ABET99

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 module that is employed in a real time control computer controlling the system.

Understanding of this report requires knowledge of basic MATLAB processing, and of structures and cell arrays introduced by MATLAB version 5. It also requires some basic control engineering knowledge, like the knowledge of concepts as model, SISO, MISO, MIMO.

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`
Parameter estimation	`estpar`
Merging prior information with model	`primerge`
Control design	`cdesign`
Control loop performance estimation	`requsat`
Forgetting estimation	`estfrg`
Controller code generation	`encreg`
Control verification	`versim`

The toolbox functions were created using specialized ABET functions [5] that are not discussed in this manual.

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 signals. The signal recorded on a channel is stored as the i-th row of the matrix; the number i is referred to as channel.

By channels description we refer to a character string expressing that corresponding channel is:

'o'	controlled output
'i'	input
'a'	auxiliary output
'r'	output reference channel

Example:

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

1.3 User's requirements

The user formulate his requirements for processing in a form processing options encoded in a cell vector referred to as cell list. Each option is represented by two arguments. The first one is a character string expressing an option type; the second one is a matrix that carries corresponding quantitative characteristics, i.e. option value. This is rather formal unifying tool.

Example of requirement for signal preprocessing:

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

1.4 Models

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. Only ARX model is used in control design.

The general model 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 parameterized 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.

In ABET we mean by model a structure with the specific fields (different 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

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;
degrees of freedom `dfm`	refers to number of data items processed
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 = cellvect(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.

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.

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 and covariance $cove$ ($\mathcal{N}(0, cove)$)

The model parameters are the model coefficients and $cove$.

The function `modarx` generates the default model, e.g.

```
ychn   = 1;                        % output channel
str    = [1 1  2 2; 1 2  0 1];     % model structure
Model  = modarx(ychn, str)         % model constructor
Model =
    ychn: 1
     str: [2x4 double]
       L: [5x5 double]
       D: [5x5 double]
     dfm: 1
```

The model is based on information matrix `V` that is a sufficient statistics for model parameters distribution. The matrix V is held decomposed into lower triangular factor L and diagonal factor D, V = L' * D * L.

Another form of ARX model is model with covariance matrix. It is used only for simulation. It can be build from the model with information matrix using `mod2c` conversion function:

```
Model = mod2c(Model)               % model conversion
Model =
    ychn: 1
     str: [2x4 double]
    cove: 1.0000e-006
     Eth: [0 0 0 0]
     Cth: [4x4 double]
     dfm: 1
```

The ARX model attributes are:

`cove`	noise covariance
`Eth`	mean of parameters distribution
`Cth`	covariance of parameters distruibution

1.5 Exponential forgetting

In identification, the estimated model attributes grow with number of data items processed so that soon reach 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).

Two types of forgetting techniques are implemented in ABET – exponential and generalized exponential [2].

The generalized exponential forgetting requires an alternative ARX model. The model attributes are forgotten by the forgetting factor and corrected by the alternative model.

A prior model can be used as the alternative ARX model with each signal item processed.

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 encapsulated controller created in the current working directory (name is specified by `cmodule`) that can be used in any C-encoded control program or as the MATLAB MEX-function.
3. the encaptulated adaptive controller created in MATLAB workspace; it is called by the function `enccall`.

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 meaning of the arguments is:

`data` specifies a name of a global matrix containing 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,10)
```

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

`cons` are constraints for controller design. The constraints are input ranges (u) and input increment ranges (du) as well as output range (y) an output increment range (dy):

```
cons   = cellvect('u', [-5  0  5], 'y', [-0.5  0  0.5], ...
                  'du', [-5  5],   'dy', [-1 1] );
```

This specification means that the controller is to be designed in such a way that the input range and input difference range fulfill the constrains specified.

The constraints on output does not influence the control design but, they are checked after the design is made.

Input and output reference value must be specified. The specification is the second number in corresponding ranges. The range itself is the 1st and 3rd value of the specification.

`pri` contains prior knowledge, e.g. the information about static gain between $< 0.99, 1.1 >$ is encoded as:

