Akademie věd České republiky
Ústav teorie informace a automatizace

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

RESEARCH REPORT

Ludvík Tesař, Miroslav Novák

Support Environment for System Identification and Controller Design - Jobcontrol

User's Guide with Examples

No. 2138 November 21, 2005

ÚTIA AV ČR, P.O.Box 18, 182 08 Prague, Czech Republic
Tel: (+420)266052422, Fax: (+420)286890378, Url: http://www.utia.cas.cz,
E-mail: utia@utia.cas.cz

Contents

Chapter 1

Introduction

Jobcontrol is a user friendly interface for Mixtools and Designer toolboxes. The Mixtools toolbox is a powerful set of utilities for system identification employing Gaussian mixture model. It is implemented as set of M-scripts and MEX-binary executables for the Matlab computing environment. It suits to the goal of finding suitable structure for given data. The Designer toolbox then serves for finding optimal controller parameters, constructing ideal controller and testing the controller found.

As an expert tools, Mixtools and Designer fulfills end user's needs, but are not suited for direct usage of the end user. In other words, they are not very user-friendly. It is why, we are developing environment, which integrates all the tasks, that are connected with system identification and controller design and helps to collect all the user's knowledge of data and the real-world system where data come from. After all calculations are finished Jobcontrol creates LaTeX-based report that integrates all the result together with the original user's settings. The Jobcontrol package, therefore, integrates endless expertise that is otherwise available only through study of the theoretical books [1, 3], Mixtools toolbox documentation [2] and experience contained in many experiments.

The Jobcontrol package help to solve user's problem in terms of the experiment (or job). Every experiment consists of description of user's data, and description of the way how the mixture is estimated and how the control is performed and what tests are to be done. Jobcontrol offers the user environment for interactive input of the description of experiment as well as lucid way of configuring experiment using one configuration file. Integral part of Jobcontrol package is the protocol generator, which automatically creates a very convenient LaTeX document, which shows all the aspects of system identification, control and user's data description.

To conclude, the Jobcontrol package performs these tasks:

- Takes user's data and estimates parameters of the mixture model.
- Verifies the Mixture model.
- Lets user to denote controlled and controlling parameters.
- Constructs the controller.
- Verifies the controller.

Chapter 2

Jobcontrol Description

To make things more systematic, Jobcontrol is divided into several steps.

- Data pre-processing
- Prior information utilization
- Initialization of the mixture model (mixinit)
- Mixture model parameter estimation (mixest)
- Mixture model validation
- User ideal calculation
- Controller design
- Controller verification

Both user's setup and processing follows the above-mentioned steps.

There are two ways to use Jobcontrol, the first is interactive (`jobmain` script) and the second is the non-interactive (or batch) usage.

Interactive is the script `jobmain.m`. It aims at collecting all the necessary information from the user by asking him for this information and then it proceeds with the the system identification, and controller design based on user's input. It colludes by creating protocol in LaTeX.

For non-interactive usage, user needs to understand at least basics of Matlab, but the non-interactive usage rewards user by many more possibilities, especially in batch processing. It allows user to use parts of jobcontrol in his scripts, jobcontrol even generates such basic script for him. Section 2.1 describes the first way to use jobcontrol by the `jobmain` script, section 2.2 describes the other non-interactive way.

We strongly recommend to read following sections at least to be able to judge where to use interactive or non-interactive jobcontrol.

2.1 Usage of interactive script `jobmain.m`

The script `jobmain.m` aims at easy and straightforward interactive usage and is ideal for the first-time user, even if he/she aims at using batch processing later. The interactive interface is launched by script `jobmain`. It guides user by printing instructions, so there is no need to describe all the steps in this text.

Important points are following:

- user has to decide, whether he wants to start the new experiment or to continue (load) the old one
- user has to provide data, that are relevant to his experiment, at least to give a clue for the program, what are the channels and what values can the channels have

- even if started from scratch, initial mixture can still be loaded from somewhere else. The concept of initial mixture is important, and user needs to understand what is he doing
- the problem setup is meant as: answer by pressing “Enter”, if you don’t know anything better. Some items are absolutely only for experts, and need not to be understood by average user.
- see section **??** to understand what steps are done during processing.

The `jobmain.m` script has a two-step concept: in the firsts one, the setup of the problem is done, which needs strong user interaction. After that, processing starts and no user interaction whatsoever is required or even desirable. Hence, importance of the question “Enter the letter that decides what will happen next” after the setup is completed. User must answer “c” which is labeled as *“c - start the evaluation”* in order to get any results at all. Otherwise he is stuck in setting up the problem again and again.

The processing step consists of several (seven) steps. After each step, the complete status of the experiment is saved, so that it can be started again from the same step.

Drawback of the script `jobmain` is that it asks user for every detail of the problem, and it is going through every step of the setup. If user makes any mistake, it is possible to change it but only after repeating the setup once more and after going through the setup once more. This may lead to aggressive behavior for some user, therefore authors strongly recommend to switch to non-interactive processing as soon as possible. While for first-time user, asking for every detail might be user-friendly, for only slightly more experienced user it is very boring process and in our opinion, excessive usage of `jobmain` may challenge user’s mental health.

2.2 Using the jobcontrol batch processing

To utilize the batch interface means to create own m-file where particular job parameters are set by hand with help of prepared assisting functions. The template of the batch file can be obtained by running `jobmain` once. It is automatically saved under the name of the experiment with the suffix `.m` (e.g. `exp1.m`).

The main feature of jobcontrol package is the `Job` structure. All information concerning task processed is in this structure in a straightforward way. This structure serves as centralizing point of the jobcontrol package. All parts of jobcontrol as well as the script `jobmain` described in 2.1, works entirely with this structure and uses it to store all the information necessary. It is constructed by the constructor jobconst, where it gets filled with default values. The lengthly process of setting up the experiment is performed by function `jobsetup`, which has the structure `Job` as both input and output, effectively it is updating it. After that, function `jobproceed` does all the processing both in interactive and you can use it also if you use jobcontrol in non-interactive way.

The template setup m-file created by `jobmain` can be used as starting point for your own setup. This m-file has the form of function, returning `Job` structure prepared for processing. For example if you name your setup m-file like `exp1.m`, than the better way to use it would be by invoking these commands:

```
prodini;
Job = exp1;   % function exp1 is called and has no parameters
Job = jobproceed(Job);
```

The Jobcontrol settings are stored in the structure `Job`. For reference on contents of the `Job` structure, you can type `help jobcontrol` at Matlab command prompt or you can use even better description in section 3.

