WIDE Toolbox Manual

Davide Barcelli and Nick Bauer and Pavel Trnka

Contents

Introduction

The Toolbox for Matlab is named after the project "WIDE - Decentralized and Wireless Control of Large-Scale Systems", contract num- ber FP7-IST-224168 of the European Commission which founded its development and is downloadable for free at the URL `http://ist-wide.dii.unisi.it/index.php?p=toolboxsp`.
The WIDE Toolbox for Matlab is set of functionalities oriented to centralized and decentralized/distributed Large-Scale control that takes explicitly into account network effects. The use of Object-Oriented programming ensures the usability of the present tools in any context.
Due to both wiring costs and improved reliability of wireless communication, nowadays many control systems sensor-to-controller and controller-to-sensor connections are implemented in the filed without a physical wire. Such fact expose the plant to inconstant, unpredictable but "describable" negative effects introduced by the *network* between the control system components.
Moreover, the layout of the plant may cover a considerably large physical space on the the field thus worsening the network effects. Such wide spread is closely connected with the complexity of the control system that is composed by many outputs/inputs and consequently internal states. In almost all systems actuator saturations or safety policies impose some sort of constraint involving plant variables, therefore making it crucial to have a constraint-handling controller.
In this Toolbox the general orientation is toward Model Predictive Control (MPC), which however suffer the so called "curse of dimensionality", *i.e.* the control computation complexity increases with the size of the plant. To cope with this and successfully apply the control approach to large-scale systems we propose decentralized/distributed versions of MPC in which the main idea is to decompose the plant model in a number of independent subsystem. Then for each of subsystem the control law is computed individually and a coordination/iterative consensus procedure may take place afterwords to improve performances and generalization.

Components

The toolbox is organized in three main areas, dealing with previously listed issues.

- DHMPC (developed by UNISI[1], UNITN[2] and IMTL[3]) offer to the user decentralized control strategies that accounts for some major network effects(dlincon,HiMPC,decLMI), facilities for wireless network simulation using TrueTime (ACG), energy aware control strategies (eampc);
- LSMM (developed by HPL[4]) is mostly oriented to model and analyze a large-scale system by providing standard analysis functionalities (LSmodel), *e.g.* Bode and Nyquist diagrams, and decentralization procedure (ϵ-decomposition), with particular focus to water networks (WNmodel) and network aware Kalman filtering (NKF);
- NCS (developed by TU/e[5]) is concerned with network effects which are dealt with using discretized and hybrid models (ncs) and network linear controller synthesis available also with a Graphical User Interface.

For a detailed description of each component the reader is redirected to either component class description or the corresponding section of the examples.

Requirements

The offered functionalities cover a wide spread of requirements, thus find below an individual list.

[1]University of Siena
[2]University of Trento
[3]Institution Market Technologies - Institute for Advanced Studies Lucca
[4]Honeywell Prague
[5]Eindhoven University of Technology

- *acg*, Automatic Code Generation is an interface for TrueTime 1.5 code generation, thus requires TrueTime.
- *decLMI* Decentralized LMI is a tool for synthesize linear decentralized controller for TLI systems that needs to solver a Semi Definite Programming problem, thus requires Yalmip, included in MPT Toolbox, and, possibly, Cplex as solver;
- *dlincon* is a decentralized version of lincon, function of the Hybrid toolbox for linear MPC, require the hybrid toolbox;
- *eampc* Energy Aware MPC is a sensor battery saving control policy, requires the MPT Toolbox;
- *HiMPC* is a Decentralized Herarchical MPC strategy, requires MPT Toolbox and, possibly, Cplex solver.
- *LSmodel* is Large Scale Model Management class, Control System Toolbox (included in Matlab) and BGL Toolbox;
- *ncs* Networked Control System requires Linsyskit;
- *NKF* Networked Kalman Filer have ho requirements;
- *sensorCommunicator* [6] is an interface to E-Senza nodes[7] and Telos Motes devices [8] and requires the devices;
- *WNmodel* is an extension of LSmodel specialized for Water networks, thus requires LSmodel.

In the toolbox download page a zipped version of all requirements (excluding IBM Ilog Clpex, shareware software available for universities via IBM Academic Initiative) is available. Some components are boosted in terms of execution speed by using Matlab Mex Files (included); for a guide on mex compilation see `www.dii.unisi.it/~barcelli/software.php`

[6] Still under development
[7] `http://www.e-senza.com/`
[8] `http://www.willow.co.uk/html/telosb_mote_platform.html`

Chapter 1

Classes definition

1.1 LSmodel

`LSmodel` is a class for representation and management of Large Scale (LS) models. LS model is described as a set of submodels together with description of submodels mutual interconnections and submodels interconnections with external inputs and outputs.

Signal interconnections is name based, i.e. to define that outputs and intpus of certain submodels are conneted requires to give them the same names. Submodels are internally represented as state-space models. However any model type, which can be converted by SS command, can by supplied as submodel and original model is stored in LSmodel class along with its state-space conversion.

LS model is created from a set of submodels with named inputs and outputs (at least interconnection signals should be named), a set of summator created for example by SUMBLK commands and string cell arrays defining external inputs and outputs (see `LSmodel_example.m`). The model can be later modyfied by adding new submodels, removing submodels, adding/removing external inputs/outputs.

LS model supports specification of different signal types for each signal

Signal Type	Abbreviation	Channel
Manipulated	MV	Input
Measured Disturbances	MD	Input
Unmeasured Disturbances	UD	Input
Measured Outputs	MO	Output
Unmeasured Outputs	UO	Output
Internal Signal	X	--

and specification of signal limits (min/max values, min/max slopes,...). This information is later used for estimator and/or controller design.

`LSmodel` has methods for structured model order reduction, decomposition of subsystems into groups for distributed control/estimation and merging of information from multiple models into single one.

Model structure can be ploted by overloaded plot command and model analysis can be done by many standard overloaded functions (see bellow).

State-space model of full `LSmodel` can be obtained by SS method. Parts of model can be obtained either for selected inputs and outputs or by extracting group of subsystems, where grouping information may be a result of built-in decomposition method.

Methods

```
LSmodel      - constructor, compatible with Matlab connect function

add_mod      - adds submodel to LS model
add_sum      - adds summator (creates new internal signal)
add_ext_inp  - adds external input to LS model
add_ext_out  - adds external output from LS model
rem_mod      - removes submodel from LS model
rem_sum      - removes summator
rem_ext_inp  - removes external input
rem_ext_out  - removes external output

set_sig_type - signal types according to the MPC toolbox
set_sig_lim  - set signal limits (min/max, delta min/max)
set_sig_data - assigns (user) data structure to selcted signal(s)

select       - extracts part of the model for selected inputs and
               outputs
group        - extracts subsystems belonging to given group
squeeze      - remove unused external inputs/outputs and unconnected
               submodels

struct_red   - structured model order reduction
merge        - merging of ARX models of different structure
freq_uncert  - return frequency uncertainty for ARX model
eps          - epsilon decomposition
bbd          - Border Block Diagonal decomposition

display      - presents the model content and shows the numbering of
               subsystems, summators, inputs and outputs
n            - returns total order (sum of subsystems order)
orders       - returns subsystems orders
plot         - plots interaction between subsystems
```

Overloaded functions

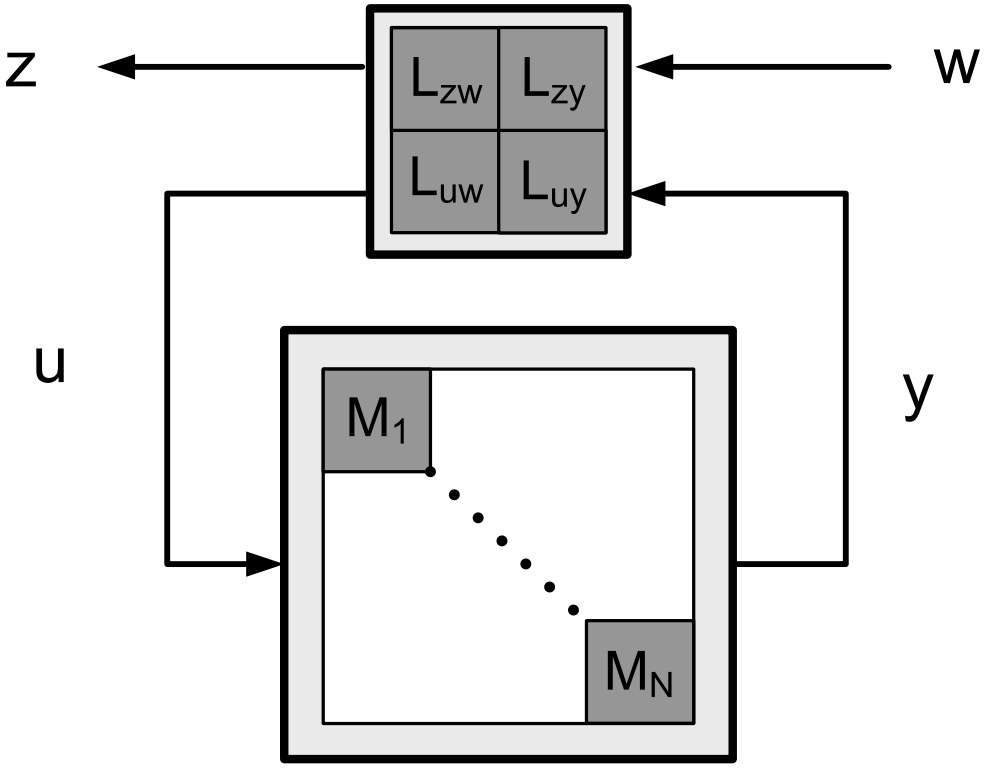
```
dcgain, pole, zero, impulse, step, bode, nyquist, pzmap, iopzmap,
ss, ...
```

Internal model structure

Internal model structure can be seen in the following figure. The signal names are:

```
w ... external inputs to LS model
z ... external outputs from LS model
u ... aggregated inputs to submodels
y ... aggregated outputs from submodels
```

Submodels are stored in cell array `M` as state-space models. The original models supplied by the user are stored in cell array `Morig`. External model inputs are stored in string cell array `Wnames` and outputs in `Znames`. Static interconnections matrices Lzw,Lzy,Luw,Luy represent submodels mutual interconnections and submodel interconnections with external inputs and outputs. They are automatically computed after each model modification.



Contents

- Extract model part for given external inputs and outputs
- Extract submodels belonging to given group
- Remove unused external inputs/outputs and submodels
- Structured Model Order Reduction
- Merging of uncertain models
- Frequency uncertainty of ARX model
- Static Methods
- Plot LS model structure

LSmodel Constructor

`LSMODEL( X1,...,Xn, INPUT_NAMES, OUTPUT_NAMES )` constructs large scale model by specifying a set of submodels, summators and a list of external input and output names. Internal connections are automatically created by matching signal names. Xi are models or summators (single or in a cell array). INPUT_NAMES and OUTPUT_NAMES are cell arrays of strings with external input and output names. Summators can be created by command SUMBLK. Examples:

```
mod = LSmodel(M,sum,{'fuel','demand'},{'pressure','flow'});

mod = LSmodel(M1,M2,M3,M4,sum1,sum2,{'fuel','demand'},{'pressure','flow'});
```

M is cell array of models (ss,tf,...) with named inputs and outputs. SUM is cell array of summators created by SUMBLK.

Constructor is compatible with standard CONNECT function. It is also possible to construct the model by using CONNECT compatible numeric indexing of inputs and outputs (see help on CONNECT). However, name based signal referencing will be prefered as numeric indexing may be confusing for large scale models.

Remove Model

`REM_MOD(OBJ,PAR)` removes model from large scale system. PAR is a vector of indexes or string cell array of model names to be deleted. (model indexes are shown by display function)

Remove Summator

`REM_SUM(OBJ,PAR)` removes summator from large scale system. PAR is the index or the name of the summator to be deleted. Summator indexes are shown by display function.

Add Model

`ADD_MOD(OBJ,NEWM)` adds new model NEWM to large scale model class and connects it according to the input / output names. NEWM can be a cell array of models.

Add Summator

`ADD_SUM(OBJ,NEWS)` adds new summator to the model and connects it according the input/output names. Numeric based interconnection is not possible. NEWS can be an array of summators.

Summator can be created by SUMBLK function.

Remove External Input

`REM_EXT_INP(OBJ,PAR)` Removes external input. PAR can be input index or its NAME.

Add External Input

`ADD_EXT_INP(OBJ,NAME)` adds external input named NAME.

Remove External Output

`REM_EXT_OUT(OBJ,PAR)` Removes external output. PAR can be output index or its NAME.

Add External Output

`ADD_EXT_OUT(OBJ,NAME)` adds external output named NAME.

Display Object

`DISPLAY` Displays basic information about LSmodel.

Return Global Model Order

`N=N(OBJ)` returns global model order as a sum of subsystem orders.

Return Orders of Subsystems

`N=ORDERS(OBJ)` returns orders of subsystems in array

Get Global SS Model

`MOD=SS(obj)` returns state space model of the whole model.

Epsilon Decomposition

`EPS(OBJ,N_TARGET)` finds \epsilon decomposition of water network model after leafs condensation. Decomposes the network to N_TARGET subnetworks.

Set Signal Type

`SET_SIG_TYPE(OBJ,NAMES,TYPE)` - set type of signals for MPC design. Signal types are compatible with MPC toolbox:

Signal Type	Abbreviation	Channel
Manipulated	MV	Input
Measured Disturbances	MD	Input
Unmeasured Disturbances	UD	Input
Measured Outputs	MO	Output
Unmeasured Outputs	UO	Output
Internal Signal	X	--

MODEL is name or index of submodel. SIGNALS is a cell of input names or their indexes and TYPE is abbreviation of signal type (MV,MD,UD).

Set Signal Limits

`SET_SIG_LIM(OBJ,NAMES,MIN,MAX,DMIN,DMAX,SMIN,SMAX)` sets MIN / MAX limits on absolute values, sets DMIN / DMAX limits on speed of change and sets SMIN / SMAX soft limits on absolute values. Limits are applied to signal(s) specified by name in NAMES

`SET_SIG_LIM(OBJ,NAMES,MIN,MAX,DMIN,DMAX)` sets MIN / MAX limits on absolute values and sets DMIN / DMAX limits on speed of change.

`SET_SIG_LIM(OBJ,NAMES,MIN,MAX)` sets MIN / MAX limits on absolute values.

`SET_SIG_LIM(OBJ,NAMES)` removes all limits (sets them to inf).

Set Signal Data

`SET_SIG_DATA(OBJ,NAMES,DATA)` assigns user data structure to signal(s) specified in NAMES.

Master Connection Matrix

`[MZW,MZY,MUW,MUY] = MASTERL(OBJ)` returns aggregated matrices indicating connection between external inputs, submodels and external outputs.

Extract model part for given external inputs and outputs

`NOBJ = SELECT(OBJ,INPUTS,OUTPUTS)` extracts part of the model for selected inputs and outputs. Submodels, which are not conotrllable or observable from selected inputs and outputs are not included in extracted model. INPUTS and OUTPUTS can be cell array of inputs/outputs signal names or their numerical index (see DISP).

`NOBJ = SELECT(OBJ,INPUTS,OUTPUTS,OPT)` parameter OPT controls, which submodels are kept and which are removed from final model:

```
'io' ........ eliminates submodels, which are not controllable and/or
              observable
'full' ...... keep all submodels
'ss' ........ (default) eliminates submodels which are not
              controllable and observable are removed
```

Extract submodels belonging to given group

`NOBJ = GROUP(OBJ,GR_IND)` extracts parts of LS model, where submodels belong to the group(s) specified in vector GR_IND.