```
pri = cellvect('gain',[0.99 1.1])
```

`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.

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

2.1 Test ARX system

In tutorial examples, SISO simulated signals are used for an insight into processing. The test system is a discrete transformation of the continuous system with transfer function

$$F(s) = 1/(1+s)^2$$

The discretization is done with the sampling period 0.1 seconds and the resulting discrete system is of the 2nd order

$$F(z) = (0.004380.00468z)/(1 - 1.81 - 0.8187z^2)$$

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

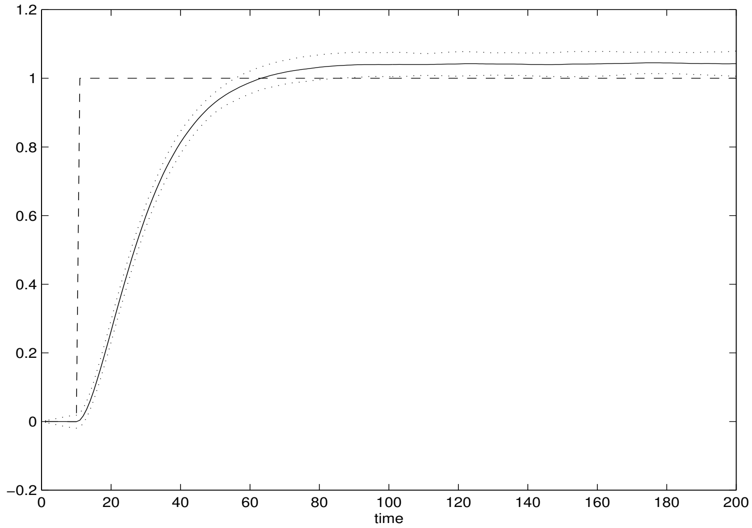
The function `testsys` is designed for close loop generation of the test signals by the same model as `testdata`. In the example that follow, the test system unit step response is analyzed. The trajectory of output (`Y`) is computed, averaged over 100 simulation runs and displayed together with standard 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);  % initialize 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. The signals are a global matrix:

```
global signals
ssig    = 0.01;                       % noise standard deviation
signals = testdata(300, ssig);        % signals
```

The design inputs are:

```
data   = 'signals';                  % name of signals
des    = 'oi';                       % channels description
maxstr = rgrord(des , 10);           % maximum prior model structure
maxper = 3;                          % maximum prior control period
pri    = cellvect('gain',[0.99  1.1]);  % prior knowledge list
cons   = cellvect('u', [-5  0  5], 'y', [-0.5  0  0.5], ...
                 'du', [-5  5],   'dy', [-1 1] );
cmodule = 'ctrl';  % name of C-encoded controller
```

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);
---------------
Project results
---------------
estimated control period: 1
estimated model structure:
     1     1     2     2     2
     1     2     0     1     9
estimated forgetting: 0.9999
controller penalization: input: 8.47e-004 input increment: 2.31e-004


-------------------------------------
Close loop performance evaluation
-------------------------------------
 90 percent ranges of :
inputs ...............    -2.697      2.617
  means                   -0.351      0.091
  standard deviations      1.112      1.879
input  increments .....   -3.885      3.253
  means                    0.002      0.008
  standard deviations      1.538      2.606
outputs ..............    -0.083      0.054
  means                   -0.044      0.015
  standard deviations      0.013      0.058
output increments .....   -0.023      0.025
-------------------------------------
```

2.3 Controller implementation in a real time 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=0;
// 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
}
```

2.3.1 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 = 0;                            % compute output
signal = [y u setpoint];                          % create signal
signal = ctrl(signal);                            % controller
u      = signal(2);                               % get input
```

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

```
% control loop
y = ...; setpoint = 0;                            % compute output
signal = [y u setpoint];                          % create signal
signal = enccall(signal);                         % controller
u      = signal(2);                               % get input
```

2.3.2 Example of implementation

Let us show how to use the controller created in the previous section. Our knowledge about the test system will be employed.