2.3 How to run mixtools/designer repeatedly from the given point?

The whole concept of `jobcontrol` package aims to ease the batch processing. Status of all calculation is saved after every step into the mat file of the form: `exp1_`n`.mat`, where `exp1` is experiment name and number n is step after which it was saved. It can be loaded by:

```
Job = jobload("exp1_n");      % loads experiment exp1 from exp1_n.mat file
```

Or alternatively you can use equivalent way: `Job = jobload1("exp1",n)`. You can directly pipe this into jobproceed by:

```
Job = jobproceed(Job);
```

Before processing, you can run through the interactive setup and change something here and there:

```
Job = jobsetup(Job);
Job = jobproceed(Job);
```

Also, an attractive way to change part of setup is to use m-file generated by `jobmain` and put `Job = jobload("exp1_n");` into appropriate part of the m-file. (You should delete parts which correspond to steps already processed).

2.4 Using saved m-files

The structure of saved m-files is following. The whole file is cut into the sections, that correspond to sections of the interactive setup.

This is an example of one section of saved m-files:

```
begin_init;
SingleOnly  = 0;        % boolean flag whether to skip mixinit and calculate
   just MixSingle
opt     = 'p';          % iterative estimation method (p | q | b | f | Q | P)
niter   = 10;           % maximum number of iterations
frg     = 0.999999;     % forgetting factor
end_init;
```

You can see that this section starts with `begin_init` and ends by `end_init`. It is similar as all other sections.

2.5 Where to find results?

There are two important moment in Mixtools processing where results are shown and especially carefully prepared for user. It is step 5, validation step where results of mixture model estimation and validation are created. Also, it is step 8, where controller verification is done, therefore much of interesting information is given to user.

2.5.1 Mixture Model Validation Results

Results of mixture-model validation are created after step 5 and are saved into the file with `.tex` suffix and with the some prefix as user have given to the experiment (so, by default it is `exp1.tex`). Results can also be found in `Job.Val` sub-structure of the `Job` structure. Also results of validation are printed as text to the console, for user's convenience. During validation, various graphs are made.

During the setup, you can greatly influence the information which is plotted. Variable `plots` from the validation section or `Job.Valid.plots` controls which types graphs are made. Variable `pchns` from the same section or `Job.Valid.pchns`, controls what channels are validated. This, also influences, for what variables, graphs are created. All details can be seen below.

2.5.2 Controller Verification Results

Results of controller design using Designer method are resulting in tuning knob values, predicted ranges of signals, which are written into the file with `.tex` suffix. The resulting tex file is very comprehensive and even some graphs in form of eps files are included. For expert usage, `Job.DataDesc.Chns`, has the same information. The graph generated are sample of predicted closed loop behavior and predicted histogram of signals in the loop.

In `Job.DataDesc.Chns` variable, results of controller verification are in the same form as the result of controller design, except the word "predicted" is replaced by the word "verified".

2.6 How to start use jobcontrol?

If you have access to the local ÚTIA network, you can start to experiment with our Jobcontrol package straight away. The necessary steps follow.

- You will need Matlab and svn client software.
- Perform "svn checkout" of the Mixtools directory svn://marabu.utia.cas.cz:1800/svn/mixtools, user/password is guest/guest. You have read-only access to files in the repository.
- In Windows copy dlls from dll subdirectory to Mixtools directory.
- Run Matlab (version 6.5.1 is tested to work with current dll-files).
- Add path to the Mixtools directory into Matlab-path.
- In Matlab go to mixtools/jobcontrol/examples/simple subdirectory.
- Running this script: `run˙siso1t`.
- If some graphs were shown and the script ended normally, then your system is prepared to do your own experiments.

User will certainly be interested to experiment with their own data. Instructions for experimenting with user's data follows.

- Make your data ready for use in Matlab (save them as mat file).
- Run jobmain to prepare your experiment (myexp), you will be asked for the experiment name (let's say that you answered (myexp), and for the file name of mat file with your own data.
- Few files were created as side-effect:
  - m-file with editable experiment setup (myexp.m).
  - Saved status after every step of experiment: myexp_n.mat.
  - Protocol (myexp.tex). If your LaTeX is correctly installed postscript file is ready as well myexp.ps.
- Experiment can be repeated from any intermediate step (using myexp_n.mat files).
- Parameters of experiment can be adjusted (using myexp.m) and run again.

Chapter 3

Reference - Description of the structure `Job`

The pivot point of the Jobcontrol environment is the structure, which holds the information about the status of the current experiment undertaken. Normallly user does not need to access this structure directly, however, if some special task needs to be performed, then it might be useful to have descriotion of this structure for reference. The name of this structure is `Job` and its description follows.

Job.Main ... main information

- **jobname** ... name of the experiment (a very short string without spaces) - will be used to make some filenames
- **authorname** ... author's name for documentation purposes
- **email** ... author's e-mail address
- **references** ... references of the experiment to literature
- **project** ... name of the project
- **consttime** ... time of creating the setup in Matlab format, use datestr(Job.Main.consttime).
- **debug** ... debug level to be used
- **desc** ... a short description given by user (string)
- **seed** ... random seed of the experiment (empty=leave the seed from calling process, -1=randomize by timer)
- **doinit** ... used internally to signal whether certain initialization should be done (do not change!)
- **steps** ... which steps are to be done
- **finished** ... which steps were already finished

Job.DataDesc ... description of data:

- **datafile** ... data file filename
- **varname** ... variable name
- **transpose** ... if 1 than transposition should be done
- **rescale_data** ... rescale new data (normaly this is necesary every time the new data is given, but it does no harm to do it every time)
- **reset_chns** ... reset channels description (normally this is needed only once, in the beginning, and if new data have vary different channels than the old one)
- **pr_chns** ... channels to be printed in protocol
- **pr_merge** ... whether printed channels are to be merged
- **used_data** ... this indicates how much of data user wants to use (-1 means all, number means maximal index)