Remove unused external inputs/outputs and submodels

`NOBJ = SQUEEZE(OBJ)` removes external inputs and outputs, which are not connected to any submodels and also removes submodels which are not connected anything.

Structured Model Order Reduction

`OBJ = STRUCT_RED(OBJ,N_TARGET)` reduces total model order to N_TARGET while preserving model structure.

Algorithm: Henrik Sandberg, Richard M. Murray: "Model reduction of interconnected linear systems". Optimal Control, Applications and Methods, Special Issue on Directions, Applications, and Methods in Robust Control, 30:3, pp. 225–245, May/June 2009.

Merging of uncertain models

`MERGE(OBJ,MOD,NEW_MOD,TARGET_NB,TARGET_NA)` updates uncertain submodel by new model. Updated submodel is determined by its index or its name in variable MOD and new model is specified by NEW_MOD. This function supports so far only ARX models (IDARX or IDPOLY classes of submodels).

Algorithm:

P. Trnka and V. Havlena, Overlapping models merging and interconnection for large-scale model management. Proceedings of IEEE Multi-Conference on Systems and Control. 2010.

Frequency uncertainty of ARX model

`[MEAN,VAR] = FREQ_UNCERT(OBJ,MOD,W)` returns mean value and variation for model MOD on frequencies W. This function supports so far only ARX models (IDARX or IDPOLY classes of submodels).

Static Methods

Plot LS model structure

PLOT(MOD) plots structure of large scale model defined in MOD (LSmodel class object). The structure is plotted as a graph, where vertexes are subsystems and edges indicate interaction between subsystems. Small triangles at vertex edges indicate which subsystems have direct connection to external inputs/outputs.

PLOT(MOD,'IO') plots inputs and outputs as additional vertexes.

It is useful to call PLOT command as

```
MOD = PLOT(MOD)
```

as plot automatically computes graph vertex position and stores it for later use. The computation can be time consuming for larger models (>50 vertexes).

1.2 WNmodel

WNmodel class is a child of large scale model LSmodel used for modeling of water (distribution) networks.

Methods:

```
WNmodel       - constructor creating empty model or importing model from
                definition file
import_scheme - imports water network model from file
plot          - overloaded plot function

eps           - overloaded epsilon decomposition for water networks
```

Contents

- WNmodel Constructor
- Import water network scheme description from file.
- Display Object
- Epsilon Decomposition
- Compute Tanks Incidence Matrix
- Plot Water Network Model

WNmodel Constructor

`WNMODEL( FILENAME )` constructs empty water network model or imports model from given definition file (see `IMPORT_SCHEME` for description of file formats).

Import water network scheme description from file.

`NOBJ = IMPORT_SCHEME(OBJ,FILENAME)` import water netwerk scheme from files. Water network is defined by two files [filename].net and [filename].mat, where '.net' defines network structure and '.net' defines variables limits, prices, etc.

Structure definition file is a simple text file describing individual tanks and their interconnections by specifying signal names and directions.

The first line of each tank definition block specifies tank number and its name:

```
Tank##,<tank name>
```

or it can specify node with it number

```
Node##
```

Following lines are same for tanks and nodes:

```
d,<demand name>
s,<source name>
+,<outlet pump/valve name>,<destination tank name>
-,<inlet pump/valve name>,<source tank name>
```

Tank/node definition blocks can be separated by empty line(s). Empty lines and Matlab type comments are ignored.

Variable definition file is standard Matlab MAT file with struct for every signal defining limits by fields

```
.min  ... min value
.max  ... max value
.dmin ... min slope value
.dmax ... max slope value
.smin ... soft limit min value
.smax ... soft limit max value
.type ... signal type ('MV','MD','UD','MO','MV','X')
```

Example (part of definition file):

```
Tank01,d450BEG
d,c450BEG
-,iBegues4,d369BEG

Tank02,d369BEG
d,c369BEG
+,iBegues4,d450BEG
-,iBegues3,d255BEG
```

```
Node1
s,AportA
d,c82PAL
+,vPalleja70,Node2
+,vPapiolATLL,d110PAP
+,iPapiol2AGBAR,d110PAP
+,vFontSanta,d54REL

Node2
d,c70PAL
+,iPalleja4,d125PAL
-,vPalleja70,Node1

<End of example>
```

Display Object

`DISPLAY` shows basic information about water network model.

Epsilon Decomposition

`EPS(OBJ,N_TARGET)` finds \epsilon decomposition of water network model after leafs condensation. Decomposes the network to N_TARGET subnetworks.

Compute Tanks Incidence Matrix

Incidence matrix I determines if there is direct connection between tanks i and j by "1" on position (i,j).

Plot Water Network Model

PLOT(WNmodel) plots structure of water network. It is useful to call PLOT command as

```
MOD = PLOT(MOD)
```

as plot automatically computes graph vertex position and stores it for later use. The computation can be time consuming for larger models (>50 vertexes).

1.3 acg

The class Automatic Code Generation (ACG) creates a ready-to-use setup networked control system simulation by generating a Simulink model based on TrueTime blocks and corresponding configuration m-files. The Network model is specified by the number of actuators and sensors, while controller and plant should be customized by the user fulfilling the respective subsystem.

Contents

Class description

`TrueTime`

is a very powerful tool which can handle simulation of a variety of tasks, with a particular focus on real-time implementation of controller and sensors/actuators codes. As a result of this plenty of possibilities offered, the user needs to set-up a lot of features, some of which might be out of scope for some needings.

This class permits to generate automatically all the configuration m-files needed to run a standard networked-control simulation. The user is asked to simply set the number of sensors and actuators that are desired to be in the network and a name for the Simulink model to be generated.

Then the code generation creates a Simulink model that contain two submodels, Plant and Controller, that can be customized by the user according to personal needing.

Class costructor

USAGE

acg_obj=ACG(Ns,Na,name)

- Ns = number of sensors in the network (Optional: default value=1);
- Na = number of actuators in the network (Optional: default value=1);
- name = name of the simulink model file (Optional: default value='acg_default');

Code Generation Method

Generated the files required for performing a standard network-aware simulation using TrueTime

SensorsR init

SensS

SensR

msgRcvActR

Actuator Code

ActS init

ActS

ActR

msgRcvActR

Configure the new system

Mux

Controller plant, Wireless

Add sensors blocks

Receiver

Sender

External Wireless

Internal Wireless

Controller and Plant

Rename actuator X1 with real number

Sender

External Wireless

Internal Wireless

Controller & plant

Actuators

Receiver

Sender

External Wireless

Internal Wireless

Controller and Plant

Connected previously added blocks

Remove generated code

Remove all generated code so as to clean the folder and permit new generation;

1.4 decLMI

This class provides a tool to synthesize a stabilizing decentralized linear controller for discrete-time LTI systems. Functionalities are also given to achieve either robust or stochastic convergence to the origin.

Contents

Class description

This work is based on the paper "Synthesis of networked switching linear decentralized controllers" by D. Barcelli, D. Bernardini and A. Bemporad, 49th IEEE Conference on Decision and Control, 2010.

The inputs for the class are

- the network state, i.e. the connection matrix of each node to each actuator;
- the matrices A and B modeling the controlled plant dynamics;
- the weights for the Riccati equation;
- the state and input constraints;
- the polytope mapping the initial state uncertainty (given as a set of vertices).
- the Markov chain modeling packet dropouts (needed for stochastic stability only)

All the methods solve an SDP problem and provide a feedback matrix gain K, and the matrix solution of the Riccati equation P. The SDP problems are dependent on the type of stability required. These methods are:

- Centralized: the network is assumed fully connected, all links are reliable and faultless (i.e., no packet dropouts are considered);
- Decentralized Ideal: the network is only partially connected, all links are reliable and faultless;
- Decentralized Robust: the network is only partially connected, and some links are faulty, i.e., packets transmitted in those links can be lost. This method provides a decentralized controller that guarantees stability for any possible configuration of packet drops, while robustly fulfilling state and input constraints;

- Decentralized Stochastic: the network is only partially connected, and some links are faulty, i.e., packets transmitted in those links can be lost. Dropouts are modeled by a Markov chain and closed-loop stability is guaranteed in the mean square sense. This solution is intended to be less conservative with respect to the robust one.

Output (read-only) properties

`Mc` is a structure which defines the Markov chain. It includes:

- T : transition matrix;
- E : emission matrix.

The other properties are the output of the SDP solution:

- P : solution of Riccati equation;
- K : feedback matrix gain (i.e. u=K*x);
- g : γ in $P = \gamma Q^{-1}$ that is the minimized parameter in the SDP optimization;

Each of the previous properties is a structure with fields:

- ci : centralized ideal;
- di : decentralized ideal
- dl : decentralized lossy, i.e. robust;
- ds : decentralized stochastic. that are computed by corresponding methods.

Private properties

Class constructor

obj=decLMI(Net,A,B,Qx,Qu,X0,xmax,umax,Mc)

`Net` : matrix of row size equal to number of actuators and columns size equal to number of states. `Net(i,j)` can be:

- 1 : if the state j is connected to actuator i by an ideal link;
- 0 : if there is no link between state j and actuator i;
- -1: if the state j is connected to actuator i by a lossy link;

`A` : state matrix of the LTI system modeling the plant;

`B` : input matrix of the LTI system modeling the plant;

`Qx` : state weight matrix of the Riccati equation;

`Qu` : input weight matrix of the Riccati equation;

`X0` : set of vertices of the polytope that defines the uncertainty of the initial state condition;

`xmax` : Euclidean norm constraint on state;

`umax` : Euclidean norm constraint on input;

`Mc` : two-states Markov chain that models the probability of losing a packet. Must be a structure with fields:

- d : array, where d(i) is the probability of losing a packet being in the i-th state of the Markov chain;
- q : array, where q(i) is the probability of remaining in the i-th state of the Markov chain.

Computation of all possible Network configurations for all possible packet-loss configuration

Number of -1 in Net(i,:)

Total no. of -1 in Net

Centralized LMI computation

Computes the solution of the SDP problem assuming the network to be fully connected and each link completely reliable. The goal is to give a reference for the performances that can be achieved via the decentralized methods.

Formulate and solve an SDP for a centralized problem with no packet loss

Define SDP variables

Positive definiteness of Q

Stability constraint

Input constraints

State constraints

Ellipsoid definition for every vertex of X0

Decentralized Ideal computation

Computes the solution considering only present links, but assuming that all of them are reliable. Basically does not account for miss-reception of packets containing mesurements.

Formulate and solve an SDP for a decentralization with no packet loss

Definition of zero components in K K(i,j) = 0 if state j is not available to compute input i

Definition of the structure of Y

Definition of the variable Y

Definition of the structure of Q

Definition of the variable Q

positive definiteness of Q

Stability constraint

Input constraints

State constraints

Ellipsoid definition for every vertex of X0

Decentralized Robust computation

Computes a decentralized controller which provides stability and constraints satisfaction for any possible occurrence of the packet dropouts.

Formulate and solve an SDP for a decentralization with packet-loss

In case there is no wireless links in a row, a fixed structure have to be applied: K0fix. It should not be necessary to allocate all K0fix, but int his way it is useful to manage indices

There is no -1 in this line

Set cooherently line in Kofix

Define a memory structure that contains all possible configurations for each line of Net and its implications in K0{i,j}, where i means the i-th line of K0 and j means one of the 2^nLi{i} possible configurations

In order to exploit the exponetial structure, the set of all possible configurations is generated using some recursive function that are not methods of the class.

Positive definiteness of Q

Ellipsoid definition for every vertex of X0

For all posssible configurations

Stability constraint

Input constraints

State constraints

Decentralized Stochastic computation

Computes a decentralized controller that exploits the available knowledge on the dropouts probabiliy distributions. A two-states Markov chain is used to model the packet losses. Mean-square stability is guaranteed.

Formulate and solve an SDP for a decentralization with packet loss and convergence in mean-square

In case there are no wireless links in a row, a fixed structure have to be applied: K0fix.

There is no -1 in this line

Set cooherently Kofix's line

Define a memory structure that contains all possible configurations for each line of Net and its implications in K0{i,j}, where i means the i-th line of K0 and j means one of the 2^nLi{i} possible configurations

For all possible states of the Markov chain

For all possible network configurations in that Markov chain state

Total number of lossy links

Positive definiteness of Q in state 1

Positive definiteness of Q in state 2

Ellipsoid definition for every vertex of X0

Ellipsoid definition for every vertex of X0

Define the constraint as [Qj [CC' CC1'];[CC;CC1] DD]

Implement hard constraints

Private methods

Not described below since their use is depicted in the code settion where the method is called.

Static Methods

1.5 dlincon

The class Decentralized LINear CONstrained (dlincon) provide the used with an extension of the object lincon, which is available with the Hybrid toolbox (by A. Bemporad) toward decentralized control.

Contents

Class description

Starting with an user defined decentralization, the toolbox automatically creates the corresponding set of decentralized lincon controllers of appropriate dimension. The user is requested to provide a starting configuration for the centralized controller (using the same parameters as in in lincon) and

that will be readapted, in proper dimension, to each subcontroller. Moreover each subcontroller can be customized according to needing as it is a lincon class instance.

Methods are available for three kinds of control action computation:

- `global` : centralized controller;
- `Dglobal`: run all DMPCs and assemble the components to return the complete set of inputs;
- `i` : run the `i`-th controller and return its results only;

Both regulator and tracking modes are supported, however the latter is realized by means of coordinates shift, accordingly to "Barcelli and Bemporad 'Decentralized Model Predictive Control of Dynamically-Coupled Linear Systems: Tracking under Packet Loss'". Update covering the normal tracking mode is matter of actual development.

Properties

`decent` : Decentralization structure: is supposed to an array of structure containing fileds:

- x (states of the subsystem);
- y (outputs of the subsystem);
- u (inputs of the subsystem);
- applied (inputs effectively applied, no overlap is allowed). each containing the array of indices to be included in the subsystem.
- Ts (subsystem sample times); [optional: if not specified all subsystem have same sample tie of the plant]

`M` : number of subsystem

`W` : selection matrix for states

`Z` : selection matrix for inputs

`G` : selection matrix for outputs

`APP` : selection matrix for applied inputs

`Zr` : state coordinate shift

`Vr` : input coordinate shift

`sub_sys` : cell array of subcontrollers models

`ccon` : centralized controller

`dcon` : cell array of lincon objects

`var_bounds` : 1 if bounds are specified online by the user, 0 else

`type` : 'reg' or 'track'

Class constructor

dl = dlincon(model,type,cost,interval,limits,yzerocon,... decent,varbounds);

- `model` : centralized LTI model of the plant
- `type` : 'reg' for regulator or 'track' for tracking
- `cost` : structure with state and input matrices weights (see lincon for more details);
- `interval` : structure with fields N and Nu, prediction and simulation horizons respectively (see lincon for more details);
- `limits` : structure with bounds on states and inputs (see lincon for more details);
- `yzerocon` : if 1 constraints are also enforced at the starting instant (see lincon for more details);
- `decent` : decentralization structure, see Properties for a more detailed help;
- `varbounds` : if 1 state are added to handle variant bounds specified by the user online to be enforced.
- `ctrlTs` : controller sample time (optional)

Control computation method

u = dl.Deval(type,xk,r) or u = Deval(dl,type,xk,r) where dl is a dlincon obj

- `type`: `global` : centralized controller; `Dglobal`: run all DMPCs and assemble the components to output the complete set of inputs; `i` : run the `i`-th controller and outputs its results only;
- `xk` : state measurements of estimations (must be coherent with state dimension
- `r` : vector of references for the outputs, ignored if type='reg'.