The diary from the previous subsection continues. The system is known:

```
uref = 0;                            % input reference
yref = 0;                            % output reference
str  = [1 1  2 2; 1 2  0 1];         % model structure
Eth  = [1.81 -0.8187  0.00468  0.00438 ];  % model parameters
```

The additional information about the simulation system:

```
ssig = 0.01;                         % noise standard deviation
msig = 0;                            % measurement standard deviation
tmax = 1000;                         % simulation length
randn('seed', 123);                  % simulation seed
```

The simulation system and initial values is created by:

```
Sys      = arxsys0(des,str,Eth,ssig,uref,yref,msig);
u        = uref;                         % initial input value
setpoint = 0;                            % output setpoint
Y = [];  U = [];                         % arrays for signal recording
```

The simulation is carried out :

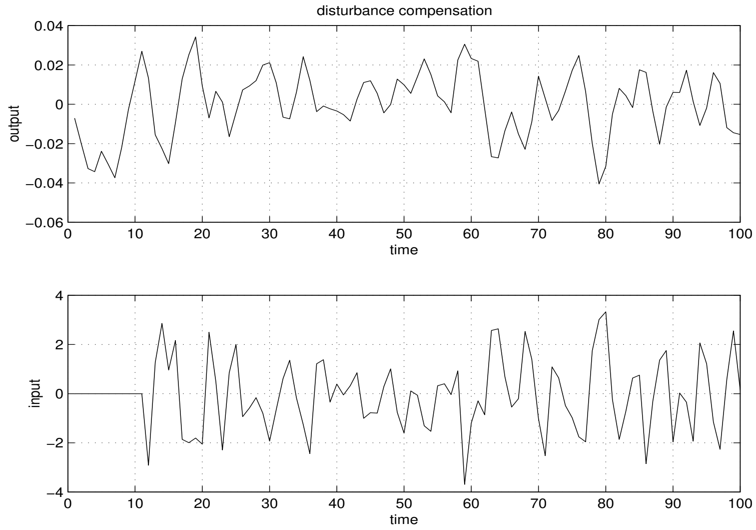
```
for t=1:tmax
      [y, Sys]    = arxsys(Sys, u);   % compute output
      signal      = [y u setpoint];   % interface to the module
      Y = [Y,y];   U = [U,u];         % record signal
      signal      = enccall(signal);  % controller
      u           = signal(2);        % get input value
end
```

Control quality is computed and diplayed:

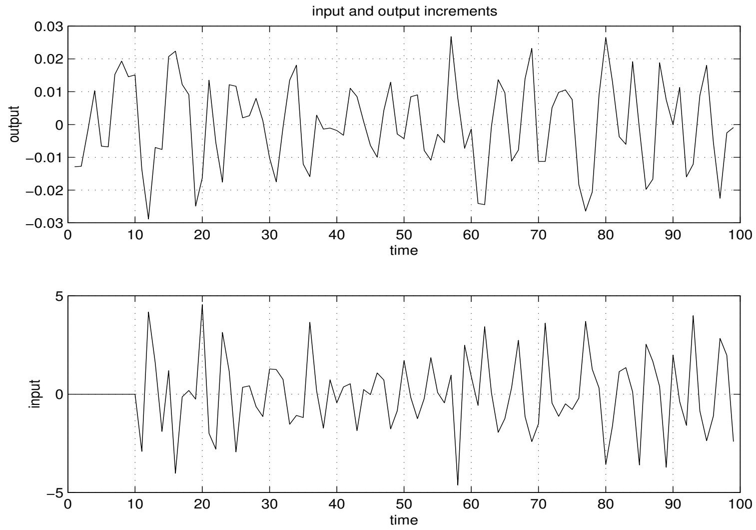
```
perfver = verres(Y-yref,U-uref);
dispperf(perfver,'verification results');
```

```
-------------------------------------
verification results
-------------------------------------
 90 percent ranges of :
inputs ................     -2.560      2.362
  means                     -0.133      0.005
  standard deviations        1.358      1.518
input  increments .....     -3.129      3.223
  means                     -0.012      0.010
  standard deviations        1.739      1.989
outputs ...............     -0.025      0.027
  means                     -0.001      0.002
  standard deviations        0.015      0.017
output increments .....     -0.021      0.021
  means                     -0.000      0.000
  standard deviations        0.012      0.013
-------------------------------------
```

The first 100 steps of the trajectories is displayed to show the disturbance compensation:

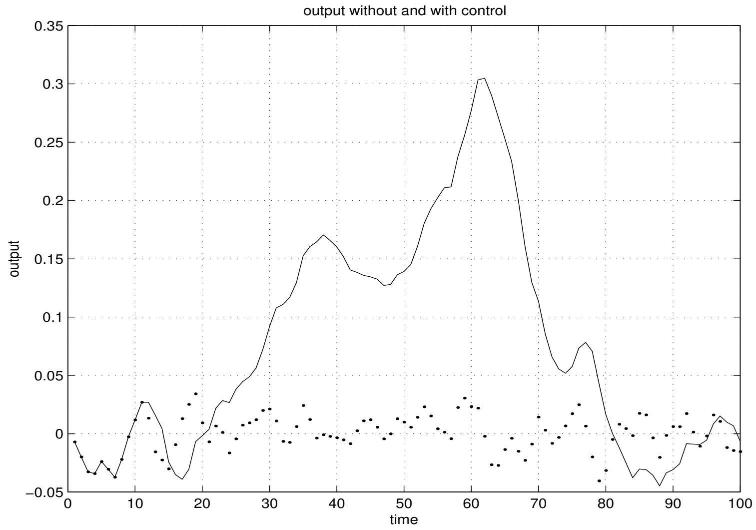


Disturbance compensation, inputs and outputs



Input and output increments

The last picture shows output without control compared with controlled output (dots).



Output controlled and not controlled

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. In control design we use normalization of input and output to their reference and to the 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 signal in the 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 normalization for channel 2 is encoded as:

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

Example 2: sampling of all channels is expressed as:

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

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

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

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

3.1.1 Scaling of requirements

In the controller design, the following transformations of signals is used:

```
[data, pre] = datahand(data, des, cons);
```

This transforms signal to the reference values (known from design constrains `con`) and to the standard deviation 1. The cell list of transformation done is returned (`pre`).

If control period `per` is not 1, the data are averaged:

```
if per>1
   holder     = cellvect('Hold', per);  % requirement to sample
   data_      = datahand(data_, holder); % sampled data
   pre        = cellvect(pre,holder);   % overall preprocessing
end
```

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 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 specified:

- 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 not surprising, the control period is one if the true model structure lies in the set of competitive model structures.

3.3 Prior knowledge in control design

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 (e.g. measured on a good simulation model) 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 $< 0.9, 1.1 >$. Then the encoded prior knowledge is the matrix `pri`:

```
pri = cellvect('measurements', stddata, 'gain', [0.9 1.1]);
```

The prior knowledge can be processed by the function `prior` that results in a prior model. The function needs the signal description `des` and the model structure `str`. The result of prior knowledge is a model referred to as <u>prior model</u>:

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

Currently, the prior knowledge in control design is merged directly with the estimated model to obtain the most authentic model.

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(200, 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    = cellvect('gain',[0.99 1.1]);       % prior knowledge
str    = eststr(data, des, str, pri) % structure estimation
str =
     1     1     2     2
     1     2     0     1
```

The true model structure has been estimated.

3.5 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. Example follows:

```
data  = testdata(200, 0.01, 111);       % signals
des   = 'oi';                           % channels description
str   = [ 1 1 2 2; 1 2  0 1 ];          % model structure
frg   = 0.98;                           % forgetting factor
Model = estpar(data, des , str, frg)    % parameters estimation
Model =
    ychn: 1
     str: [2x4 double]
       L: [5x5 double]
       D: [5x5 double]
     dfm: 48.1210
```

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 generalized exponential forgetting. There is a possibility to used a prior model for both arguments:

```
pri   = cellvect('gain', [0.99 1.1]);   % prior knowledge list
Pri   = prior(des, str, pri);           % prior model
Model = estpar(data, des , str, frg, Pri, Pri); % parameters estimation
M = mod2c(Model); M.Eth                 % model conversion
ans =
    1.8293   -0.8421    0.0043    0.0029
```

In control design, the parameter estimation is done without forgetting (if the signals are not extremely long):

```
frg   = 1;                              % forgetting factor
Model = estpar(data, des, str, frg);    % parameters estimation
```

Prior knowledge can be merge with the estimated model to obtain as authentic model as possible:

```
Model = primerge(Model,des,pri);        % merge model with prior knowledge
M = mod2c(Model); M.Eth                 % model conversion
ans =
    1.8501   -0.8614    0.0041    0.0036
```

3.6 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 `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 structure).

3.7 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 and their increments are calculated.

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`. The results will be shown in the summary example at the end of this chapter.

3.8 Forgetting estimation

The forgetting is a technique that allows to forget obsolete model attributes and creates 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 generalized exponential [2]. Each function that uses forgetting allows specification of the technique to be used.

The generalized exponential forgetting requires an alternative ARX model. The model attributes are forgotten by the forgetting factor and corrected by the alternative model. A prior model can be used as the alternative ARX model.

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 estimation
frg =
  0.9999
```

The result shows that no forgetting is required, the value shown is the highest used in control design. 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    = cellvect('gain',[ 1 2 0.99 1.1]); % prior knowledge
Pri   = prior(des, str, pri);     % prior model
```

The function `estfrg` allows to specify an initial (4th argument) and an alternative (5th argument) model. The prior model can be used as initial and alternative models:

```
frg  = estfrg(data, des , str, Pri, Pri)
frg =
     0.9999
```

The currently used estimation of forgetting uses the identified model `Model` as alternative model and the prior model as initial one:

```
frg = estfrg(data, des, str, Pri, Model);
```

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

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`.

3.10 Summary example

An example of overall adaptive controller design follows. To show all processing options, after the control period estimation, the control period is set to 2. The matrices suffixed by '_' means transformed entities, e.g. `data_` means transformed `data`.

The data are generated by the test system with noise standard deviation

```
ssig    = 0.01;                          % noise standard deviation
```

The design input arguments are:

```
data   = testdata(300, ssig,11);        % signals
des    = 'oi';                          % channels description
maxper = 3;                             % maximum prior control period
maxstr = rgrord(des , 10);              % maximum prior model structure
pri    = cellvect('gain',[0.99  1.1]);  % prior knowledge list
% design constraints
cons   = cellvect('u', [-5  0  5], 'y', [-0.5  0  0.5], ...
                 'du', [-5  5],   'dy', [-1 1] );
```

Control period selection and data preprocessing follow:

```
[data_, pre] = datahand(data, des, cons);  % data preprocessing

per = peride(data_,des,maxper,maxstr)  % control period
per =
     1
per = 2;
if per>1
holder     = cellvect('Hold', per);     % requirement for sampling
data_      = datahand(data_, holder);    % transformed signals
pre        = cellvect(pre,holder);      % preprocessing list
end
disp( size(data) );
     2   150
```

Prior knowledge is filtered and constrains are scaled :

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

Model structure is estimated:

```
str    = eststr(data_, des , maxstr, pri_) % model structure estimation
str =
     1     1     1     2     2
     1     2     3     0     1
```

Parameters are estimated and prior knowledge is merged with the estimated model:

```
Model = estpar(data_, des, str);        % parameters estimation
Model = primerge(Model,des,pri_);       % merge model with prior knowledge
```

Control is designed, the penalization are displayed:

```
cfinal = cdesign(Model, des, cons_);   % control design
dispreg(cfinal);
controller penalization: input: 2.15e-003 input increment: 1.00e-030
```

Control loop performance is evalueated:

```
perf = requsat(Model, des, cfinal, pre); % control loop performance
dispperf(perf,'controller verification');
-------------------------------------
controller verification
-------------------------------------
 90 percent ranges of :
inputs ...............     -3.133      2.971
  means                    -0.204      0.089
  standard deviations       1.390      2.661
input  increments .....    -5.333      5.258
  means                     0.008      0.033
  standard deviations       2.213      4.639
outputs ..............     -0.052      0.053
  means                    -0.023      0.011
  standard deviations       0.016      0.045
output increments .....    -0.047      0.047
-------------------------------------
```

Prior model is created and forgetting estimated:

```
Pri    = prior(des,str,pri_);
frg    = estfrg(data_, des, str, Pri, Model) % estimation of forgetting
frg =
    0.9981
```

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