- **reset_val_min_max** ... this is internal variable that ensures that validation min and max is reset only once
- **new_pre** ... internal scaling parameter to be used with actual data described here (it is reset in order to avoid doing it repeatedly)
- **ndat** ... internal variable holding number of data vectors
- **Chns** ... channel description structure vector (every vector item corrsponds to one channel). description of the structure follos:
  - **chn_name** ... short name of the channel
  - **chn_oitem** ... visibility by operator
  - **chn_raction** ... available for control (input has 1 here)
  - **chn_prty** ... presentation priority
  - **chn_type** ... 1=continuous, 0=discrete
  - **chn_prange** ... range used for scaling [min,max]
  - **chn_drange** ... desired range used for control [min,max]
  - **chn_irange** ... desired range used for control - increments
  - **chn_preinfo** ... preprocessing information
  - **chn_gain** ... static gain [uchannel, min, max]
  - **chn_stime** ... sampling time (unit=seconds), needed for frequency-based priors
  - **chn_ampl** ... frequency response [input_channel, frequency, ampl_low, ampl_high, phase_in_degrees]
  - **chn_cut** ... cut-off frequency [input_channel, frequency]
  - **chn_tc** ... time constant [input_channel, low, high]

Job.Prior ... prior information

- **doflattening** ... forces flattening
- **Mix** ... prior mixture generated by genmixe
- **MixSingle** ... prior single-component mixture generated by genmixe

Job.IniMix ... initial mixture:

- **ncom** ... number of components
- **ord** ... order
- **diaCth** ... diagonal of Cth
- **diacove** ... diagonal of cov(Eth)
- **mult_dfm** ... multiplicator of degrees of freedom
- **mult_dfcs** ... multiplicator of degrees of freedom of components
- **dfm** ... degrees of freedom
- **dfcs** ... degrees of freedom of components
- **Mix0** ... mixture generated by genmixe
- **Mix0Single** ... single-component mixture generated by genmixe

Job.Init ... initialization of the mixture:

- **SingleOnly** ... boolean flag whether to skip mixinit and calculate just MixSingle
- **opt** ... options for mixinit - method,etc.
- **niter** ... number of iterations
- **frg** ... forgetting
- **Mix** ... result: identified mixture
- **MixSingle** ... result: identified single-component mixture

time ... time needed for mixinit

Job.Estim ... mixture estimation parameters:

opt ... options - method,etc.
niter ... number of iterations
frg ... forgetting
Mix ... result: estimated mixture
frgSingle ... single-component forgetting
MixSingle ... result: estimated single-component mixture
time ... time needed for mixest

Job.Batch ... mixture initialization/estimation made by batch processing:

do_batch ... 1 if batch processing should be done, 0 otherwise
batchlen ... length of batch

Job.Valid ... validation of the mixture:

nsteps ... number of steps for prediction
pchns ... predicted channels
cchns ... channels in condition
tstart ... starting time for validation
tend ... ending time for validation
epss ... print epss
pauses ... number of seconds to pause
plots ... which plots to make (this is vector with four 0/1)
segments ... number of segments fo segmentation-type validation
alt ... preform validation with alternative forgetting

Job.Val ... results of mixture validation tests:

mixll ... mixture log-likelihood
testSE ... ratio of average square of ep and sds of data
sumCep ... whiteness test result (sum of 10 delayed correlations of ep)
ChnStat ... vector structure of channel statistics (min, max, mean, median, std)
DiffStat ... vector structure of time differences statistics (min, max, mean, median, std)
YpStat ... vector structure of prediction statistics (min, max, mean, median, std)
EpStat ... vector structure of error prediction statistics (min, max, mean, median, std)
testNois ... noise standard deviations
dfcs ... dfcs

Job.UserIdeal ... User ideal description

method ... method to create user ideal (d - Designer toolbox, t - Target)
userSetpointEths ... setpoint Eths
userSetpointCoves ... setpoint Coves

Job.Design ... controller design parameters

typloss ... type of evaluating the constraints violation, possible values:

'prob' ... evaluating maximum overshoot

'max' ... evaluating probability of constrains violation

constrtol ... when constraints are evaluated as probability (`typloss = 'prob'`) this value contains maximum allowed probability of constraints violation

simlength ... simulation length of the evaluation algorithm, it should be long enough to contain all the reference changes

adaptive ... type of adaptivity, possible values:

0 controller is non-adaptive. The calculation is fast in this situation and so it is useful for first experiments with new system.

1 controller is adaptive with exponential forgetting.

2 controller is adaptive with alternative forgetting. Alternative forgetting is more stable than the exponential one.

horizon ... LQG horizon is specified by tuple. The first element is the horizon length at the start of simulation and the second one is the horizon in following steps. If it is negative, the previously calculated Riccatti matrix is used, so small number can be used such as one. This improves the simulation speed.

initialData ... initial data for ARX model It has same structure as DATA matrix, but it is short. The length must be at least as long as the `ord` variable to initialize ARX model. Empty matrix defaults to all zero initial conditions.

penal ... initial and final penalization (tuning knob) vector. Possible values are.

-1 automatically guess using the FPD design. This choice is equivalent to empty matrix.

-2 automatically guess using the Riccatti matrix

positive value all elements of penalization vector will be initialized to this value

initial penalization vector specified directly

penalY ... fixed penalization of outputs vector. If empty matrix is specified, all outputs channels has same wight equal to one. This is a default behavior. If a vector of length equal to number of output channels is assigned, the outputs will have different relative variances equal to inverse of respective elements of `penalY`. The rate holds for the scaled signals.

designtype ... procedure to create a controller

't' "target", non-adaptive, non-tuned controller, for mixtures

'd' "designer", adaptive, tuned controller, single component models

aMix ... controller mixture, result of "target" procedure

CtlT ... controller object, result of "designer" procedure

Job.Verify ... controller verification parameters

verifyopt ... verification is determined by this cell-vector which can have following structure:

{'none'} no verification is performed

{'simulink',*Simulink model name***}** verification using Simulink model. Simulink scheme must have a particular structure see XXX.

{'mixture',*mixture***}** verification using Mixtools mixture. If mixture is empty matrix the estimated Job.Mix is used instead, taking into account Job.Mix is already scaled.