Obtain i-th subsystem state

Compute i-th sbsystem's inputs

Apply only the ones which deserve to be

Controller to be evaluated

Obtain i-th subsystem state

Compute the inputs of the i-th subsystem

Apply only the ones which deserve to be

Stability test around the origin

stability_test(dl,range,cost)

Test described in Barcelli and Bemporad, NECSYS09, returning true or false.

Create explicit controllers ranges

Create explicit controllers

Computational optimization

Check the region

We consider the stability test in an area around the origin for which the controllers are linear (not affine -> Gs=0)

Our method stability test

Lyapunov equation solution computation

Global optimal input

Local optimal inputs

Maximum length of missing packet sample times

Compute sum(WWQWW)

Positive definitiveness of result is to be checked

Since the reference may be unknown at this stage, the offset cannot be computed

Cost

Limits

Only those input that will really be applied need to be constrained

Add constant states umin,umax

Set y2=umin(t)-u(t), y3=umax(t)-u(t)

Limit

Check existence of related fields

Change ymin and ymax if required

Output weight

Input increment weight

Output weight

Input increment weight

Hard constraints

Input constraints horizon k=0,...,Ncu

Output constraints horizon k=1,...,Ncy

Check also output constraints for k=0

Add constant states umin,umax

Set y2=umin(t)-u(t), y3=umax(t)-u(t)

Since the limits will be given online, the fixed limits should be removed

Change ymin and ymax if required

Output weight

Input increment weight

Input constraints horizon k=0,...,Ncu

Output constraints horizon k=1,...,Ncy

Check also output constraints for k=0

All elements in decent have to be >0 and < number of corresponding elements in system (because they are indices)

Flag to determine if decentralization is wrong

Indices have to array columns

Indices have to >0 and <= #corresponding element

Bounds references

1.6 eampc

This class provides an implementation of an explicit MPC controller, where communications between controller and sensor nodes are subject to an energy-aware policy intended to lower the number of transmissions and, utlimately, save sensor nodes battery.

by D. Bernardini, 2010.

Contents

Class description

This class is based on the papers:

- D. Bernardini and A. Bemporad, "Energy-aware robust model predictive control based on wireless sensor feedback," in Proc. 47th IEEE Conf. on Decision and Control, Cancun, Mexico, 2008, pp. 3342–3347.
- D. Bernardini and A. Bemporad, "Energy-aware robust model predictive control with feedback from multiple noisy wireless sensors," 10th European Control Conference, Budapest, Hungary, 2009, pp. 4308–4313.

The input arguments to create an EAMPC object are:

- the A and B matrices of the controlled plant model, which is a discrete-time LTI system.
- the WSN configuration, namely the number of sensor nodes to use, the thresholds for the transmission policy, and the length of the state predictions buffer.
- the desired constraints on inputs and outputs.
- the weight matrices and other parameters which define the MPC objective function.

The overall control system is assumed to have the following configuration. A number of wireless sensor nodes collect noisy state measurements for disturbance rejection, compute an estimated state value, and transmi this estimation to the controller according to a transmission policy. This policy is based on a threshold logic. Namely, at every time step the estimated state value `Ymean` is transmitted to the controller if and only if there exists `i` such that

$|$`Ymean(i) - Xhat(i)`$| \geq$ `th(i)`

where `th` is a vector of user-defined thresholds, and `Xhat` is the predicted states buffer, which is precomputed by the controller and transmitted beforehand to the sensor nodes. Updated state predictions are computed and transmitted to the sensors every time new measurements are received by the controller. It is assumed that every sensor node collects a (noisy) measurement of the complete state vector, and that the energy cost of a (short-range) transmission between sensors is negligible with respect to a (long-range) transmission between sensors and controller.

Public properties

`Net` : specifies the network configuration. It is a structure with fields

- `nodes` : number of wireless sensor nodes used.
- `th` : threshold vector for measurement transmission policy. **Note:** if `th` is a vector with all zero entries, then a standard MPC control law is implemented, where at every time step measurements are sent to controller regardless to the threshold-based transmission policy.
- `Ne` : estimation horizon for state predictions computation.

Output (read-only) properties

`nu` : number of inputs of the controlled system.

`ny` : number of outputs of the controlled system.

`Plant` : controlled plant, modeled as a discrete-time LTI system. It is a structure with fields

- `A` : state matrix (`ny` x `ny`).
- `B` : input matrix (`ny` x `nu`).

`Net` : network configuration. It is a structure with fields

- `nodes` : number of wireless sensor nodes used.
- `th` : threshold vector for measurement transmission policy. **Note:** if `th` is a vector with all zero entries, then a standard MPC control law is implemented, where at every time step measurements are sent to controller regardless to the threshold-based transmission policy.
- `Ne` : estimation horizon for state predictions computation.

`Limits` : element-wise constraints on outputs and inputs. It is a structure with fields

- `ymin` : lower bounds on output (`ny` x 1).
- `ymax` : upper bounds on output (`ny` x 1).
- `umin` : lower bounds on input (`nu` x 1).
- `umax` : upper bounds on input (`nu` x 1).
- `dumin` : lower bounds on input rate (`nu` x 1). Optional.
- `dumax` : upper bounds on input rate (`nu` x 1). Optional.

`Weights` : weight matrices to be used in the MPC objective function. It is a structure with fields

- `Qu` : input weight (`nu` x `nu`).
- `Qy` : output weight (`ny` x `ny`).
- `Qn` : terminal state weight (`ny` x `ny`). Optional.
- `rho` : positive weight for soft output constraints (1 x 1). Optional.

`Params` : other parameters for the MPC problem design. It is a structure with fields

- `pnorm` : norm used in the MPC objective function.
- `N` : prediction horizon.
- `Nc` : control horizon. Optional.

`Ctrl` : explicit MPC controller obtained with MPT Toolbox. See mpt_control for more details.

`Sim` : Data used for simulations. It is a structure with fields

- `Y` : set of current measurements.
- `Ymean` : estimated output value, taken as the mean value of Y.
- `Xhat` : buffer of predicted state values.
- `tx` : records transmissions from sensor nodes to controller.
- `rx` : records transmissions from controller to sensor nodes.

Class constructor

`obj = eampc(Plant,Net,Limits,Weights,Params)`

`Plant` : controlled plant, modeled as a discrete-time LTI system. It is a structure with fields

- `A` : state matrix (`ny` x `ny`). Mandatory.
- `B` : input matrix (`ny` x `nu`). Mandatory.

`Limits` : element-wise constraints on outputs and inputs. It is a structure with fields

- `ymin` : lower bounds on output (`ny` x 1). Mandatory.
- `ymax` : upper bounds on output (`ny` x 1). Mandatory.
- `umin` : lower bounds on input (`nu` x 1). Mandatory.
- `umax` : upper bounds on input (`nu` x 1). Mandatory.
- `dumin` : lower bounds on input rate (`nu` x 1). Optional (default: `-Inf`).
- `dumax` : upper bounds on input rate (`nu` x 1). Optional (default: `Inf`).

`Weights` : weight matrices to be used in the MPC objective function. It is a structure with fields

- `Qu` : input weight (`nu` x `nu`). Mandatory.
- `Qy` : output weight (`ny` x `ny`). Mandatory.
- `Qn` : terminal state weight (`ny` x `ny`). Optional (default: `Qy`).
- `rho` : positive weight for soft output constraints (1 x 1). Optional (default: `Inf`).

`Params` : other parameters for the MPC problem design. It is a structure with fields

- `pnorm` : norm used in the MPC objective function. It can be 1, 2, `Inf`. Mandatory.
- `N` : prediction horizon. Mandatory.
- `Nc` : control horizon. Optional (default: `N`).

---

init_sim

Initializes EAMPC object for simulations. This method must be called before running a new simulation.

Usage:

`obj = init_sim(obj)`

where

`obj` : EAMPC object. Mandatory.

---

send_predictions

Checks if an updated prediction buffer needs to be provided to the sensor nodes. In this case, a sequence of `obj.Net.Ne` future state predictions are computed and transmitted to the sensor nodes. **Note:** if all the components of `obj.Net.th` are zeros, no transmission takes place.

Usage:

```
obj = send_predictions(obj,Xk,Uprev)
```

where

`obj` : EAMPC object. Mandatory.

`Xk` : current estimated state value. Mandatory.

`Uprev` : previous input value. Mandatory if input rate constraints are imposed, otherwise ignored.

---

get_measurements

Given the actual state `X` and the output noise `V`, provides to the controller an estimation of the state vector. Measurements from all sensor nodes are gathered (assuming all the state components are measured by each node) and the average output `Ymean` is computed. Then, `Ymean` is transmitted to the controller if and only if there exists `i` such that

$|$`Ymean(i) - Xhat(i)`$| \geq$ `th(i)`

Usage:

```
[Xestim,obj] = get_measurements(obj,X,V)
```

where

`obj` : EAMPC object. Mandatory.

`X` : state value (`ny` x 1). Mandatory.

`V` : output noise matrix (`ny` x no. of nodes). Mandatory.

`Xestim` : estimated state value.

---

get_input

Computes MPC control move given a controller and an initial state.

Usage:

```
Uopt = get_input(obj,X,Uprev)
```

where

`obj` : EAMPC object. Mandatory.

`X` : initial state. Mandatory.

`Uprev` : previous contrl move. Mandatory if input rate constraints are imposed, otherwise ignored.

---

build_mpc

Computes the explicit solution of the MPC problem:

$$J(x(0)) = \min_u \|Q_N x(N)\|_p + \sum_{j=0}^{N-1} \|Q_y y(j)\|_p + \|Q_u u(j)\|_p$$

$$\text{subject to } \; x(j) \in X, \; u(j) \in U$$

for every $x(0) \in X$. Assume full state-feedback.

Usage:

```
obj = build_MPC(obj)
```

where

`obj` : EAMPC object. Mandatory.

1.7 himpc

This class implements the hierarchical MPC control paradigm presented in `Barcelli, Bemporad, Ripaccioli, CDC10` and decentralized extension (`Barcelli, Bemporad, Ripaccioli, IFAC11`.

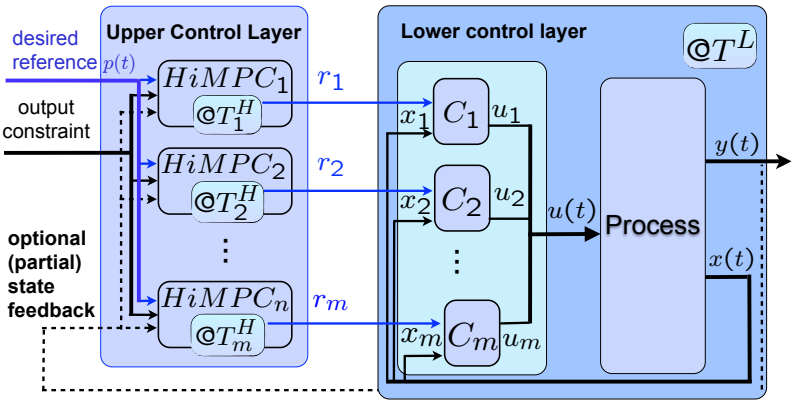
Contents

- Class description
- Properties
- Constructor
- MOARS determination for each subsystem
- Plot of all MOARS
- Compute DeltaR and DeltaR/N for each sub-model
- Plot DeltaR and DeltaR/N for each sub-model
- Static Methods

Class description

In this work we propose a decentralized hierarchical multi-rate control design approach to linear systems subject to linear constraints on input and output variables. At the lower level, a set of linear controllers stabilize the open-loop process without considering the constraints. A higher-layer, composed of a set of independent controllers, commands reference signals at a lower sampling frequency so as to enforce linear constraints on the variables of the process. By optimally constraining the magnitude and the rate of variation of the reference signals applied to the lower control layer, we provide quantitative criteria for selecting the ratio between the sampling rates of the upper and lower layers to preserve closed-loop stability without violating the prescribed constraints.

The HiMPC class has been developed to automatically generate the proposed multi-layer control architecture, which structure is shown in the following figure.



The layout of the controller is composed by two levels in a hierarchical structure which, in turn, are composed by a set of independent and decentralized sub-controllers. We assume that the stability of the process is guaranteed by the action of the lower control layer, which runs at the same sample time of the plant. Therefore is required that the LTI model of the plant in closed-loop with the set of lower layer controllers, given by the user, is stable. Future update to the toolbox will allow the user to automatically generate the stabilizing lower layer given the plant model and the desired decentralization, for example though LMI (See corresponding section of the toolbox). The decentralization of the inner loop controller, i.e. the sets of input, states and outputs assigned to each sub-controllers is one of the strenghts of the proposed approach. The decentralized design has been exploited together with the LTI model of the plant and the constraints on inputs and states to compute the polytopic sets of each submodel which is the bound the mutual influence of the different subsystems on each other. This result is fundamental because allows to treat each subsystem independently, since for all of them a single Maximal Output Admissible Set (MOAS) is computed [1]. The independece of the MOASs is guaranteed by taking into account the uncertainty polytope, which bounds the interaction with the others, in the invariant set computation.

Moreover, the MOAS determination is related to the restriction of the admissible output set, which is a polytope tightening. The contraction factor, Δk, is the main tuning knob of the approach: the smaller the components of Δy, the larger is the set of admissible set points, but the smaller will be the admissible reference increments to maintain tracking errors within the admissible error set.

Such control architecture is then extended to allow a decentralized superviros which is then capable of speedup the sampling frequencies in those spatial regiogion presenting less strict challanges. Cooperativeness is avoided to guarantee complete independece, a clear advantage from communi-

cation and parallelism viewpoints. So as to achieve this, inter-subsystem interractions are treated as disturbances, leading to consider Maximal Output Admissible Robust Set (MOARS) instead of MOAS.

Then for each subsystem, the maximum element-wise reference variation such that for any admissible state and interaction, the system state will be in a MOAS at the next execution of higher level controller. It is evident that such variation is a function of the ratio between the two layers sample time. The constraint computed in that way bounds the reference variation that is possible to give to the sub-controllers, while preserving the closed-loop stability and enforcing the constraints. Such task is performed by the higher level controllers.

Finally the maximum element-wise reference variation for each subsystem is computed. The maximum element-wise reference variation is defined as the smallest change of reference vector that can be applied to the closed-loop system of the lower layer controllers with the plant such that, starting from an invariant set, the state vector lands outside a new invariant set a given number of steps. Or, in other words, for all reference changes the closed-loop system is such that, starting from an invariant set, the state vector always arrives into a new invariant set after given number of steps.

[1] Kolmanovsky, I. and Gilbert, E.G., *Maximal output admissible sets for discrete-time systems with disturbance inputs*, 14th American Control Conference 1995, Seattle, WA

Properties

`DeltaK` : Array of the same size of the constraints that determines their tightening

`Xcon` : Structure of constraints with fields

- min;
- max;

`dec` : Decentralization structure, is a cell array of structures each of which with fileds:

- x (states of the subsystem);
- y (outputs of the subsystem);
- u (inputs of the subsystem);
- applied (inputs effectively applied, no overlap is allowed). i-th structure contains the indices of the respective elements that are currently included in the i-th subsystem

`model` : ss object with the plant LTI model

`comp` : Cell array of the state indices that the do not belong to the i-th subsystem

`coupledCons`: Cell array of non element-wise state constraints with fields H and K to be appended to the constraint polytope of the i-th subsystem

---

Constructor

obj = HiMPC(model,dec,Ycon,DeltaK,Xcon,coupledCons)

`model` : ss object with the plant LTI model

`dec` : Decentralization structure: is supposed to an array of structure containing fileds:

- x (states of the subsystem);
- y (outputs of the subsystem);
- u (inputs of the subsystem);
- applied (inputs effectively applied, no overlap is allowed). each containing the array of indices to be included in the subsystem.