```
encreg(Model,des,cfinal,frg,pre,'actr',Model);
```

Verification follows.

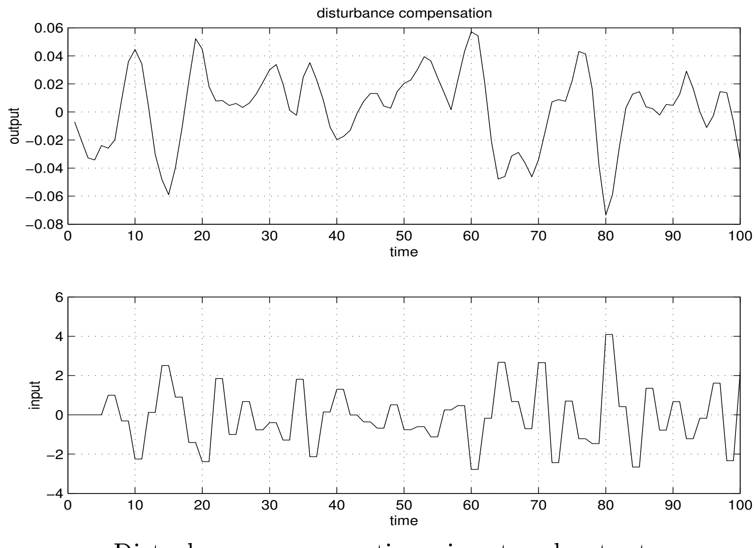
```
uref = 0;                                % input reference
yref = 0;                                % output reference
str  = [1 1  2 2; 1 2  0 1];             % model structure
Eth  = [1.81 -0.8187  0.00468  0.00438 ];   % model parameters


ssig = 0.01;                             % noise standard deviation
msig = 0;                                % measurement standard deviation
tmax = 1000;                             % simulation length
randn('seed', 123);                      % simulation seed


% simulation system
Sys      = arxsys0(des,str,Eth,ssig,uref,yref,msig);
u        = uref;                         % initial input value
setpoint = 0;                            % output setpoint
Y = [];  U = [];                         % arrays for signal recording
```

```
for t=1:tmax
    [y, Sys]   = arxsys(Sys, u);          % compute output
    signal     = [y u setpoint];          % interface to the module
    Y = [Y,y];   U = [U,u];               % record signal
    signal     = enccall(signal);         % controller
    u          = signal(2);               % get inut value
end
```

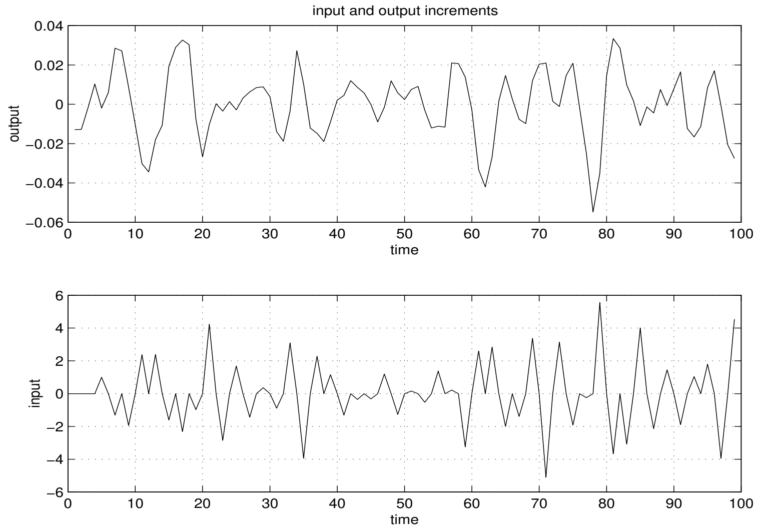
The verification results are displayed:

```
perfver = verres(Y-yref,U-uref);         % control quality evaluation
dispperf(perfver,'verification results');
-------------------------------------
verification results
-------------------------------------
 90 percent ranges of :
inputs ................     -2.579      2.512
  means                     -0.133      0.005
  standard deviations        1.338      1.603
input  increments .....     -3.182      3.381
  means                     -0.013      0.010
  standard deviations        1.414      1.846
outputs ...............     -0.046      0.053
  means                     -0.001      0.006
  standard deviations        0.026      0.030
output increments .....     -0.030      0.029
  means                     -0.000      0.000
  standard deviations        0.016      0.018
-------------------------------------
```

The first 100 steps of the trajectories is displayed to show the disturbance compensation. Notice the incremental mode of inputs:

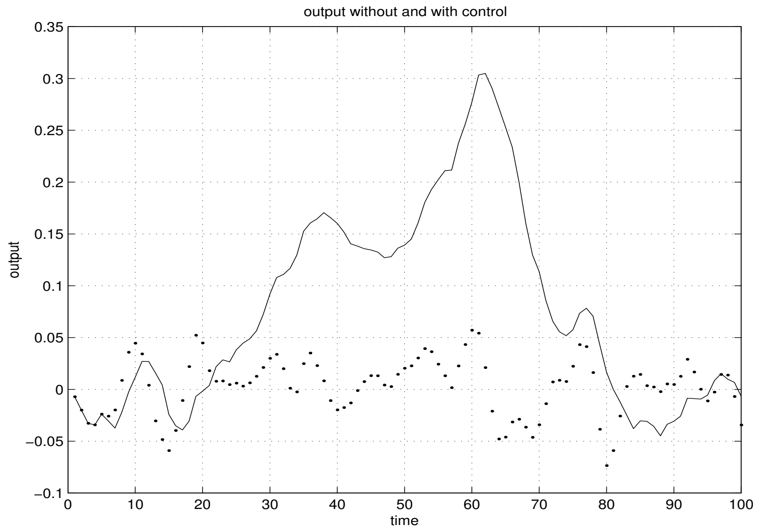


Disturbance compensation - input and output



Input and output increments

The last picture shows output without control compared with controlled output (dots).



Output controlled and not controlled

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 modeling 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 realized necessary blocks for verification. The scheme consists of:

- parameter estimation block
- controller synthesis (design) block
- controller block
- various blocks analyzing 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)
```

3.12 Experiments in control design

This section summarizes functions that are designed for experimentation in individual design steps.

The experimental features are designed for MISO cases. The relevant functions use the output channel `ychn` instead of channel description `des`.

3.12.1 Model structure estimation

The structure estimation is the most demanding step of a controller design.

A prior knowledge can influenced the model structure estimation; some experimentation can help.

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.

Current algorithm of searching in the space of competitive regressors finds optimum based on 100 searches with random start. The "best regressors" found can be displayed by setting the matrix `dbg=1` at the beginning of the function `eststr`.

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 function `estrglob` 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 and displayed.

3.12.2 Forgetting estimation

The estimation of forgetting is another crucial task solved in control desing. It is based on multiple model identification.

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).

Multiple model estimation is solved by the function `estmul`. It offers the possibility to observe whole process of multiple ARX model identification.

3.12.3 Control verification

The encapsulated modules makes it possible to change:
- controller penalization
- identification forgetting

The functions `encchng` makes the change in the encapsulated controller in workspace, while `encchngc` can make the change in encapsulated MEX module.

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

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

[5] P. Nedoma and M. Kárný and J. Böhm, "Decission making with ABET", Tech. Rep. No. 1964, ÚTIA AV ČR, Praha, 1999

5 ABET99 functions

5.1 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
`Model`	identified model	structure
`Ini`	initial model	structure
`Pri`	prior 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)
Model   = estpar  (data,   des,   str,    frg,  Ini, Alt)
Model   = primerge(Model,  des,   cons)
cfinal  = cdesign (Model,  des,   cons)
perf    = requsat (Model,  des,   cfinal, pre)
Pri     = prior   (des,    str,   pri,    Ini)
frg     = estfrg  (data,   des,   str,    Ini,  Alt)

[des, str, ref] = encreg  (Model,  des,   cfinal, frg,  pre, cmodule,
Pri)
Reg     = encreg  (Model,  des,   cfinal, frg,  pre, cmodule, Pri)
encreg  (Reg,    cmodule)

data    = datahand(data,   pre)
cons    = scaleit (cons,   'u',  pre)
pri     = scaleit (cons,   'p',  pri)
perf    = verres(input, output);
```

5.2 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
-------------------------------------
```

5.3 arxsys, arxsys0

Purpose

ARX system.