{'matlabfunction',*function***}** verification using Matlab function. Simulated result is obtained from global matrix DATA, which is filled by used controller.
Example verification function:

```
function verifyfun(Job,maxtd)
global TIME
[uchns,ychns] = getuychns(Job.DataDesc.Chns);
```

```
Ctl = Job.Verify.CtlUnscaled;
d = [];
for TIME=maxtd+1:Job.Verify.smlsimlength
   [Ctl,d] = ctlunscaledstep(Ctl,d);
   u = d(uchns); % vector d contains valid elements only for uchns
   [y,u] = vlastni_simulace(u);
   d(ychns)=y; % system output is written to d
   d(uchns)=u; % measured realization of proposed inputs
               % can be fed back to the controller
end
```

Ordering of y and u can be obtained by function `[uchns,ychns] = getuychns(Job.DataDesc.Chns)`

smlsimlength ... length of simulation for verification purposes

smlperiode ... sampling period when using Simulink for verification

CtlScaled ... scaled optimal controller for purpose of verifyopt='matlabfunction'

CtlUnscaled ... un-scaled optimal controller for purpose of verifyopt='matlabfunction'

Job.Mix ... actual processed mixture

Job.MixSingle ... actual processed single-component mixture

Chapter 4

Examples

Examples showing abilities of jobcontrol, are in the directory `mixtools/jobcontrol/examples`.

All experiments are run by the script run_*xxxx*, where *xxxx* is the name of particular experiment.

To document the Jobcontrol package usage, we are giving an example of the LaTeX report, that is created as the result of one experiment run. There are many illustrative experiments together with the Jobcontrol package, we have selected one of them only, the other are present and described in another publication, we are. In these reports, that are automatically created as the result of Jobcontrol run, user can find all the important information concerning the experiment performed.

In the beginning, the experiment setting is given, the result is included in form of tables and figures showing the output of different steps of processing.

The experiment described here is called siso1t. It is the simple single component SISO system example identified and controlled using the mixture approach.

The siso1t experiment protofcol follows.

Experiment: `siso1t` - Simple SISO system experiment with Mixtools Target

Author	:	Miroslav Novak
Contact	:	`mira@utia.cas.cz`
Address	:	AS, UTIA, AV CR, POBox 18, 182 08 Prague, Czech Republic
Basic references	:	
Source texts	:	`siso1t`
Project	:	Designer

4.1 Aims of the study

Experiment "siso1t" is a simple testing system for the Mixtools toolbox using Target function for creating a controller.

4.2 Description of the study

System used for generating identification data and for verification is:

$$y_t = 1.81 y_{t-1} - 0.8187 y_{t-2} + 0.00468 u_t + 0.00438 u_{t-1} + \sqrt{0.001} e_t, \quad e_t \sim N(0,1)$$

4.3 Data

For generating identification data the inputs are

$$u_t \sim N(0,1)$$

4.4 Processing

Whole experiment files are kept under svn in `mixtools/jobcontrol/examples/simple`. Data used for identification are generated by the script `dv_genrawdata_siso1`. To run the experiment:
1) generate the identification data by the `dv_genrawdata_siso1` script, unless it was done before.
2) call "prodini" to initialize mixtools
3) call `jobproceed(siso1t)`, where `siso1t.m` is a function generating the Job description.
4) to generate a nice latex report do `latex siso1t`

4.5 Description

4.5.1 Experiment definition

```
jobname    = 'siso1t';  % name of experiment  ...  no spaces!
% choose short 'jobname' that serves as name root for temporary and saved files
authorname = 'Miroslav Novak';  % author of experiment
email      = 'mira@utia.cas.cz';  % E-mail of author of experiment
address    = 'AS, UTIA, AV CR, POBox 18, 182 08 Prague, Czech Republic';  %
   author's address
references = '';  % references to literature
project    = 'ProDaCTool';  % project name
desc       = 'Simple SISO system experiment with Mixtools Target';  % description
   of experiment printed in protocol
debug      = 0;    % debug level determining information during evaluations

steps    = [1, 1, 1, 1, 1, 1, 1, 1];  %           % Steps to be performed (1/0)
   = (yes/no)
```

4.5.2 Data description

```
datafile     = 'dv_genrawdata_siso1';  % filename with full path specifying
   the mat file with data
varname      = 'rawdata';  % variable name of data in 'datafile'
transpose    = 0;   % transpose data matrix to have channels to rows (1=yes
   0=no)
rescale_data = 1;   % rescale new data (normaly this is necesary every time
   the new data is given, but it does no harm to do it every time)  (1=yes 0=no)
reset_chns   = 0;   % reset selected channels (normally this is needed only
   once, in the beginning, and if new data does not have anything to do with
   the old data)  (1=yes 0=no)

chns         = [1, 2];  % modelled channels
pr_chns      = [1, 2];  % vector of numbers of channels from which original
   data plots are printed to protocol (-1 means all channels)
pr_merge     = 1;  % whether original data plots in protocol are merged or not
   (0=separated, 1=merged, 2=tiled).
used_data    = -1;   % % At the moment, there are 10000 data samples
```