`DeltaK` : Array of the same size of the constraints that determines their tightening

`Xcon` : state bounds structure eight fields min and max

`coupledCons`: Cell array of non element-wise state constraints with fields H and K to be appended to the constraint polytope of the corresponding subsystem

---

MOARS determination for each subsystem

Compute the MOARS of all the submodels

Optinal parameter, the ray of the unit ball used to avoid empty polytopes (deault 1e-6)

Computes the MOARS

Inv set inequatily description

Plot of all MOARS

Plot a figure with as many subfigures as the submodels each showing in red the MOAS with no disturbance and in blue the real MOARS.

Compute DeltaR and DeltaR/N for each sub-model

Computes the maximum reference variation of each sub-model as a function of the sample time ratio, also with sample time normalization. No parameter is required.

For each sub-model

Variable declaration

Continuos

Binary

DC gain

Constraints

Epsilon = max(r1-r2)

x(0) in omega(r1)

xr2

Big M

xN is the state at next sample time of the hierarchical, that is after Nh samples of the fast model

xN NOT in omega(r2)

big M

One constraint must be violated in xN

r1,r2 in (Ho,Ko) polytope

each disturbance realization in Pnoise

Solver setup

Plot DeltaR and DeltaR/N for each sub-model

Plot both $\Delta r(N)$ and the ratio $\Delta r(N)/N$ over the sample times ratio N.

Static Methods

1.8 ncs

- Constructor Function
- Class Functions
- Restrictions on setting the variable type

Constructor Function

Class Functions

Restrictions on setting the variable type

Chapter 2

Examples

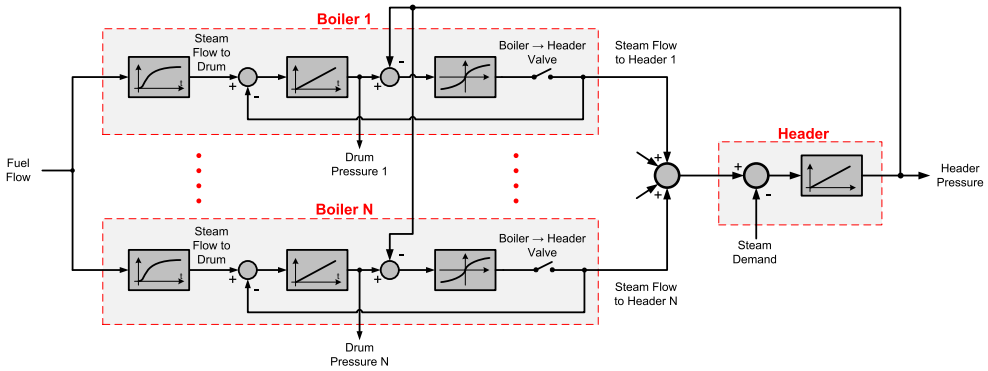
2.1 LSmodel

Demonstration of Large Scale Model Class `LSmodel`
This script demonstrates the use of `LSmodel` class for modeling and model management of Large Scale (LS) systems. The first part shows definition of `LSmodel` for multiple boilers connected to a single header. The second part is demonstration of `LSmodel` methods on randomly generated model with 15 submodels.

Contents

Submodels Definition

This section defines models of individual boilers and header. These models will be used to create LS model in the next section. The structure of the model is in following figure.




```
% --- Boiler parameters -------------------------------
n_boilers = 5;                % number of boilers (1-5)
T = [50 100 150 200 250];     % fuel -> steam time constants [s]
Ks = 10;                      % amount of steam [t/hrs] from unit of
                              % fuel [t/hrs]
V = [300 500 700 800 600];    % boiler volumes [m3] (1/V ~ pressure
                              % integration constant)
K = [100 130 150 130 140];    % boiler->header pipe "conductivity",
                              % =flow/(pressure difference) [t/hrs/bar]
Kh = 1;                       % "conductivity" to turbine
Vh = 2e3;                     % header volume [m3]

% --- Boiler models ----------------------------------
% Inputs:
%   FF ... fuel flow - same value flows to all boilers [t/hrs]
%   ph ... header pressure [MPa]
% -------------------------------------------------------
% States:
%   GS ... generated steam [t/hrs]
%   pb ... boiler pressure [MPa]
% -------------------------------------------------------
% Outputs:
%   SF ... overall steam flow from all boilers [t/hrs]
%   pb ... boiler pressure [MPa]
% -------------------------------------------------------
for i=1:n_boilers,
  A = [ -1/T(i) , 0
        1/V(i)  , -K(i)/V(i) ];
  B = [ Ks/T(i) , 0
        0       , K(i)/V(i) ];
  C = [ 0 , K(i)
        0 , 1    ];
  D = [ 0 , -K(i)
        0 , 0    ];
  M{i} = ss(A,B,C,D);           % models of individual boilers
  M{i}.InputName  = { 'FF' , 'ph' };
  M{i}.StateName  = { ['GS' num2str(i)] , ['pb' num2str(i)] };
  M{i}.OutputName = { ['SF' num2str(i)] , ['pb' num2str(i)] };
  %M{i}.Name = ['B' num2str(i) ];
  M{i}.Name = ['Boiler ' num2str(i) ];
```

```
end


% --- Header model ----------------------------------
% Inputs:
%   SF ... total steam flow from boilers [t/hrs]
%   SD ... steam demand from (from turbine) [t/hrs]
% -------------------------------------------------
% Outputs:
%   ph ... header pressure [MPa]
% -------------------------------------------------
A = -Kh/Vh;
B = [1/Vh -1/Vh];
C = 1;
D = 0;
M{n_boilers+1} = ss(A,B,C,D);
M{n_boilers+1}.InputName  = { 'SF' , 'SD' };
M{n_boilers+1}.StateName  = { 'ph' };
M{n_boilers+1}.OutputName = { 'ph' };
%M{n_boilers+1}.Name       = 'H';
M{n_boilers+1}.Name       = 'Header';

% --- Steam flow summing block -----------------------
% sums steam flows from boilers to header
sum1 = sumblk('SF','SF1','SF2','SF3','SF4','SF5');
%sum1.Name = 'SFsum';
sum1.Name = 'Steam Flow Summator';
```

Construction of LSmodel

LS model is contructed by specifying submodels, summators and a list of external inputs and outputs. Internal connections are automatically created by matching input/output names

```
mod = LSmodel(M,sum1,{'FF','SD'},{'ph','SF'})
```

```
Large scale model (total order=11, 6 subsystem(s), 1 summator(s)):
  --- Subsystems -----------
    M1: 2 inputs, 2 outputs, order=2, name="Boiler 1"
    M2: 2 inputs, 2 outputs, order=2, name="Boiler 2"
    M3: 2 inputs, 2 outputs, order=2, name="Boiler 3"
    M4: 2 inputs, 2 outputs, order=2, name="Boiler 4"
    M5: 2 inputs, 2 outputs, order=2, name="Boiler 5"
    M6: 2 inputs, 1 outputs, order=1, name="Header"
  --- Summators -----------------
    Sum1: 5 inputs, name = "Steam Flow Summator"
  --- External Inputs -----------
    IN01: FF               X
    IN02: SD               X
  --- External Outputs ----------
    OUT01: ph              X
    OUT02: SF              X
```

Constructor function is compatible with standard CONNECT function, where the same model would be defined as (connect does not accept cell array of models)

```
mod_connect = connect(M{1},M{2},M{3},M{4},M{5},M{6},sum1, ...
                      {'FF','SD'}, ...
                      {'ph','SF'});
```

It is also possible to construct the model by using numeric indexing of inputs and outputs. However, name based signal referencing will be prefered as numeric indexing may be confusing for LS systems.

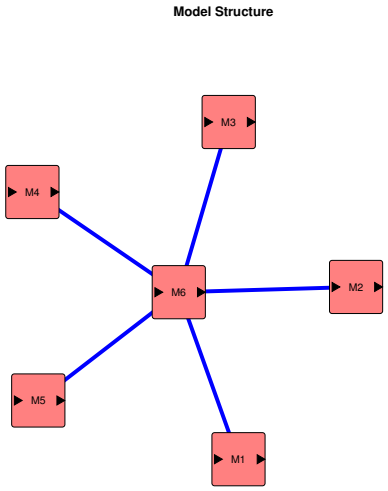
```
inputs = { [1 3 5] , 8 };   % extended notation of standard connect
Q = [ 2 7 0 0
      4 7 0 0
      6 7 0 0
      7 1 3 5 ];
outputs = { 7 , [1 3 5] };

inputs = { [1 3 5 7 9], 12 };
Q = [2  11 0  0  0  0
     4  11 0  0  0  0
     6  11 0  0  0  0
     8  11 0  0  0  0
     10 11 0  0  0  0
     11 1  3  5  7  9 ];
outputs = { 11 , [1 3 5 7 9] };

mod2 = LSmodel(M{1},M{2},M{3},M{4},M{5},M{6},Q,inputs,outputs);
```

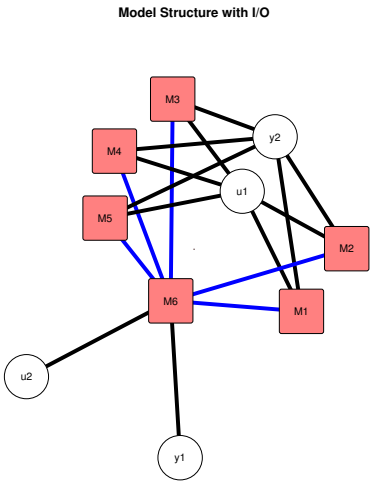
Plot model structure

Plots structure of large scale model as a graph, where vertexes are subsystems and edges indicate interaction between subsystems.

```
figure(10); clf; subfigure(10,3,3,[1 5]); % publish
subplot(121);
plot(mod);
title('\bfModel Structure');
subplot(122);
plot(mod,'io');
title('\bfModel Structure with I/O');
```

```
Computing vertex positions... Done.
Computing vertex positions... Done.
```





Random LS model with 15 sub-models

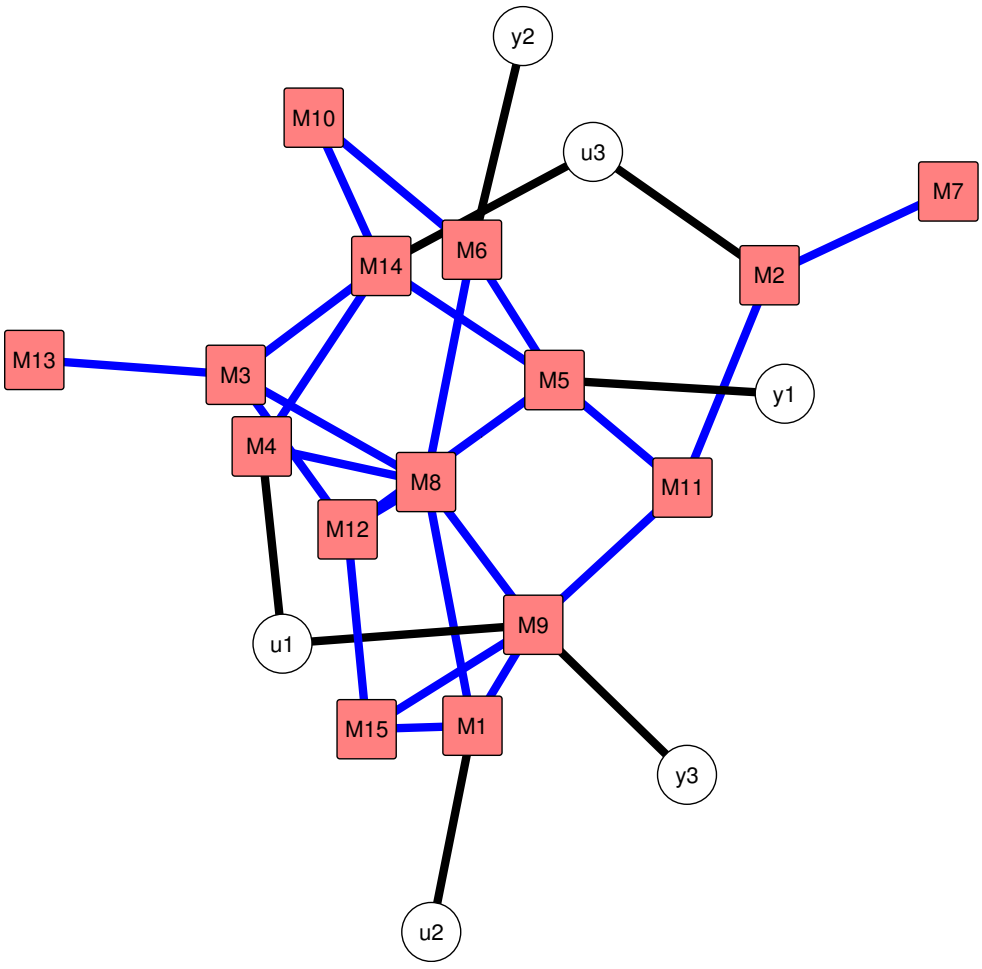
Previous model is rather small. Larger model will be randomly generated by using command `RLS` parameterized by number of sub-models, maximum sub-model order, number of external outputs and number of external inputs.

```
seed=963;rand('seed',seed);randn('seed',seed); % to get stable CL model
mod = rls(15,3,3,3)
clf;
plot(mod,'io');
title('\bfModel Structure');
```

```
Large scale model (total order=35, 15 subsystem(s), 8 summator(s)):
  --- Subsystems -----------
    M1: 1 inputs, 1 outputs, order=3, name="subs1"
    M2: 1 inputs, 1 outputs, order=3, name="subs2"
    M3: 1 inputs, 2 outputs, order=3, name="subs3"
    M4: 2 inputs, 2 outputs, order=1, name="subs4"
    M5: 2 inputs, 1 outputs, order=1, name="subs5"
    M6: 1 inputs, 1 outputs, order=2, name="subs6"
    M7: 1 inputs, 1 outputs, order=2, name="subs7"
    M8: 2 inputs, 2 outputs, order=2, name="subs8"
    M9: 2 inputs, 2 outputs, order=2, name="subs9"
    M10: 1 inputs, 1 outputs, order=3, name="subs10"
    M11: 2 inputs, 2 outputs, order=1, name="subs11"
    M12: 1 inputs, 2 outputs, order=3, name="subs12"
    M13: 1 inputs, 2 outputs, order=3, name="subs13"
    M14: 2 inputs, 2 outputs, order=3, name="subs14"
    M15: 1 inputs, 1 outputs, order=3, name="subs15"
  --- Summators -----------------
    Sum1: 3 inputs, name = "S1"
    Sum2: 4 inputs, name = "S2"
    Sum3: 3 inputs, name = "S3"
    Sum4: 5 inputs, name = "S4"
    Sum5: 3 inputs, name = "S5"
```

```
    Sum6: 3 inputs, name = "S6"
    Sum7: 5 inputs, name = "S7"
    Sum8: 5 inputs, name = "S8"
  --- External Inputs -----------
    IN01: s24               X
    IN02: s25               X
    IN03: s26               X
  --- External Outputs ----------
    OUT01: s7               X
    OUT02: s8               X
    OUT03: s13              X

Computing vertex positions... Done.
```

