);
des  = 'oi';
str   = [ 1 1 2 2; 1 2  0 1 ];
Model = estpar(data, des, str);
disp4(Model);
ABET structure with 7 fields
1) ychn :double   = 1
2) str  :double  (2,4) = 1  1  1  2  2  ...
3) covy :double   = 0.000109426
4) Eth :double  (4) = 1.84815  -0.859401  ...
5) Cth :double  (4,4) = 7.66672  -0.994089  ...
6) dfm :double   = 202
```

Let us find ABET objects:

```
I = objlast('ide');
M = objmbr(I,'Model');
R = objmbr(M,'str ');
```

Now, the objects are converted and displayed:

```
I5 = objcnv4(I); disp4(I5);
ABET structure with 6 fields
1) ychn :double   = 1
2) str  :double  (2,4) = 1  1  1  2  2  ...
3) covy :double   = 0.000109426
4) Eth :double  (4) = 1.84815  -0.859401  ...
5) Cth :double  (4,4) = 7.66672  -0.994089  ...
6) dfm :double   = 202

M5 = objcnv4(M); disp4(M5);
ABET structure with 6 fields
1) ychn :double   = 1
2) str  :double  (2,4) = 1  1  1  2  2  ...
3) covy :double   = 0.000109426
4) Eth :double  (4) = 1.84815  -0.859401  ...
5) Cth :double  (4,4) = 7.66672  -0.994089  ...
6) dfm :double   = 202
str= objcnv4(R)
str =
      1     1     2     2
      1     2     0     1
```

7.4 presetm

Purpose

Set preprocessing according to cell list.

Synopsis

`prep = presetm(pre)`

Description

The input arguments are :

`pre`	preprocessing
`prep`	preprocessing object

7.5 class reg

Purpose

Controller block.

Synopsis

`Reg=regconst(chany, uchn, mode,umax,udmax,chanw, chanu0)`

Description

Controller calculates system input variable (manipulated variable) according to a control law and available data.

```
u(t)  = r(1)u(t-1)+...+r(n)u(t-n) + s(1)y(t-1)+...s(n)y(t-n)
      = cl*x(t)
x(t)  = [u(t),u(t-1),...,u(t-n+1),y(t),...,y(t-n+1)]
```

The quadratic criterion is described by:
`J= min(u(t0+1),u(t0+T)) Esum(t0+1,t0+T) q(y)(y(t)-w(t))*2 +`
`+ q(u)(u(t)- u0(t))*2`

Two types of controllers are available :

type 1 : is a classical discrete PID controller.

type 2 represents a controller having a structure of a regression model. (or a discrete transfer function).

Constructor

Constructor `regconst` builds the controller object:

`Reg=regconst(chany, uchn, mode,umax,udmax,chanw, chanu0)`

The input arguments are:

- `chany` - output channel of the controlled system;
- `chanu` - input channel;
- `mode` - The variable enables to call either tupe 1 or type 2 controllers; in case of PID controller (mode 1) it is a vector containing parameters of the PID terms
- `chanw` - channel where is the system output reference;
- `umax` - saturation of the generated variable;
- `dumax` - limitation of the differences of the generated signal;
- `chanu0` - optional channel where is the reference for input;

Note: the structure and parameters of the regression type controller (mode 2) are contained from the object Modelu created by the modcon1 either by the user or automatically by the task Syn.

Examples: zsyn;

7.6 class syn

Purpose

LQ controller synthesis.

Description

LQ synthesis is based on a linear model and quadratic criterion. The approach used here uses as a model linear regression model of the form:

```
y(t)   = b(0)u(t)+...+b(n)u(t-n) + a(1)y(t-1)+...a(n)y(t-n)
       = parms*z(t)
z(t)   = [u(t),x(t-1)]
x(t)   = [u(t),u(t-1),...,u(t-n+1),y(t),...,y(t-n+1)]
```

The quadratic criterion is described by:
`J= min(u(t0+1),u(t0+T)) Esum(t0+1,t0+T) q(y)(y(t)-w(t))*2 +`
`+ q(u)(u(t)- u0(t))*2`

The optimization is based on the solution of the corresponding Riccati equation. Square root base iterations are used for the solution.
It generates an optimal control law in the form:

```
u(t) = -L x(t) + Lw w(t) +Lu0 u0(t)
```

Two types of LQ synthesis are available :

type 1 : is full SISO optimization for variable w(t) and u0(t) with multirate property, i.e. signals u(t) and y(t) can be sampled at different rates. Additionally to control law `L`, `Lw,Lu0` are provided in a matrix `clext`, which respect the penalization of the set point value and reference input in the criterion.

type 3 represents full MIMO optimization based on a set of MISO regression models. Each model can have different sampling period.

Constructor

Constructors synCon1, and synCon3 (synCon$x$, x=1,3) have the form

```
Syn=synCon1(chany, chanu, horizont, penal, nstep,mode,name)
```

The input arguments specify

- `chany` - output channel of the controlled system;
- `chanu` - input channel;
- `horizont` - synthesis horizont;
- `penal` - penalizations;
- `nstep` - number of iterations;
- `mode` -The variable enables to call the synthesis only once in the initialization stage (`mode` =1) or its call is controlled in the simulation experiment by a channel to which it is attached (mode=2).

Description

The synthesis looks for a model connected with an output channel `chany`. The model is created by the `modCon` constructor. The constructor checks if the parameter length corresponds to assumed structure. The structure of the controller is by default deduced from the system model or can be given in advance.

The `syncon1` constructor prepares the model of the controller `Modelu` and connect it with the input channel. The parameters of this model contains the control law coefficients. The structure of the controller has always the system input signal first, followed by the system output variables.

The criterion is defined by the vector `horizont` and `penal`. The first one contains the type of action (penalization of output(P), input(Q), input differences(D) minimization(M)) and the second one a numeric values of these action (a value for minimization is not used).

Type 3

The corresponding constructor is:

```
Syn=synCon3(Mudely,Modelu,horizont,channels,htime,penal,nstep,name)
```

where the input arguments specify:

- `Modely` - array of controlled system models;

- `Modelu` - array of controller models;
- `horizont` - synthesis horizont;
- `channels` - list of channels ;
- `htime` - list of time instants;
- `penal` - penalizations;
- `nstep` - number of iterations;

Description

The method minimizes the finite horizon criterion :

```
J=  min ESuma y(ty)Py y(ty) + Suma u(tu)Qu u(tu)
    u(tm)
```

where

E means expected value,

ty is a set of time instants where penalizations of output y are applied

tu is s set of time instants where penalizations of inputs u are applied

tm is s set of time instants where minimization are performed

The sets ty,tu,tm may be disjoint.

The method requires:

The arrays `Modely`, `Modelu` of system and controller models. Both list must be of the same length. The possition in the array determines the output – input pairs.

The criterion J above is introduced by matrices:

`horizon` is in the form of a character string. It says what actions represent the criterion. It consists of a sequence of penalizations of outputs (abbreviated by P or 1), inputs (abbreviated bu Q or 2) and minimization (abbr. as M or 3). Example: PQM is a valid sequence. Similarly (PM, QM, PPQQM, PPQQMM ...). The Horizont should contain at least one minimization (usually at the end).

`channels`. This integer vector of the lenght of Horizont indicates what channels are treated by Horizont. Example: [2 1 1] could correspond to the first example of `horizont`.

`htime`. This real vector of the length of `horizont` gives the time of variables in channels to be treated. The time means the future relative to the time when the minimization was activated. Example [1 1 1] is valid sequence for the simple example.

`penals`. This real vector contains values of penalizations for actions of the type P or Q. Places corresponding to M, are not used by the routine and can contain arbitrary value.

`nstep`. This integer value tells how many times the operations in `horizont` will repete. The values in `htime` are incremented by the value of max(`htime`)* `nstep`.

Examples

zsyn; demo;

written by J. Böhm, revised March 12, 1999

7.7 versim

Purpose

Verification of the behaviour of pre-tuned controller in an experiment simulating real environment.

Synopsis

```
ver = versim(system,prefilt,Model,csetup,requ,frg,Alt)
```

Description

The aim is to provide a verification of the results obtained from the preceding steps of the design. The experiment is set up using the designed controller in the loop with the real or simulated system. The experiment uses SIMULINK environment. Standard SIMULINK features are used for system simulation, real time toolbox makes possible the connection with real controlled system.

The input arguments are :

matrix	meaning	default
`system`	SIMULINK block of real plant	System0
`prefilt`	information about pre-filtering	prefilt0
`Model`	model used for the design	no
`csetup`	results of a pretuning step	csetup0
`requ`	user design requirements	requ0
`frg`	exponential forgetting factor	1
`Alt`	alternative model	no

The `system` is a structure containing the information about the SIMULINK system simulation. It consists of the items:

matrix	meaning	type of variable	default
`integr`	SIMULINK integration method	string	'euler'
`tol`	the tolerance	number	.001
`maxstep`	largest allowed integration step	number	.1
`explen`	recommended experiment length	number	no
`plant`	name of a M-file with SIMULINK scheme	string	system0.m

The scheme must have one input and three outputs. The internal structure of the scheme is not restricted. The input is system input, the outputs of the scheme are reserved for system output, measurable disturbance and set point variable. If some of these variables are not considered in the experiment, they must be set to zero.

The output parameter `ver` is a structure consisting of simulation results and represents Control Specification Response. It consists of

matrix	meaning
`sig2e`	$\sum(y-y0)^2$
`einf`	steady state error
`sig2u`	$\sum(u-u0)^2$
`ylim`	relation in % of the samples of the output out of user defined limits of outputs to the total number of samples
`dylim`	simmilar, but for the output differences
`ulim`	simmilar, but for input
`dulim`	simmilar, but for the input differences
`oversht`	relation in % of max(y)/yinf

7.8 estr,...

Purpose

Experimental ARX model structure estimation.

Synopsis

```
str = estr1(nchn,ychn,str)
str = estr2(nchn,ychn,str,V,n,V0,n0)
[V,n] = sta1 (data,ychn,str)
```

Description

The function `estr1` returns the prior structure used internally in the task of structure estimation.

The function `estr2` makes the structure estimation using data matrix `V` with number of degree of freedom `n`.

The function `sta1` makes the data matrix `V` with the number od degrees of freedom `n`

The input arguments are:

nchn	number of channels
ych	output channel
str	prior model structure
V,n	statistics
V0,n0	initial statistics

Examples: zestrrp1, zestr

7.9 etc...

Purpose

Low level usefull hints.

Description

zcrg	work with signal databases
zmap	mapping signal to different channels
zsim1a	efective simulation
zsimst1	stop simulation before final time is reached
zsimst2	" , using a task
zsim1	make effective simulation study
zgetrgr	get regressor in .m dynamic task
	generate signal from a dynamic task
	build dynamic tasks
zmergit	merge of two sorted arrays
zmapsel	select MAP estimate (from each column one)
zmod2obj	convert ABET97 model into ABET object

Index