4.5.3 Channels description

```
% Description of the channel 1
chn_name       = 'y';  % name of the channel
chn_oitem      = 1;  % visibility by operator
chn_raction    = 0;  % available for control
chn_prty       = 0.5;  % presentation priority
chn_type       = 1;  % (1/0) = (continuous/discrete) channel
chn_prange    = [-0.0121804, 1.11326];  % expected physical range [min,max]
chn_drange    = [-1000;
1000];  % desired range [min,max]
chn_irange    = [];  % desired range of increments [min,max]
chn_preinfo   = 'olymedian', 'c', 1;  % pre-processing information (see help
   preinit)
% Prior informations follow.  You won't get very good documentation for this,
% the best is what you see here or you can find the file guidex.pdf in svn.
% In all cases prior information stacks under each other forming matrices.
chn_gain      = [];  % [uchn, mingain, maxgain] static gain first column is
   index of input channel and then minimum and maximum
chn_stime     = [];  % sampling time is the scalar variable (unit=seconds)
% It provides the time-scale for different prior informations.
% All prior informations that follow need to have sampling time (chn_stime)
   set,
% because they operate on Hertz (you must give the time-scale)
chn_ampl      = [];  % frequency response [uchn, frequency_in_hertz, amplitude_low,
   amplitude_high, phase_in_degrees]
chn_cut       = [];  % cut-off frequency [uchn, frequency_in_hertz]
chn_tc        = [];  % time constant [uchn, tclow, tchigh]
type = '';
% Description of the channel 2
chn_name       = 'u';  % name of the channel
chn_oitem      = 1;  % visibility by operator
chn_raction    = 1;  % available for control
chn_prty       = 0.5;  % presentation priority
chn_type       = 1;  % (1/0) = (continuous/discrete) channel
chn_prange    = [0.207821, 0.784081];  % expected physical range [min,max]
chn_drange    = [-1;
1];  % desired range [min,max]
chn_irange    = [];  % desired range of increments [min,max]
chn_preinfo   = 'olymedian', 'c', 2;  % pre-processing information (see help
   preinit)
% Prior informations follow.  You won't get very good documentation for this,
% the best is what you see here or you can find the file guidex.pdf in svn.
% In all cases prior information stacks under each other forming matrices.
chn_gain      = [];  % [uchn, mingain, maxgain] static gain first column is
   index of input channel and then minimum and maximum
chn_stime     = [];  % sampling time is the scalar variable (unit=seconds)
% It provides the time-scale for different prior informations.
% All prior informations that follow need to have sampling time (chn_stime)
   set,
% because they operate on Hertz (you must give the time-scale)
chn_ampl      = [];  % frequency response [uchn, frequency_in_hertz, amplitude_low,
   amplitude_high, phase_in_degrees]
chn_cut       = [];  % cut-off frequency [uchn, frequency_in_hertz]
chn_tc        = [];  % time constant [uchn, tclow, tchigh]
type = '';
```

4.5.4 Prior information

```
ncom    = 1;       % the number of components
ord     = 2;       % order of the richest regressor
diacove = 0.0001;  % diagonal of noise covariancecove
diaCth  = 10000;   % diagonal of par.  covariance
dfm     = 1000;     % degrees of freedom of factors
dfcs    = 1000;  % degrees of freedom of components

doflattening = 0;  % indicates whether the initial mixture should be flattened
```

4.5.5 Mixture initialization

```
SingleOnly  = 0;       % boolean flag whether to skip mixinit and calculate
   just MixSingle
opt     = 'p';    % iterative estimation method (p | q | b | f | Q | P)
niter   = 10;    % maximum number of iterations
frg     = 0.999999;    % forgetting factor
```

4.5.6 Mixture estimation

```
opt         = 'p';       % iterative estimation method (p | q | b | f | Q |
   P)
niter       = 40;           % maximum number of iterations
frg         = 0.999999;           % forgetting factor
frgEstType  = 0;           % estimation type of forgetting factor (0-none, 1-zero
   alt, 2-prev estimate)
frgEstGrid    = [1, 0.99, 0.983362, 0.972317, 0.953941, 0.923366, 0.872495,
   0.787855, 0.647029, 0.412721, 0.022876];  %
% estimation grid of forgetting factor
```

4.5.7 Mixture validation

```
nsteps  = 1;     % the prediction is made nsteps ahead
pchns    = [1, 2];  % channels to be predicted
cchns    = [];  % channels in condition
tstart  = 1;     % starting time determining the part of data used in validation
tend  = 10000;  %   ending time determining the part of data used in validation
epss  = 1;     % (1/0) = (produce/do not produce) encapsulated postscript plots
pauses  = 0;     % pause in seconds after each plot set
plots    = [0, 0, 0, 0];  % [show cluster plot , show time plot, mixture plot,
   histogram], where (1/0)=(y/n)
segments    = 0;     % number of segments for segmentation test (0 means perform
   no segm.  test).  These tests take a very long time.
alt         = 1;     % perform validation using alternative forgetting estimation
   test (takes very long time, 0=do not run, 1=run this test).
```

4.5.8 User ideal

```
method = 't';  % method determining the user target (t | d | z)
% t ...  User defined (initialized by target.m) - this is default option
% d ...  Designer is used
% z ...  User defined (initialized by zeros)
userSetpointEths   = [0, 0];  % user target component Eths
userSetpointCoves  = [1e+06, 1];  % user target component Coves
userSetpointCorrect  = 1;  % do the user target component Correction ?  (0=no,1=yes)
incremental  = 0;  % use incremental penalization controller ?  (0=no,1=yes)
```

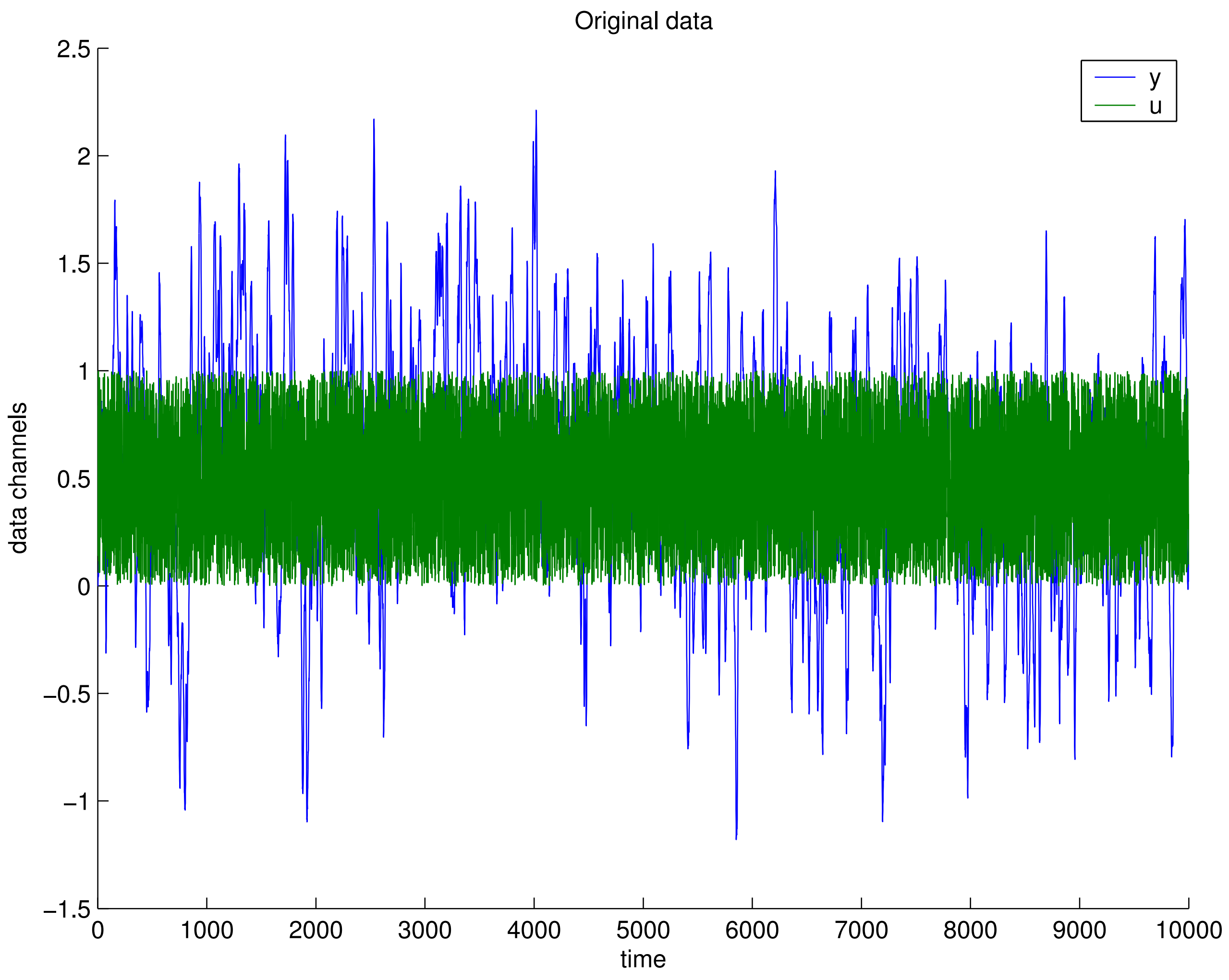
4.5.9 Design

```
designtype = 'i';  % design type (a | i | s)
% a ...  academic design
% i ...  industrial design
% s ...  simultaneous design
horizon    = [50, -1];  % horizon for evaluation of KLD
ufc        = [];  % ufcgen(aMix, aMixu) is used if ufc=[]
```