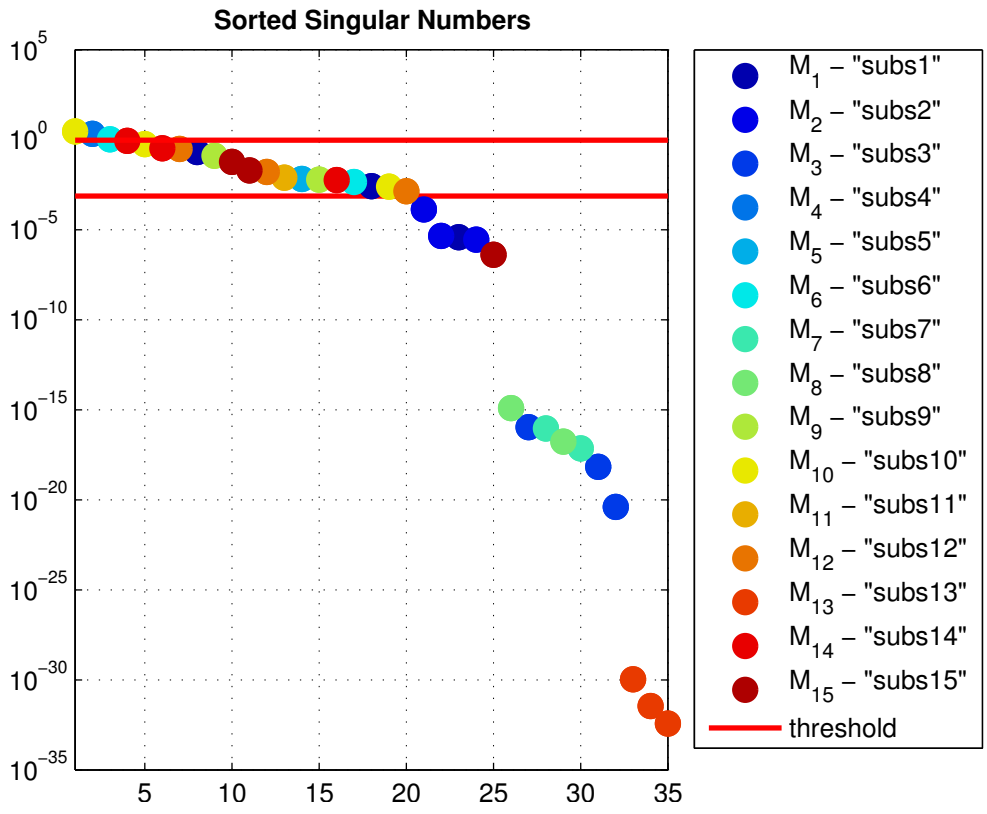
Model Structure

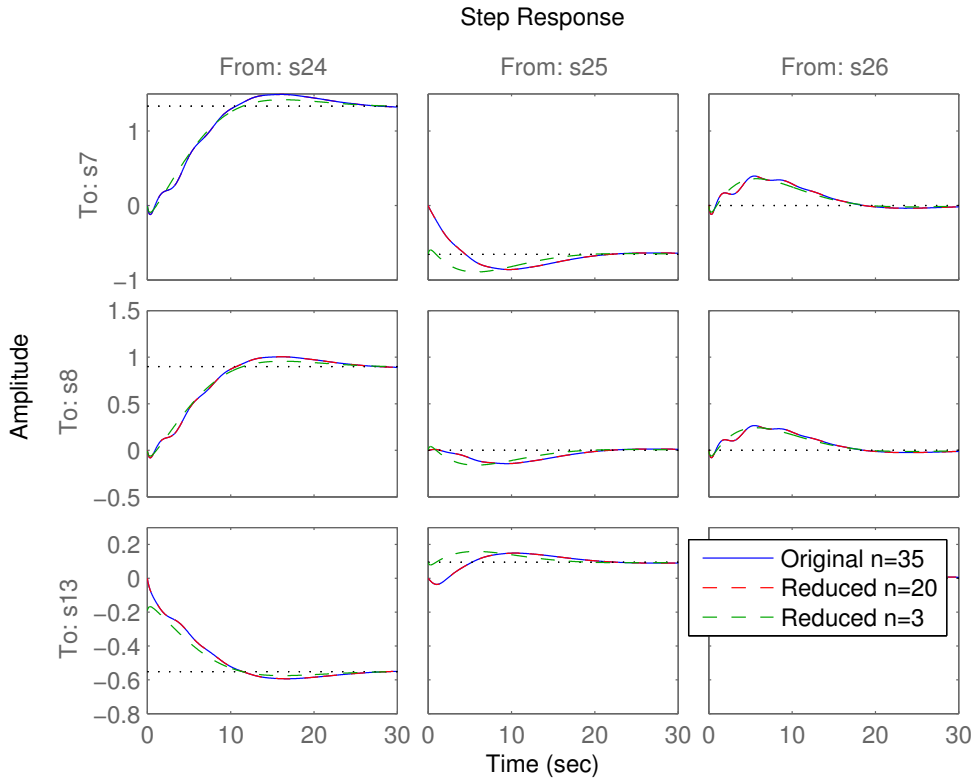


Structure preserving model order reduction

Standard model order reduction destroys model internal structure. Structure preserving model order reduction reduces order of individual sub-models to achieve minimum mismatch with original model from external input/output point of view. Structured singular numbers are plotted to show which sub-models are reduced in order to achieve target global model order.

```
figure(2);clf;
mod_r20=mod.struct_red(20);       % model reduced to 20th order
mod_r3=mod.struct_red(3);         % model reduced to 3rd order
% Compare step responses of original and reduced order models
figure(1);
clf;step(mod,30);hold all;
step(mod_r20,'r--',30);
step(mod_r3,'g--',30);
legend(['Original n=' num2str(mod.n)],...
       ['Reduced n=' num2str(mod_r20.n)],...
       ['Reduced n=' num2str(mod_r3.n)]);
```





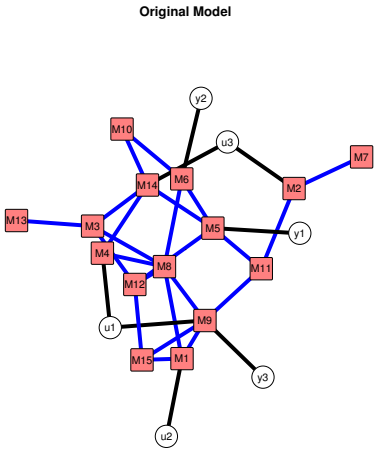
"On-line" Model changes

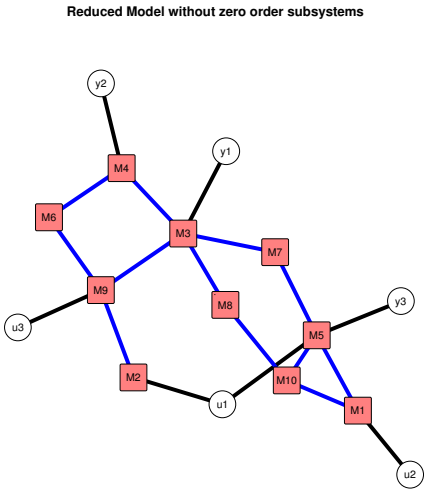
Removing submodel(s)

Remove subsystems, which have zero order after structured reduction (just for demonstration of large scale model manipulations).

```
ind = find(mod_r20.orders==0);
removed_models = mod.M(ind);
modX = mod.rem_mod(ind);
figure(10); clf;
subplot(121);plot(mod,'io');
title('\bfOriginal Model');
subplot(122);plot(modX,'io');
title('\bfReduced Model without zero order subsystems');
```

```
Computing vertex positions... Done.
Computing vertex positions... Done.
```

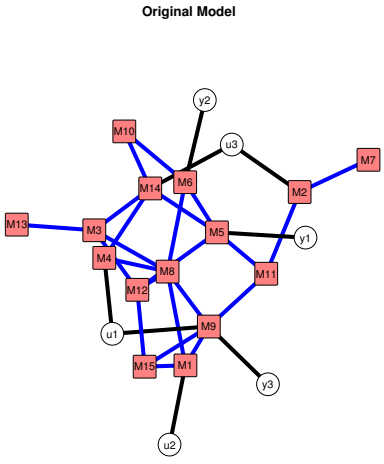


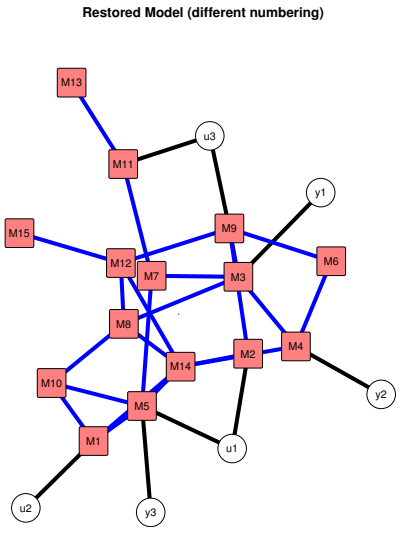


Adding submodel(s)

```
modX = modX.add_mod( removed_models );
cla; plot(modX,'io');
title('\bfRestored Model (different numbering)');
```

```
Computing vertex positions... Done.
```



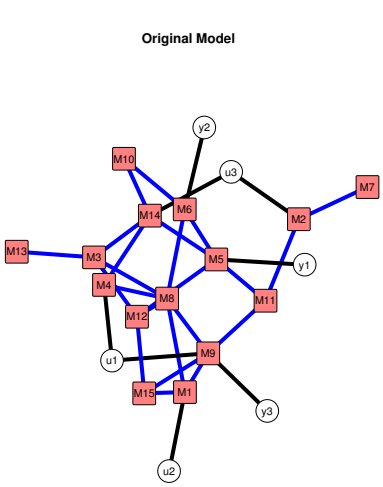


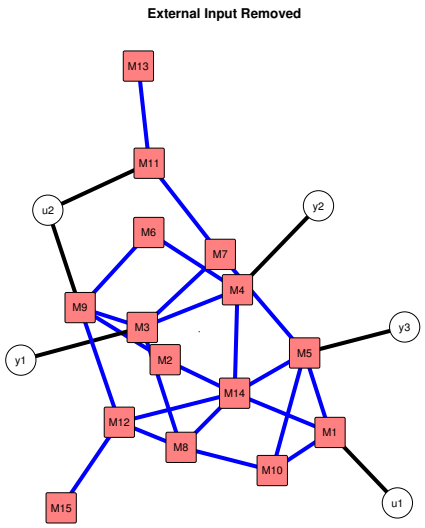
Removing external input

```
modX = modX.rem_ext_inp('s24')
cla; plot(modX,'io');
title('\bfExternal Input Removed');
```

```
Large scale model (total order=35, 15 subsystem(s), 8 summator(s)):
  --- Subsystems ----------
    M1: 1 inputs, 1 outputs, order=3, name="subs1"
```

```
    M2: 2 inputs, 2 outputs, order=1, name="subs4"
    M3: 2 inputs, 1 outputs, order=1, name="subs5"
    M4: 1 inputs, 1 outputs, order=2, name="subs6"
    M5: 2 inputs, 2 outputs, order=2, name="subs9"
    M6: 1 inputs, 1 outputs, order=3, name="subs10"
    M7: 2 inputs, 2 outputs, order=1, name="subs11"
    M8: 1 inputs, 2 outputs, order=3, name="subs12"
    M9: 2 inputs, 2 outputs, order=3, name="subs14"
    M10: 1 inputs, 1 outputs, order=3, name="subs15"
    M11: 1 inputs, 1 outputs, order=3, name="subs2"
    M12: 1 inputs, 2 outputs, order=3, name="subs3"
    M13: 1 inputs, 1 outputs, order=2, name="subs7"
    M14: 2 inputs, 2 outputs, order=2, name="subs8"
    M15: 1 inputs, 2 outputs, order=3, name="subs13"
  --- Summators -----------------
    Sum1: 3 inputs, name = "S1"
    Sum2: 4 inputs, name = "S2"
    Sum3: 3 inputs, name = "S3"
    Sum4: 5 inputs, name = "S4"
    Sum5: 3 inputs, name = "S5"
    Sum6: 3 inputs, name = "S6"
    Sum7: 5 inputs, name = "S7"
    Sum8: 5 inputs, name = "S8"
  --- External Inputs -----------
    IN01: s25              X
    IN02: s26              X
  --- External Outputs ----------
    OUT01: s7              X
    OUT02: s8              X
    OUT03: s13             X

Computing vertex positions... Done.
```

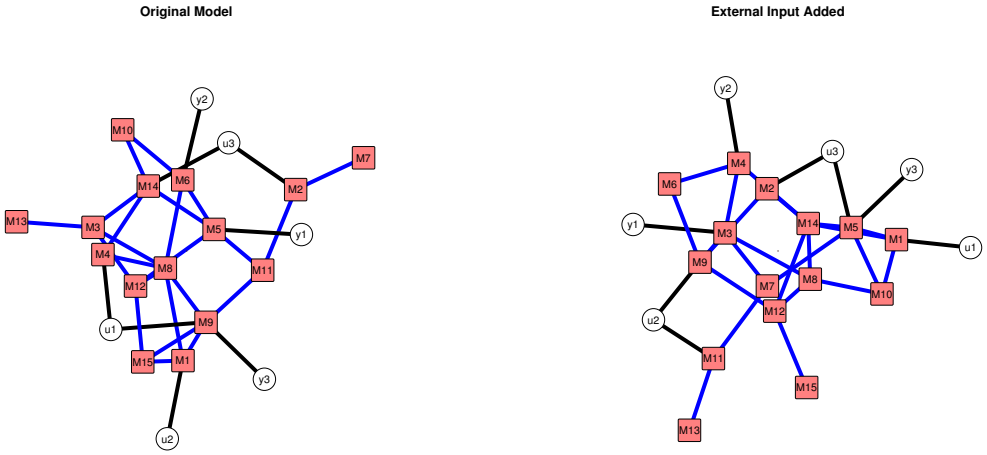




Adding external input

```
modX = modX.add_ext_inp('s24')
cla; plot(modX,'io');
title('\bfExternal Input Added');
```

```
Large scale model (total order=35, 15 subsystem(s), 8 summator(s)):
  --- Subsystems -----------
    M1: 1 inputs, 1 outputs, order=3, name="subs1"
    M2: 2 inputs, 2 outputs, order=1, name="subs4"
    M3: 2 inputs, 1 outputs, order=1, name="subs5"
    M4: 1 inputs, 1 outputs, order=2, name="subs6"
    M5: 2 inputs, 2 outputs, order=2, name="subs9"
    M6: 1 inputs, 1 outputs, order=3, name="subs10"
    M7: 2 inputs, 2 outputs, order=1, name="subs11"
    M8: 1 inputs, 2 outputs, order=3, name="subs12"
    M9: 2 inputs, 2 outputs, order=3, name="subs14"
    M10: 1 inputs, 1 outputs, order=3, name="subs15"
    M11: 1 inputs, 1 outputs, order=3, name="subs2"
    M12: 1 inputs, 2 outputs, order=3, name="subs3"
    M13: 1 inputs, 1 outputs, order=2, name="subs7"
    M14: 2 inputs, 2 outputs, order=2, name="subs8"
    M15: 1 inputs, 2 outputs, order=3, name="subs13"
  --- Summators -----------------
    Sum1: 3 inputs, name = "S1"
    Sum2: 4 inputs, name = "S2"
    Sum3: 3 inputs, name = "S3"
    Sum4: 5 inputs, name = "S4"
    Sum5: 3 inputs, name = "S5"
    Sum6: 3 inputs, name = "S6"
    Sum7: 5 inputs, name = "S7"
    Sum8: 5 inputs, name = "S8"
  --- External Inputs -----------
    IN01: s25              X
    IN02: s26              X
    IN03: s24              X
  --- External Outputs ----------
    OUT01: s7              X
    OUT02: s8              X
    OUT03: s13             X

Computing vertex positions... Done.
```