Synopsis

```
Sys = arxsys0(des,str,Eth,ssig,uref,yref,msig)
[y, Sys]=arxsys(Sys,u)
```

Description

The function `arxsys0` builds the ARX system.

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

The arguments of `arxsys0` are:

`des`	channels description
`str`	model structure
`ssig`	standard deviation of the model noise
`uref`	output reference
`yref`	input reference
`msig`	standard deviation of measurement noise

5.4 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 [4]. 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 and reference
`'du'`	input increment range
`'y'`	output range and reference
`'dy'`	output increment range

Input and output reference value must be specified. The specification is the second number in corresponding ranges. The range itself is the 1st and 3rd value of the specification.

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   = cellvect('u', [-5  0  5], 'y', [-0.5  0  0.5], ...
                  'du', [-5  5],    'dy', [-1 1] );
cfinal = cdesign(Model, des, cons);
```

This specification means that the controller is to be designed in such a way that the input range and input difference range fulfill the constrains specified.

5.5 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);
```

5.6 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

5.7 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

```
[des, str, ref] = encreg(Model, des, cfinal, frg, pre, cmodule, Pri)
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.

The `encreg` can return model description, model structure and reference values `ref` (output - input). This usage was designed for controller verification.

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
```

5.8 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  = cellvect('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  = cellvect(str1,str3);     %  model structure
pri  = cellvect('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
```

5.9 estmul

Purpose

Parameter estimation, multiple ARX model.

Synopsis

```
[Model, pmod, 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 likelihoods are transformed into partial models probabilities `pmod`.

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
`pmod`	trajectory of partial models probabilities
`parms`	cell vector of recursive parameters of partial models
`ep`	prediction error of individual models

5.10 MISO estimation functions: eststr1, estfrg1, estpar1

Purpose

Estimation functions, ARX model.

Synopsis

```
str                 = eststr1( data, ychn, str, frg, pri)
[frg, pmod, Models]    = estfrg1( data, ychn, str, frgs, frgu, Ini, Alt)
[Model, parms, ep] = 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`	forgetting estimation
`Models`	cell vector of partial models
`pmod`	trajectory of model probabilities
`Model`	identified model
`parms`	trajectory of parameter estimates
`ep`	trajectory of prediction error

5.11 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   = cellvect('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    = cellvect(str1,str3);        % model structure
pri    = cellvect('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
```

5.12 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

5.13 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

5.14 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

5.15 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    = cellvect('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   = cellvect(str, str);       % prior model structure
pri   = cellvect('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
```

5.16 modarx, mod2v, mod2c

Purpose

Create ARX model using defaults.

Synopsis

```
Model = modarx(ychn, str)
Model = mod2v(Model)
Model = mod2c(Model)
```

Description

The function `modarx` returns ARX default model. The functions `mod2c` and `mod2c` make model conversion between models with information matrix and models with covariance matrix.

The input arguments are :

`ychn`	output channel
`str`	model structure
`Model`	model

Example

```
    ychn  = 1;                                % output channel
    str   = [1 1  2 2;  1 2  0 1];           % model structure
    Model = modarx(ychn, str) % model
Model =
    ychn: 1
     str: [2x4 double]
       L: [5x5 double]
       D: [5x5 double]
     dfm: 1
    Model = mod2c(Model)                      % ARX model with covariance matrix
Model =
    ychn: 1
     str: [2x4 double]
    cove: 1.0000e-006
     Eth: [0 0 0 0]
     Cth: [4x4 double]
     dfm: 1
    Model = mod2v(Model)                      % model conversion
Model =
    ychn: 1
     str: [2x4 double]
       L: [5x5 double]
       D: [5x5 double]
     dfm: 1
```

5.17 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
```

5.18 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 [?].

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 = cellvect(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:

dynamic signals 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 = cellvect('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 = cellvect([ 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 = cellvect(frgs, data);
```

We encode prior knowledge:

```
pri = cellvect('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);
```

5.19 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
`oversht`	overshoot (max(y)/y(inf))

5.20 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

5.21 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

5.22 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

5.23 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

5.24 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
```

Index