4.5.10 Verification

```
method = 't';  % verification method (t | d)
% t ...  verification of controller designed by step 7
% d ...  verification of controller designed by UserIdeal with Designer in step
   6
type = 'mixture'; % simulation type (none,simulink,mixture)
MixVer = varload('siso1t_mixver','MixVer'); % verification mixture, enter empty
   matrix to use the identification result
smlsimlength = 1000;     % simulation length for verification
```

4.5.11 Original Data Plots



4.6 Results

4.6.1 Results of Mixture Identification and Parameter Estimation (and Model Validation)

Comprehensive tests of the model validity:

Value of mixll:	1.959e+03	(the bigger the better)
Test of validity of the model:	1	(1=O.K.,0=bad)
Relative SE of pred.err:	[0.000563259, 0.0100006]	(standard error of ep relative to std of data)
Test of whiteness:	[0.0569913, 0.0524522]	
	(sum of correlations with delayed predictions)	

Elementary statistics for the channel y:

	MIN	MAX	MEAN	MEDIAN	STD
data	-1.17939	2.21166	0.550652	0.559995	0.562724
differences	-0.198751	0.228216	1.91239e-05	-0.000612289	0.0557418
predictions	-1.22481	2.24095	0.550465	0.559804	0.561802
errors of prediction	-0.123575	0.116885	0.000186817	0.00047614	0.0316938

Elementary statistics for the channel u:

	MIN	MAX	MEAN	MEDIAN	STD
data	9.01343e-05	0.999918	0.496014	0.495356	0.288124
differences	-0.98473	0.988182	-7.35039e-05	0.00352103	0.406616
predictions	0.48947	0.503043	0.496339	0.496339	0.00269178
errors of prediction	-0.499192	0.508115	-0.000324842	-0.000919476	0.288127

Noise noise-variance estimates for individual factors:

	1	2	dfcs
component 1	0.00332	0.0908	0.184
component 2	0.00295	0.0481	0.139
component 3	0.00288	0.0105	0.0809
component 4	0.0031	0.133	0.136
component 5	0.00304	0.113	0.169
component 6	0.0029	0.0113	0.0769
component 7	0.0027	0.0296	0.0816
component 8	0.00273	0.161	0.132

Mixture Factors

This mixture has 8 components with 2 factors each. Mixture consists of ARX factors. Interpretation of tables below: Each column correspond to one delay. Each row corresponds to the channel, and the first row employs the offset.

1. component, dfcs=1992.764705, factors:
   Factor 1, modelled channnel: 1, called 'y', cove=0.00332444, dfm=1396.443574

delays →		0	1	2
offset			0.0254656	
1	y		1.81056	-0.818404
2	u	0.0386102	0.00306867	0.00264698

   Factor 2, modelled channnel: 2, called 'u', cove=0.0908168, dfm=643.449676

delays →		0	1	2
offset			-0.84018	
1	y		-0.0248394	
2	u		0.0214779	-0.0253872

2. component, dfcs=1507.208982, factors:
   Factor 3, modelled channnel: 1, called 'y', cove=0.00295473, dfm=1042.004114

delays →		0	1	2
offset				
1	y		1.81278	-0.825892
2	u	0.00549226		

   Factor 4, modelled channnel: 2, called 'u', cove=0.0480899, dfm=538.067667

delays →		0	1
offset			1.17689
1	y		
2	u		

3. component, dfcs=877.848773, factors:
   Factor 5, modelled channnel: 1, called 'y', cove=0.002882, dfm=759.558895

delays →		0	1	2
offset				
1	y		1.82671	-0.835076
2	u		0.00357796	0.00239426

   Factor 6, modelled channnel: 2, called 'u', cove=0.0105385, dfm=588.537443

delays →		0	1
offset			-1.55992
1	y		
2	u		0.00852314

4. component, dfcs=1477.492899, factors:
   Factor 7, modelled channnel: 1, called 'y', cove=0.003096, dfm=730.192481

delays →		0	1	2
offset			-0.0094869	
1	y		1.73629	-0.733612
2	u			-0.00625134

   Factor 8, modelled channnel: 2, called 'u', cove=0.132959, dfm=296.212771

delays →		0	1
offset			0.575572
1	y		
2	u		0.0626245

5. component, dfcs=1835.268888, factors:
   Factor 9, modelled channnel: 1, called 'y', cove=0.0030384, dfm=1034.829765

delays →		0	1	2
offset			-0.00504452	
1	y		1.81475	-0.822106
2	u			

   Factor 10, modelled channnel: 2, called 'u', cove=0.11339, dfm=387.774721

delays →		0	1
offset			-0.238939
1	y		
2	u		

6. component, dfcs=834.370300, factors:
   Factor 11, modelled channnel: 1, called 'y', cove=0.00289895, dfm=732.072393

delays →		0	1	2
offset			-0.0857733	
1	y		1.82453	-0.832945
2	u	0.056043	0.00511369	