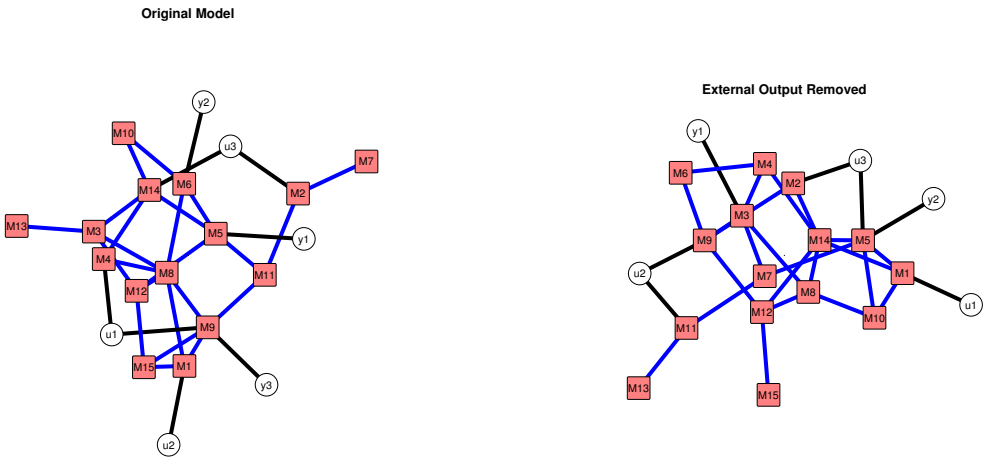


Removing external output

```
modX = modX.rem_ext_out('s8')
cla; plot(modX,'io');
title('\bfExternal Output Removed');
```

```
Large scale model (total order=35, 15 subsystem(s), 8 summator(s)):
  --- Subsystems -----------
    M1: 1 inputs, 1 outputs, order=3, name="subs1"
    M2: 2 inputs, 2 outputs, order=1, name="subs4"
    M3: 2 inputs, 1 outputs, order=1, name="subs5"
    M4: 1 inputs, 1 outputs, order=2, name="subs6"
    M5: 2 inputs, 2 outputs, order=2, name="subs9"
    M6: 1 inputs, 1 outputs, order=3, name="subs10"
    M7: 2 inputs, 2 outputs, order=1, name="subs11"
    M8: 1 inputs, 2 outputs, order=3, name="subs12"
    M9: 2 inputs, 2 outputs, order=3, name="subs14"
    M10: 1 inputs, 1 outputs, order=3, name="subs15"
    M11: 1 inputs, 1 outputs, order=3, name="subs2"
    M12: 1 inputs, 2 outputs, order=3, name="subs3"
    M13: 1 inputs, 1 outputs, order=2, name="subs7"
    M14: 2 inputs, 2 outputs, order=2, name="subs8"
    M15: 1 inputs, 2 outputs, order=3, name="subs13"
  --- Summators -----------------
    Sum1: 3 inputs, name = "S1"
    Sum2: 4 inputs, name = "S2"
    Sum3: 3 inputs, name = "S3"
    Sum4: 5 inputs, name = "S4"
    Sum5: 3 inputs, name = "S5"
    Sum6: 3 inputs, name = "S6"
    Sum7: 5 inputs, name = "S7"
    Sum8: 5 inputs, name = "S8"
  --- External Inputs -----------
    IN01: s25              X
    IN02: s26              X
    IN03: s24              X
  --- External Outputs ----------
    OUT01: s7              X
```

```
    OUT02: s13              X

Computing vertex positions... Done.
```

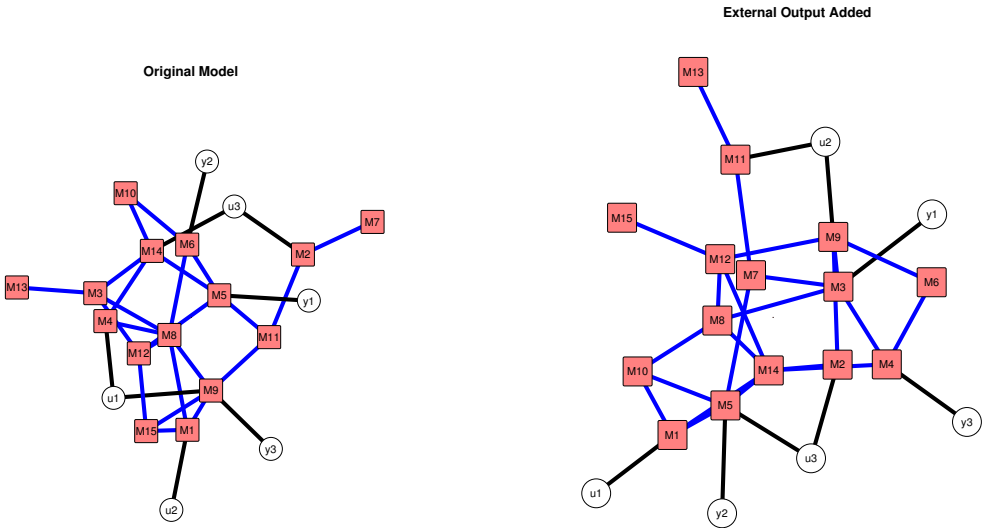


Adding external output

```
modX = modX.add_ext_out('s8')
cla; plot(modX,'io');
title('\bfExternal Output Added');
```

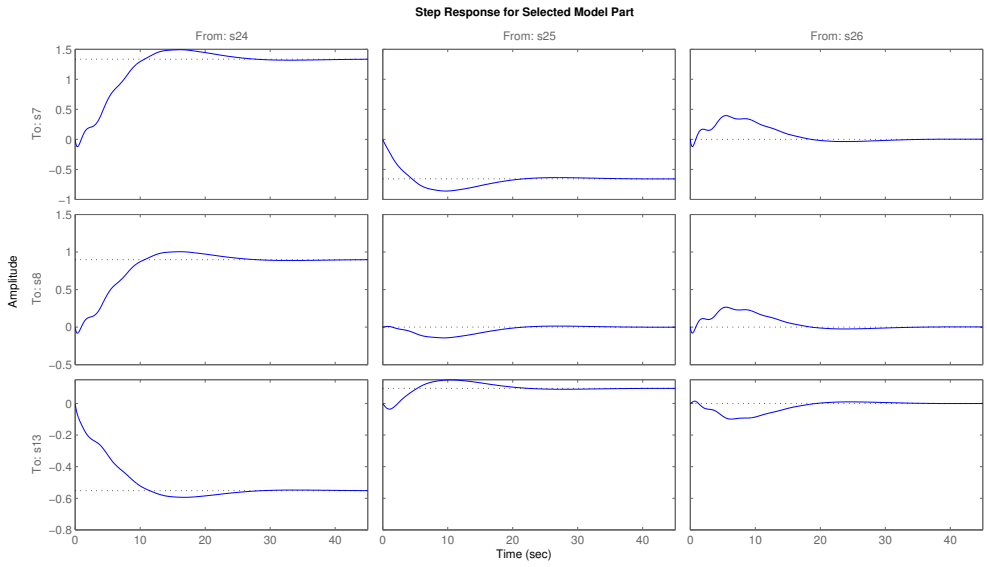
```
Large scale model (total order=35, 15 subsystem(s), 8 summator(s)):
  --- Subsystems -----------
    M1: 1 inputs, 1 outputs, order=3, name="subs1"
    M2: 2 inputs, 2 outputs, order=1, name="subs4"
    M3: 2 inputs, 1 outputs, order=1, name="subs5"
    M4: 1 inputs, 1 outputs, order=2, name="subs6"
    M5: 2 inputs, 2 outputs, order=2, name="subs9"
    M6: 1 inputs, 1 outputs, order=3, name="subs10"
    M7: 2 inputs, 2 outputs, order=1, name="subs11"
    M8: 1 inputs, 2 outputs, order=3, name="subs12"
    M9: 2 inputs, 2 outputs, order=3, name="subs14"
    M10: 1 inputs, 1 outputs, order=3, name="subs15"
    M11: 1 inputs, 1 outputs, order=3, name="subs2"
    M12: 1 inputs, 2 outputs, order=3, name="subs3"
    M13: 1 inputs, 1 outputs, order=2, name="subs7"
    M14: 2 inputs, 2 outputs, order=2, name="subs8"
    M15: 1 inputs, 2 outputs, order=3, name="subs13"
  --- Summators ----------------
    Sum1: 3 inputs, name = "S1"
    Sum2: 4 inputs, name = "S2"
    Sum3: 3 inputs, name = "S3"
    Sum4: 5 inputs, name = "S4"
    Sum5: 3 inputs, name = "S5"
    Sum6: 3 inputs, name = "S6"
    Sum7: 5 inputs, name = "S7"
    Sum8: 5 inputs, name = "S8"
  --- External Inputs -----------
```

```
    IN01: s25              X
    IN02: s26              X
    IN03: s24              X
  --- External Outputs ----------
    OUT01: s7              X
    OUT02: s13             X
    OUT03: s8              X

Computing vertex positions... Done.
```



Extraction of model part - eliminates submodels which are not controlable and observable for selected inputs and outputs.

```
ex_mod = mod.select('s24','s7');
clf;step(ex_mod);
title('\bfStep Response for Selected Model Part');
```

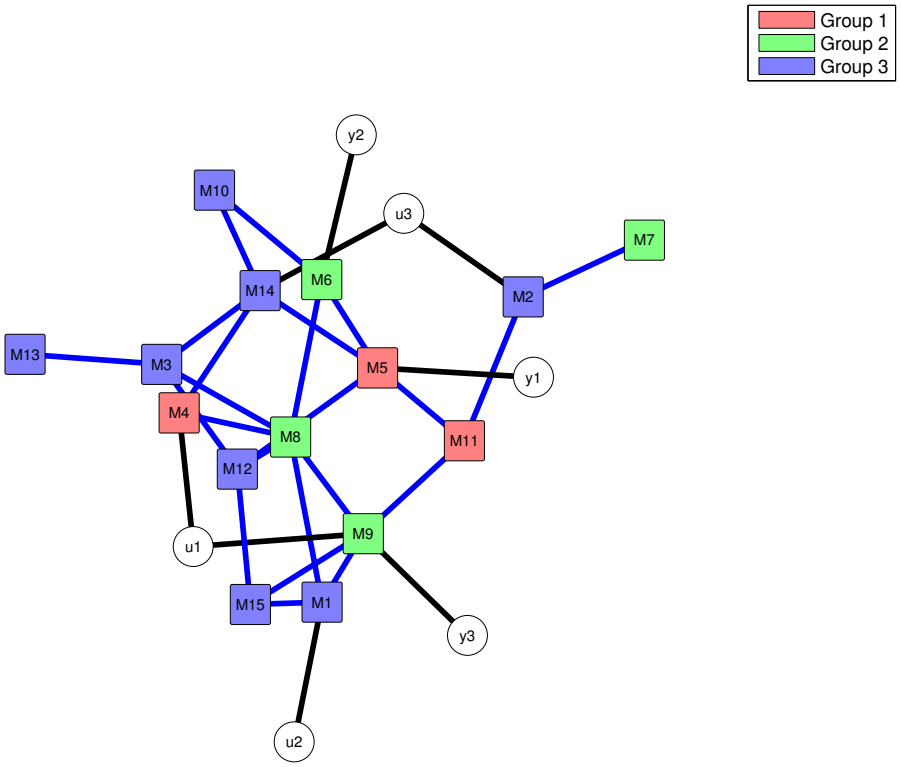


Submodels grouping

Each sub-model can be assigned to a group. Each group has its own number and sub-models are assigned to given groups in property GROUPING. Following command combines sub-models to groups according to their order.

```
mod.grouping = mod.orders;
clf; plot(mod,'io');
```

```
Computing vertex positions... Done.
```



Time domain models merging

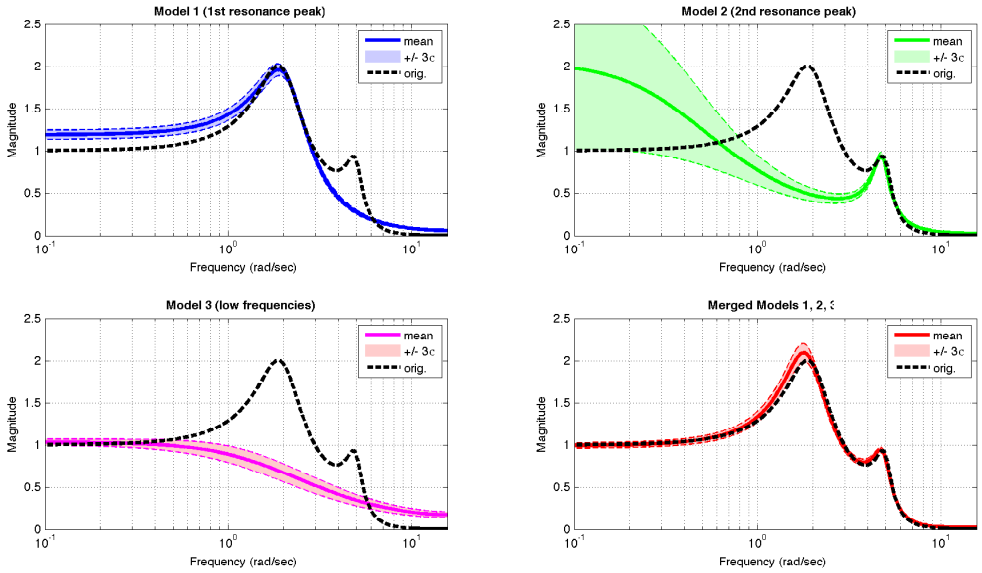
Merging of multiple models describing the same system. The merging is done in time domain and allows to combine information from uncertain ARX models with different orders (`idarx` or `idpoly`).

```
% Load 3 models of the same system identified from different data
% sets. The models have different structure.
[id_mod,sys] = demo_models;

for i=1:3,
  LS{i} = LSmodel(id_mod{i},'u1','y1');
```

```
end

LS12 = LS{1}.merge('M1',id_mod{2},4,4);
LS123 = LS12.merge('M1',id_mod{3},4,4);


% --- Plot the result ---------------------------
sc = 3; % variance scale
figure(10);clf;
w = logspace(-1,log10(pi/id_mod{1}.Ts),300);
X = [w w(end:-1:1)];
axx = [w(1) w(end) 0 2.5];

for i=1:3,
  [meanG{i}  ,varG{i}  ] = LS{i}.freq_uncert('M1',w);
end
[meanG123,varG123] = LS123.freq_uncert('M1',w);

line_c = { 'b','g','m' };
fill_c = { [.8 .8 1],[.8 1 .8],[1 .8 .8]};
for i=1:3,
  subplot(2,2,i);
  h(i) = semilogx(w,meanG{i},line_c{i},'LineWidth',3); hold on;
  Y = meanG{i}+sc*sqrt(varG{i});
  Y = [meanG{i}-sc*sqrt(varG{i}) Y(end:-1:1)];
  hf(i) = fill(X,Y,fill_c{i},'EdgeAlpha',0,...
               'FaceAlpha',0.5,'LineStyle','none');
  semilogx(w,[meanG{i}-sc*sqrt(varG{i}) ; ...
              meanG{i}+sc*sqrt(varG{i})]',...
              [line_c{i} '--'],'LineWidth',1);
  switch i,
    case 1, title('\bf Model 1 (1st resonance peak)');
    case 2, title('\bf Model 2 (2nd resonance peak)');
    case 3, title('\bf Model 3 (low frequencies)');
  end
end

subplot(2,2,4);
h(4) = semilogx(w,meanG123,'r','LineWidth',3); hold on;
Y = meanG123+sc*sqrt(varG123);
Y = [meanG123-sc*sqrt(varG123) Y(end:-1:1)];
hf(4) = fill(X,Y,[1 .8 .8],'EdgeAlpha',0,...
             'FaceAlpha',0.5,'LineStyle','none');
semilogx(w,[meanG123-sc*sqrt(varG123) ; ...
            meanG123+sc*sqrt(varG123)]','r--','LineWidth',1);
title('\bf Merged Models 1, 2, 3');

[mag,phase] = bode(sys,w);
for i=1:4,
  subplot(2,2,i);
  horig(i) = semilogx(w,squeeze(mag),'k--','LineWidth',3);
  axis(axx); grid on;
  xlabel('Frequency (rad/sec)'); ylabel('Magnitude');
  legend([h(i) hf(i) horig(i)] , ...
```

```
        'mean',['+/- ',num2str(sc),'\sigma'],'orig.');
end
```

```
Current iteration error = 30568.7304
Current iteration error = 30568.7304
Current iteration error = 30568.7304
Current iteration error = 370494.737
Current iteration error = 410827.2347
Current iteration error = 68383.1445
Current iteration error = 30568.7304
Current iteration error = 30568.7304
Current iteration error = 30568.7304
Current iteration error = 203078.9615
Maximum number of iterations exceeded.
Precission matrix quadratic error (normalized)= 0.02566
Current iteration error = 75719.3471
Current iteration error = 914945.2183
Current iteration error = 870103.539
Current iteration error = 54613.5888
Current iteration error = 155621.7346
Current iteration error = 92255.2874
Current iteration error = 137822.2708
Current iteration error = 863230.4205
Current iteration error = 66342.421
Current iteration error = 947935.3184
Maximum number of iterations exceeded.
Precission matrix quadratic error (normalized)= 0.068154
Current iteration error = 4469951.3757
Current iteration error = 2424856.2526
Current iteration error = 2315073.8027
Current iteration error = 2853276.0984
Current iteration error = 2511910.3196
Current iteration error = 2487229.5597
Current iteration error = 2455166.6466
Current iteration error = 2764858.2541
Current iteration error = 2181018.5937
Current iteration error = 2194488.5714
Maximum number of iterations exceeded.
Precission matrix quadratic error (normalized)= 1.1806
 **********  Warning: Fictive data quality may be low.  **********
Current iteration error = 1.5882e-20
Precission matrix quadratic error (normalized)= 9.4199e-26
```



Overloaded functions

Many standard functions are overloaded for LSmodel class.

```
dcgain(mod)
```

```
Warning: Matrix is singular, close to singular or badly scaled.
         Results may be inaccurate. RCOND = NaN.

ans =

   NaN   NaN   NaN
   NaN   NaN   NaN
   NaN   NaN   NaN
```

```
pole(mod)
```

```
ans =

  -6.6715
  -4.4112 + 1.3459i
  -4.4112 - 1.3459i
  -0.2786 + 1.6244i
  -0.2786 - 1.6244i
  -2.2857
  -2.1010 + 0.2367i
  -2.1010 - 0.2367i
  -0.1335 + 0.1818i
  -0.1335 - 0.1818i
  -0.3904
```

```
  -1.8142
  -0.7314
  -0.7882
  -0.9910
  -1.6759
  -1.1631
  -1.1784
  -1.5414
  -1.3276
  -1.3349
  -1.4047
  -0.5040 + 2.7656i
  -0.5040 - 2.7656i
  -0.3553
  -6.6356
  -0.3594
  -3.4410
  -3.8862
  -3.7298
  -1.2730
  -3.6092
  -9.9454
  -2.3938
  -0.7164
```

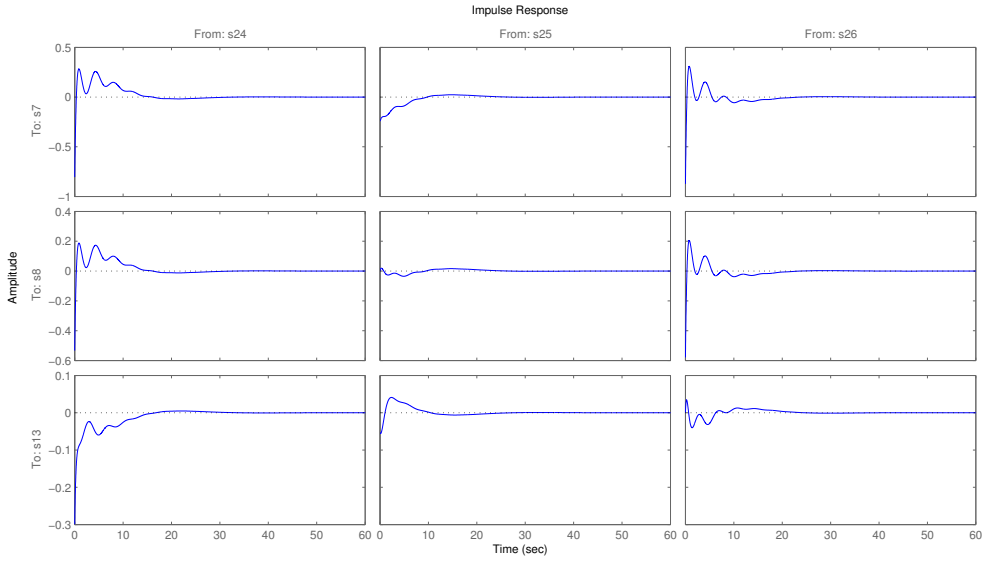
```
zero(mod)
```

```
ans =

  -7.3180
  -6.2397
  -0.5040 + 2.7655i
  -0.5040 - 2.7655i
   1.1169
  -0.0229 + 1.2125i
  -0.0229 - 1.2125i
  -2.8319
  -0.0010
  -0.8764 + 0.6952i
  -0.8764 - 0.6952i
  -0.3551
  -0.4630
  -0.7853
  -2.0245
  -1.9684
  -1.9588
  -1.1695
  -1.7014
  -1.5418
  -1.4130
  -1.4413
  -6.6356
```

```
-0.3594
-3.4410
-3.8862
-3.7298
-1.2730
-3.6092
-9.9454
-2.3938
-0.7164
```

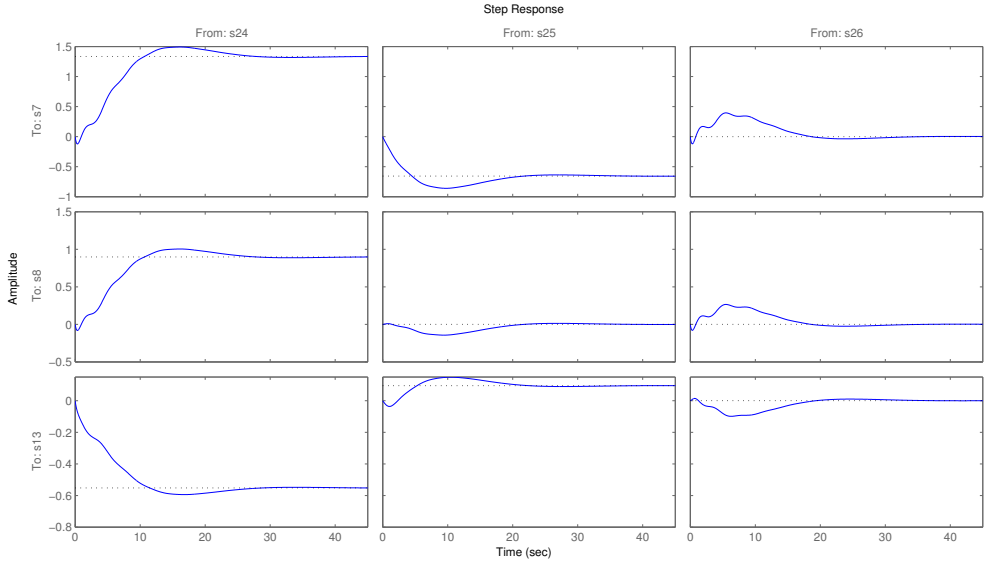
Impulse response

```
figure(10);clf;
impulse(mod);
```



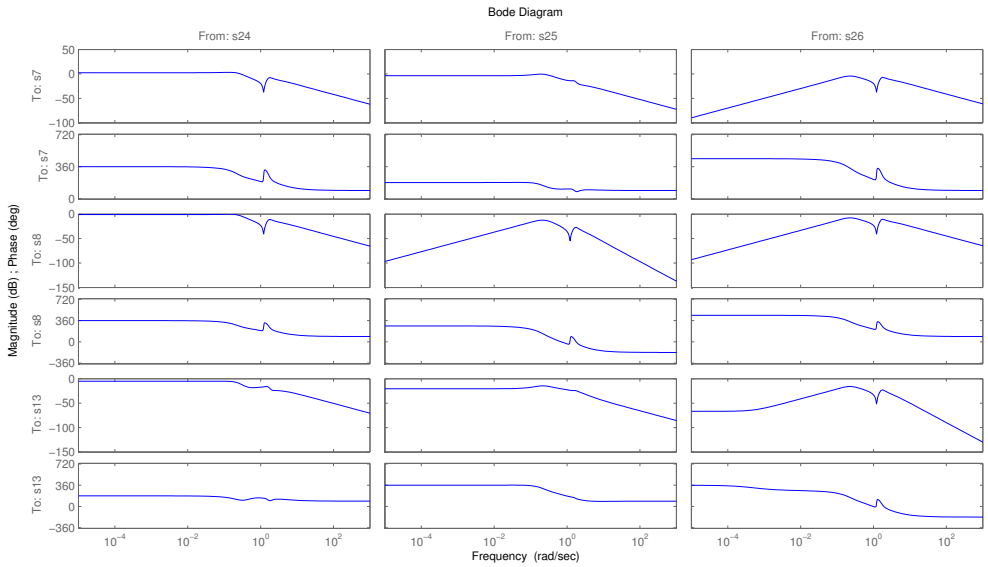
Step response

```
step(mod);
```



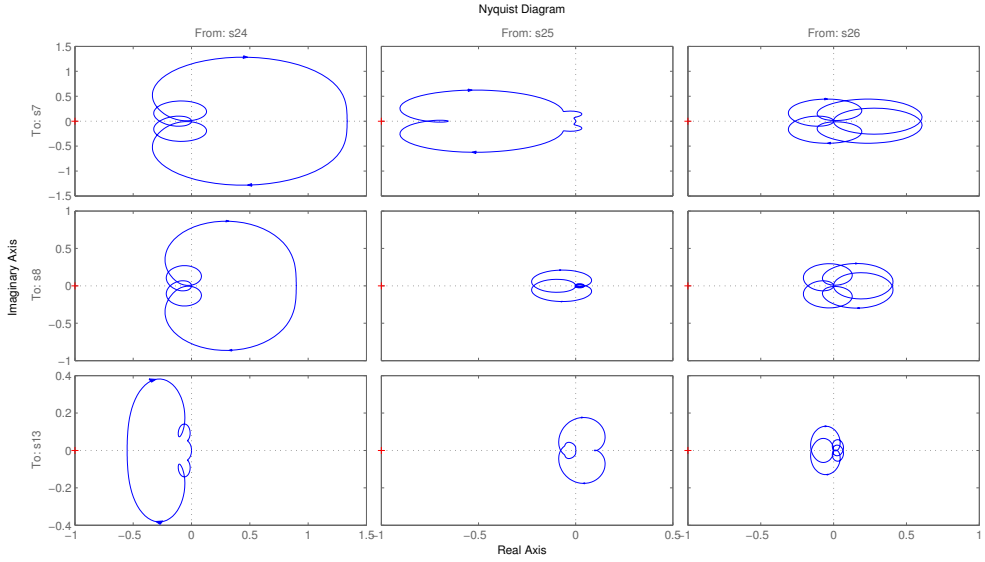
Bode frequency response

```
bode(mod);
```



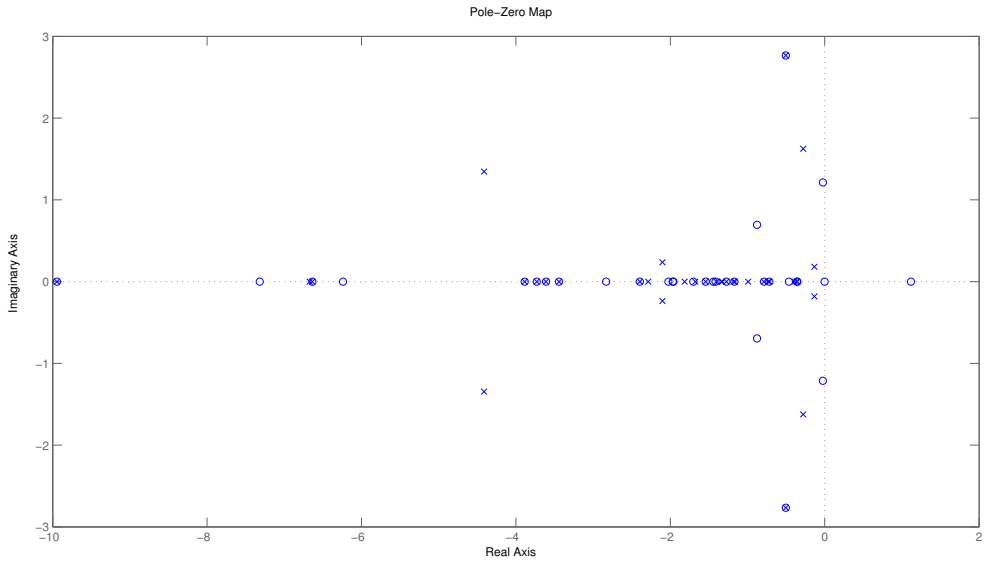
Nyquist frequency response

```
nyquist(mod);
```



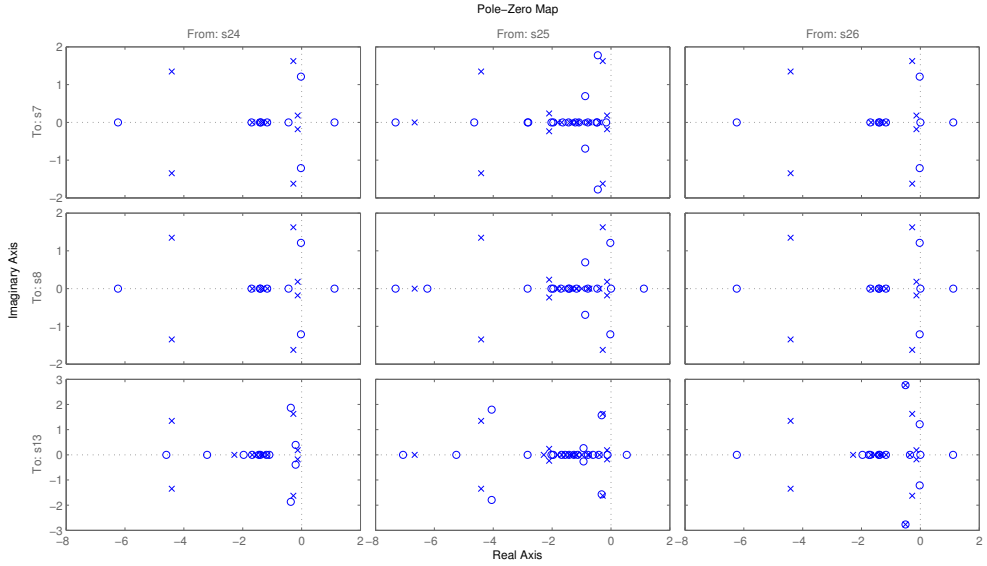
Pole-zero map

```
pzmap(mod);
```



Pole-zero map for I/O pairs

```
iopzmap(mod);
```



2.2 WNmodel

Demonstration of Water Deistribution Network Model Class WNmodel
This script demosntrates the use of WNmodel class for modeling and model management of Water Distribution Networks (WN). The demonstration is based on Barcelona water network.