   Factor 12, modelled channnel: 2, called 'u', cove=0.0112816, dfm=548.187967

delays →		0	1
offset			1.58209
1	y		
2	u		

7. component, dfcs=884.666363, factors:
   Factor 13, modelled channnel: 1, called 'y', cove=0.00270471, dfm=352.131763

delays →		0	1	2
offset			0.0264342	
1	y		1.81473	-0.825945
2	u	0.0115676		-0.00416591

   Factor 14, modelled channnel: 2, called 'u', cove=0.0296442, dfm=203.645471

delays →		0	1	2
offset			-1.22122	
1	y			0.0136029
2	u			0.0120795

8. component, dfcs=1435.531948, factors:
   Factor 15, modelled channnel: 1, called 'y', cove=0.00273428, dfm=575.673990

delays →		0	1	2
offset			0.0186852	
1	y		1.88061	-0.898927
2	u	-0.0254451	0.00211818	0.00469921

   Factor 16, modelled channnel: 2, called 'u', cove=0.160593, dfm=250.840317

delays →		0	1	2
offset			0.440853	
1	y			0.0127503
2	u		-0.035918	

4.6.2 Results of Single Component Identification and Parameter Estimation (and Model Validation)

Comprehensive tests of the model validity:

Value of mixll:	3.682e+02	(the bigger the better)
Test of validity of the model:	0	(1=O.K.,0=bad)
Relative SE of pred.err:	[0.000563556, 0.0100005]	(standard error of ep relative to std of data)
Test of whiteness:	[0.0549918, 0.0507126]	
	(sum of correlations with delayed predictions)	

Elementary statistics for the channel y:

	MIN	MAX	MEAN	MEDIAN	STD
data	-1.17939	2.21166	0.550652	0.559995	0.562724
differences	-0.198751	0.228216	1.91239e-05	-0.000612289	0.0557418
predictions	-1.22471	2.24139	0.550642	0.559758	0.561829
errors of prediction	-0.125308	0.117345	1.0206e-05	0.000114461	0.0317111

Elementary statistics for the channel u:

	MIN	MAX	MEAN	MEDIAN	STD
data	9.01343e-05	0.999918	0.496014	0.495356	0.288124
differences	-0.98473	0.988182	-7.35039e-05	0.00352103	0.406616
predictions	0.496017	0.496017	0.496017	0.496017	6.37855e-14
errors of prediction	-0.495927	0.503901	-2.78884e-06	-0.000660595	0.288124

Noise noise-variance estimates for individual factors:

	1	2	dfcs
component 1	0.00341	1.07	1

Mixture Factors

This mixture has 1 components with 2 factors each. Mixture consists of ARX factors. Interpretation of tables below: Each column correspond to one delay. Each row corresponds to the channel, and the first row employs the offset.

1. component, dfcs=10845.145960, factors:
   Factor 1, modelled channnel: 1, called 'y', cove=0.00341127, dfm=10845.145972

delays →		0	1	2
offset				
1	y		1.81299	-0.821943
2	u			

Factor 2, modelled channnel: 2, called 'u', cove=1.07418, dfm=10845.146001

delays →		0	1
offset			0.000229105
1	y		
2	u		

4.6.3 User Ideal Mixture

This mixture has 1 components with 2 factors each. Mixture consists of ARX factors. Interpretation of tables below: Each column correspond to one delay. Each row corresponds to the channel, and the first row employs the offset.

1. component, dfcs=1.000000, factors:
   Factor 1, modelled channnel: 1, called 'y', cove=0.00108019, dfm=1.000000

delays →		0	1
offset			-0.978354
1	y		
2	u		

Factor 2, modelled channnel: 2, called 'u', cove=0.089177, dfm=1.000000

delays →		0	1
offset			-1.72128
1	y		
2	u		

4.6.4 Controller Mixture

This mixture has 8 components with 2 factors each. Mixture consists of ARX factors. Interpretation of tables below: Each column correspond to one delay. Each row corresponds to the channel, and the first row employs the offset.

1. component, dfcs=1992.764705, factors:
   Factor 1, modelled channnel: 1, called 'y', cove=0.00332444, dfm=1396.443574

delays →		0	1	2
offset			0.0254656	
1	y		1.81056	-0.818404
2	u	0.0386102	0.00306867	0.00264698

Factor 2, modelled channnel: 2, called 'u', cove=0.035041, dfm=1.000000

delays →		0	1	2
offset			-2.10251	
1	y		-12.4589	9.19651
2	u		-0.0540221	-0.00100543

2. component, dfcs=1507.208982, factors:
   Factor 3, modelled channnel: 1, called 'y', cove=0.00295473, dfm=1042.004114

delays →		0	1	2
offset				
1	y		1.81278	-0.825892
2	u	0.00549226		

Factor 4, modelled channnel: 2, called 'u', cove=0.035041, dfm=1.000000

delays →		0	1	2
offset			-2.10251	
1	y		-12.4589	9.19651
2	u		-0.0540221	-0.00100543

3. component, dfcs=877.848773, factors:

Factor 5, modelled channnel: 1, called 'y', cove=0.002882, dfm=759.558895

delays →		0	1	2
offset				
1	y		1.82671	-0.835076
2	u		0.00357796	0.00239426

Factor 6, modelled channnel: 2, called 'u', cove=0.035041, dfm=1.000000

delays →		0	1	2
offset			-2.10251	
1	y		-12.4589	9.19651
2	u		-0.0540221	-0.00100543

4. component, dfcs=1477.492899, factors:

Factor 7, modelled channnel: 1, called 'y', cove=0.003096, dfm=730.192481

delays →		0	1	2
offset			-0.0094869	
1	y		1.73629	-0.733612
2	u			-0.00625134

Factor 8, modelled channnel: 2, called 'u', cove=0.035041, dfm=1.000000

delays →		0	1	2
offset			-2.10251	
1	y		-12.4589	9.19651
2	u		-0.0540221	-0.00100543

5. component, dfcs=1835.268888, factors:

Factor 9, modelled channnel: 1, called 'y', cove=0.0030384, dfm=1034.829765

delays →		0	1	2
offset			-0.00504452	
1	y		1.81475	-0.822106
2	u			

Factor 10, modelled channnel: 2, called 'u', cove=0.035041, dfm=1.000000

delays →		0	1	2
offset			-2.10251	
1	y		-12.4589	9.19651
2	u		-0.0540221	-0.00100543

6. component, dfcs=834.370300, factors:

Factor 11, modelled channnel: 1, called 'y', cove=0.00289895, dfm=732.072393

delays →		0	1	2
offset			-0.0857733	
1	y		1.82453	-0.832945
2	u	0.056043	0.00511369	

Factor 12, modelled channnel: 2, called 'u', cove=0.035041, dfm=1.000000

delays →		0	1	2
offset			-2.10251	
1	y		-12.4589	9.19651
2	u		-0.0540221	-0.00100543

7. component, dfcs=884.666363, factors:

Factor 13, modelled channnel: 1, called 'y', cove=0.00270471, dfm=352.131763

delays →		0	1	2
offset			0.0264342	
1	y		1.81473	-0.825945
2	u	0.0115676		-0.00416591

Factor 14, modelled channnel: 2, called 'u', cove=0.035041, dfm=1.000000

delays →		0	1	2
offset			-2.10251	
1	y		-12.4589	9.19651
2	u		-0.0540221	-0.00100543

8. component, dfcs=1435.531948, factors:

Factor 15, modelled channnel: 1, called 'y', cove=0.00273428, dfm=575.673990

delays →		0	1	2
offset			0.0186852	
1	y		1.88061	-0.898927
2	u	-0.0254451	0.00211818	0.00469921

Factor 16, modelled channnel: 2, called 'u', cove=0.035041, dfm=1.000000

delays →		0	1	2
offset			-2.10251	
1	y		-12.4589	9.19651
2	u		-0.0540221	-0.00100543

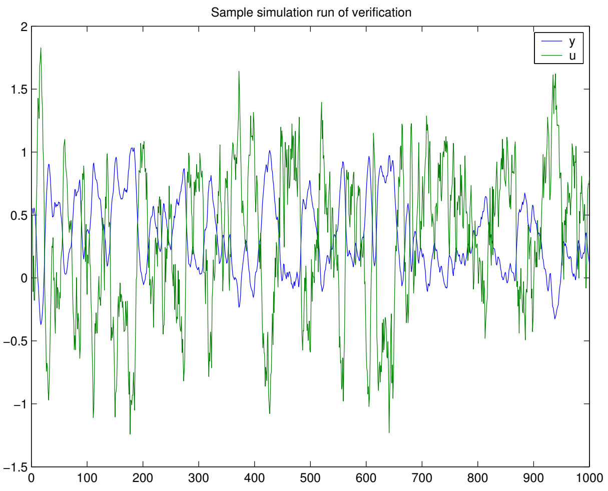
4.7 Experimental Controller Verification Results

Elementary statistics for simulated channels:

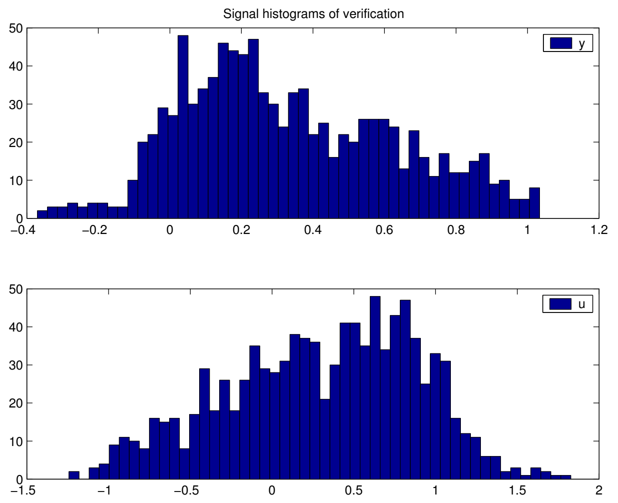
channel	mean	variance	range	constr.sat.
1 "y"	0.33103	0.295568	[-0.370158, 1.03331]	1
Δ 1 "y"	-0.000441788	0.0505706	[-0.158734, 0.156836]	1
2 "u"	0.304823	0.587081	[-1.24137, 1.82741]	0.887
Δ 2 "u"	0.000308341	0.221705	[-0.817178, 0.712314]	1

Note: Symbol Δ means increments of the signal, "range" means the minimum a maximum of simulated signal values, "constr.sat." means constraints satisfaction ratio for given channel and constraint described in the Section Channel Description.

Sample simulation signals :



Simulation signals histogram :



4.8 Conclusion

The identification part of processing found a mixture of 8 components. The components are very similar and close to the parameters of the model used for generating of the identification data. The controller designed works well according to the cerification.

Chapter 5

Summary and Conclusions

With Jobcontrol user-friendly interface, Mixtools package becomes powerful set of utilities for system identification employing Gaussian mixture model. Together with the Designer toolbox, which serves the purpose of finding optimal controller parameters, and consequently for constructing ideal controller. The choice for Matlab environment makes it possible to simplify complicated matrix calculations connected with the task of system identification and visualize results in a very convenient way.

Chapter 6

Acknowledgements

This report was supported by MŠMT 1M0572 (DAR), project AV ČR 1075351, project GA ČR 102/03/0049, and project AV ČR 1ET100750401 (Baddyr).

Bibliography

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

[2] P. Nedoma, M. Kárný, T.V. Guy, I. Nagy, and L. Tesař. Learning and prediction with normal mixtures. Technical Report 2045, UTIA AV CR, 2002.

[3] V. Peterka. Bayesian system identification. In P. Eykhoff, editor, *Trends and Progress in System Identification*, pages 239–304. Pergamon Press, Oxford, 1981.