WNmodel modle is child class of general LSmodel class.

Contents

Load water distribution network model from file

Water network is defined by two files [filename].net and [filename].mat, where '.net' defines network structure and '.net' defines variables limits, prices, etc.

Structure definition file is a simple text file describing individual tanks and their interconnections by specifying signal names and directions.

The first line of each tank definition block specifies tank number and its name:

```
Tank##,<tank name>
```

or it can specify node with it number

```
Node##
```

Following lines are same for tanks and nodes:

```
 d,<demand name>
 s,<source name>
 +,<outlet pump/valve name>,<destination tank name>
 -,<inlet pump/valve name>,<source tank name>
```

Tank/node definition blocks can be separated by empty line(s). Empty lines and Matlab type comments are ignored.

Variable definition file is standard Matlab MAT file with struct for every signal defining limits by fields

```
.min  ... min value
.max  ... max value
.dmin ... min slope value
.dmax ... max slope value
.smin ... soft limit min value
.smax ... soft limit max value
.type ... signal type ('MV','MD','UD','MO','MV','X')
```

Example (part of definition file):

```
Tank01,d450BEG
d,c450BEG
-,iBegues4,d369BEG

Tank02,d369BEG
d,c369BEG
+,iBegues4,d450BEG
-,iBegues3,d255BEG

Node1
s,AportA
d,c82PAL
+,vPalleja70,Node2
+,vPapiolATLL,d110PAP
+,iPapiol2AGBAR,d110PAP
+,vFontSanta,d54REL

Node2
d,c70PAL
+,iPalleja4,d125PAL
-,vPalleja70,Node1

<End of example>
```

```
mod = WNmodel('BCN_network')
```

```
No data for signal v70FLL70LLO
No data for signal v70LLO70FLL
Water network model (80 subsystem(s), 0 summator(s)):
  --- Tanks & Nodes ----------------
```

```
M01: Tank "d450BEG"    , 2 inputs
M02: Tank "d369BEG"    , 3 inputs
M03: Tank "d175LOR"    , 2 inputs
M04: Tank "d185VIL"    , 2 inputs
M05: Tank "d190SCL"    , 2 inputs
M06: Tank "d255BEG"    , 3 inputs
M07: Tank "d150SBO"    , 3 inputs
M08: Tank "d135VIL"    , 3 inputs
M09: Tank "d114SCL"    , 3 inputs
M10: Tank "d184BEG"    , 3 inputs
M11: Tank "d263CES"    , 2 inputs
M12: Tank "d205CES"    , 3 inputs
M13: Tank "d147SCC"    , 3 inputs
M14: Tank "d361CGY"    , 2 inputs
M15: Tank "d268CGY"    , 3 inputs
M16: Tank "d374CGL"    , 2 inputs
M17: Tank "d313CGL"    , 2 inputs
M18: Tank "d200CGY"    , 3 inputs
M19: Tank "d246CGY"    , 2 inputs
M20: Tank "d252CGL"    , 4 inputs
M21: Tank "d195TOR"    , 5 inputs
M22: Tank "d125PAL"    , 4 inputs
M23: Tank "d205FON"    , 3 inputs
M24: Tank "d320FON"    , 3 inputs
M25: Tank "d175PAP"    , 2 inputs
M26: Tank "d400MGB"    , 3 inputs
M27: Tank "d110PAP"    , 4 inputs
M28: Tank "d320MGB"    , 3 inputs
M29: Tank "d437VVI"    , 3 inputs
M30: Tank "d300BAR"    , 4 inputs
M31: Tank "d176BARsud", 4 inputs
M32: Tank "d200BLL"    , 5 inputs
M33: Tank "d200BSO"    , 2 inputs
M34: Tank "d328SGE"    , 2 inputs
M35: Tank "d260SGE"    , 3 inputs
M36: Tank "d255CAR"    , 2 inputs
M37: Tank "d200ALT"    , 5 inputs
M38: Tank "d130BAR"    , 9 inputs
M39: Tank "d197BET"    , 3 inputs
M40: Tank "d200FDM"    , 2 inputs
M41: Tank "d132CMF"    , 3 inputs
M42: Tank "d215VALL"   , 4 inputs
M43: Tank "d184SMM"    , 2 inputs
M44: Tank "d101MIR"    , 9 inputs
M45: Tank "d169CME"    , 2 inputs
M46: Tank "d202CRU"    , 2 inputs
M47: Tank "d225GUI"    , 2 inputs
M48: Tank "d197GUI"    , 3 inputs
M49: Tank "d151BON"    , 2 inputs
M50: Tank "d117MTG"    , 2 inputs
M51: Tank "d190TCA"    , 2 inputs
M52: Tank "d171SAM"    , 3 inputs
M53: Tank "d144TPI"    , 2 inputs
M54: Tank "d120POM"    , 3 inputs
```

```
  M55: Tank "d70BBE"    , 8 inputs
  M56: Tank "d100CFE"   , 10 inputs
  M57: Tank "d54REL"    , 4 inputs
  M58: Tank "d10COR"    , 6 inputs
  M59: Tank "dPLANTA"   , 4 inputs
  M60: Tank "d80GAVi80CAS85", 9 inputs
  M61: Tank "d115CAST"  , 6 inputs
  M62: Tank "d130LSE"   , 2 inputs
  M63: Tank "d145MMA"   , 3 inputs
  M64: Node1            , 6 inputs
  M65: Node2            , 3 inputs
  M66: Node3            , 4 inputs
  M67: Node4            , 7 inputs
  M68: Node5            , 3 inputs
  M69: Node6            , 9 inputs
  M70: Node7            , 7 inputs
  M71: Node8            , 2 inputs
  M72: Node9            , 6 inputs
  M73: Node10           , 4 inputs
  M74: Node11           , 8 inputs
  M75: Node13           , 6 inputs
  M76: Node14           , 3 inputs
  M77: Node15           , 3 inputs
  M78: Node16           , 6 inputs
  M79: NodeA            , 5 inputs
  M80: NodeB            , 7 inputs
--- External Inputs -----------
  IN01: c450BEG         MD
  IN02: iBegues4        MV  (min=0,max=0.09)
  IN03: c369BEG         MD
  IN04: iBegues3        MV  (min=0,max=0.09)
  IN05: c175LOR         MD
  IN06: iOrioles        MV  (min=0,max=0.008)
  IN07: c185VIL         MD
  IN08: iViladecans2    MV  (min=0,max=0.015)
  IN09: c190SCL         MD
  IN10: iStCliment2     MV  (min=0,max=0.06)
  IN11: c255BEG         MD
  IN12: iBegues2        MV  (min=0,max=0.1)
  IN13: c150SBO         MD
  IN14: iStBoi          MV  (min=0,max=0.08)
  IN15: c135VIL         MD
  IN16: iViladecans1    MV  (min=0,max=0.08)
  IN17: c114SCL         MD
  IN18: iStCliment1     MV  (min=0,max=0.03)
  IN19: c184ESP         MD
  IN20: iBegues1        MV  (min=0,max=0.1)
  IN21: c263CES         MD
  IN22: iCesalpina2     MV  (min=0,max=0.025)
  IN23: c205CES         MD
  IN24: iCesalpina1     MV  (min=0,max=0.035)
  IN25: c147SCC         MD
  IN26: iStaClmCervello MV  (min=0,max=0.04)
  IN27: c361CGY         MD
```

```
IN28: iCanGuey3       MV  (min=0,max=0.009)
IN29: c268CGY         MD
IN30: iCanGuey2       MV  (min=0,max=0.01)
IN31: c374CGL         MD
IN32: iCanGuell2d5    MV  (min=0,max=0.01)
IN33: c313CGL         MD
IN34: iCanGuell2d3    MV  (min=0,max=0.009)
IN35: c200CGY         MD
IN36: iCanGuey1d2     MV  (min=0,max=0.015)
IN37: c246CGY         MD
IN38: iCanGuey1d5     MV  (min=0,max=0.01)
IN39: c252CGL         MD
IN40: iCanGuell1      MV  (min=0,max=0.02)
IN41: c195TOR         MD
IN42: iCanRoig        MV  (min=0,max=0.027)
IN43: aMS             MV
IN44: c125PAL         MD
IN45: iPalleja4       MV  (min=0,max=0.035)
IN46: iPalleja1       MV  (min=0,max=0.03)
IN47: c205FON         MD
IN48: iPalleja2       MV  (min=0,max=0.03)
IN49: c320FON         MD
IN50: c356FON         MD
IN51: c175PAP135PAP   MD
IN52: iPapiol1        MV  (min=0,max=0.02)
IN53: c400MGB         MD
IN54: c475MGB         MD
IN55: iMasGuimbau2    MV  (min=0,max=0.005)
IN56: c110PAP         MD
IN57: vPapiolATLL     MV  (min=0,max=0.075)
IN58: iPapiol2AGBAR   MV  (min=0,max=0.025)
IN59: c320MGB         MD
IN60: iMasGuimbau1    MV  (min=0,max=0.008)
IN61: c437VVI         MD
IN62: c541TIB         MD
IN63: iTibidabo       MV  (min=0,max=0.066)
IN64: c300BAR         MD
IN65: iFinestrelles300 MV  (min=0,max=0.5)
IN66: c176BARsud      MD
IN67: vFinestrllEsplg MV  (min=0,max=0.2)
IN68: iFinestrelles176 MV  (min=0,max=0.23)
IN69: vBonanova       MV  (min=0,max=0.4)
IN70: c200BLL         MD
IN71: iBellsoleig     MV  (min=0,max=0.006)
IN72: iCornella130    MV  (min=0,max=0.2)
IN73: iFinestrelles200 MV  (min=0,max=0.6)
IN74: vFinestrelles   MV  (min=0,max=0.8)
IN75: c200BSO         MD
IN76: c328SGE         MD
IN77: iStGenis2       MV  (min=0,max=0.03)
IN78: c260SGE         MD
IN79: iStGenis1       MV  (min=0,max=0.25)
IN80: c255CAR         MD
IN81: iCarmel         MV  (min=0,max=0.1)
```

```
IN82: c200ALT         MD
IN83: c150ALT         MD
IN84: iAltures        MV  (min=0,max=0.425)
IN85: vBaroStLluis    MV  (min=0,max=0.15)
IN86: c130BAR         MD
IN87: vMix1           MV  (min=0,max=3.2)
IN88: iCollblanc      MV  (min=0,max=0.9)
IN89: vCollblanc      MV  (min=0,max=0.8)
IN90: vEsplugues      MV  (min=0,max=0.5)
IN91: c197BET         MD
IN92: c238UAB         MD
IN93: iCerdUAB        MV  (min=0,max=0.2)
IN94: c200FDM         MD
IN95: iFlorMaig       MV  (min=0,max=0.01)
IN96: c132CMF         MD
IN97: iCerdMontflorit MV  (min=0,max=0.3)
IN98: c260VALL        MD
IN99: c275BEV         MD
IN100: c215VALL        MD
IN101: iVallensana1    MV  (min=0,max=0.01)
IN102: c184SMM         MD
IN103: iStaMaMontcada  MV  (min=0,max=0.008)
IN104: c101MIR         MD
IN105: c250VASAB       MD
IN106: iTorreBaro1     MV  (min=0,max=0.2)
IN107: vTerMontcada    MV  (min=0,max=0.35)
IN108: vBesosMontcCerd MV  (min=0,max=0.8)
IN109: c169CME         MD
IN110: c202CRU         MD
IN111: iCanRuti        MV  (min=0,max=0.04)
IN112: c225GUI         MD
IN113: iGuinardera2    MV  (min=0,max=0.008)
IN114: c197GUI         MD
IN115: iGuinardera1    MV  (min=0,max=0.04)
IN116: c151BON         MD
IN117: iBonavista      MV  (min=0,max=0.01)
IN118: c117MTG         MD
IN119: vMontigala      MV  (min=0,max=0.1)
IN120: c190TCA         MD
IN121: iTorreoCastell  MV  (min=0,max=0.04)
IN122: c171SAM         MD
IN123: iMntjcStaAmalia MV  (min=0,max=0.18)
IN124: c144TPI         MD
IN125: iMntjcTresPins  MV  (min=0,max=0.2)
IN126: c120POM         MD
IN127: iMorera         MV  (min=0,max=0.06)
IN128: vConflent       MV  (min=0,max=0.1)
IN129: c70BBE          MD
IN130: c55BAR          MD
IN131: vPsgStJoan      MV  (min=0,max=1)
IN132: vTrinitat70     MV  (min=0,max=2)
IN133: vCerdaTraja     MV  (min=0,max=1.8)
IN134: c100CFE         MD
IN135: vSJD            MV  (min=0,max=1)
```

```
IN136: iEsplugues      MV  (min=0,max=0.65)
IN137: vRossichMaq     MV  (min=0,max=3.61)
IN138: vZonaFranca     MV  (min=0,max=3)
IN139: iCornella100    MV  (min=0,max=3.5)
IN140: iRelleu         MV  (min=0,max=3.5)
IN141: vPousEstrella   MV  (min=0,max=0.23)
IN142: vFontSanta      MV  (min=0,max=1.2)
IN143: iCornella50     MV  (min=0,max=0.6)
IN144: iSJD50          MV  (min=0,max=1.8)
IN145: cRECARREGA      MD
IN146: iCornella70     MV  (min=0,max=0.5)
IN147: iSJD10          MV  (min=0,max=2.9)
IN148: iSJD70          MV  (min=0,max=0.4)
IN149: vSJDTot         MV  (min=0,max=5.3)
IN150: c80GAVi80CAS85  MD
IN151: aPousCAST       MV
IN152: vGava100a80     MV  (min=0,max=0.4)
IN153: iGava4          MV  (min=0,max=0.06)
IN154: vCanyars        MV  (min=0,max=0.15)
IN155: iLaSentiu       MV  (min=0,max=0.008)
IN156: iBellamar       MV  (min=0,max=0.06)
IN157: iCastelldefels  MV  (min=0,max=0.15)
IN158: vCanRoca        MV  (min=0,max=0.05)
IN159: c115CAST        MD
IN160: aCAST8          MV
IN161: iMasJove        MV  (min=0,max=0.025)
IN162: c130LSE         MD
IN163: c145MMA         MD
IN164: c175BVI         MD
IN165: AportA          MV
IN166: c82PAL          MD
IN167: vPalleja70      MV  (min=0,max=0.06)
IN168: c70PAL          MD
IN169: c140LLO         MD
IN170: c200BARsc       MD
IN171: c176BARnord     MD
IN172: vMinaCiutat     MV  (min=0,max=2)
IN173: vPortola        MV  (min=0,max=0.15)
IN174: c176BARcentre   MD
IN175: c135MGA         MD
IN176: AportT          MV
IN177: vTrinitat200    MV  (min=0,max=0.8)
IN178: vTerStaColoma   MV  (min=0,max=0.28)
IN179: vMix2           MV  (min=0,max=3.4)
IN180: c100LLO         MD
IN181: c100BES         MD
IN182: vBesosStaColoma MV  (min=0,max=0.5)
IN183: c100BLLsud      MD
IN184: vMix3           MV  (min=-5.9,max=5)
IN185: vTorrassa       MV  (min=0,max=2)
IN186: c100BLLcentre   MD
IN187: vCncpcioArenal  MV  (min=0,max=1.5)
IN188: c70FLL          MD
IN189: c70CFE          MD
```

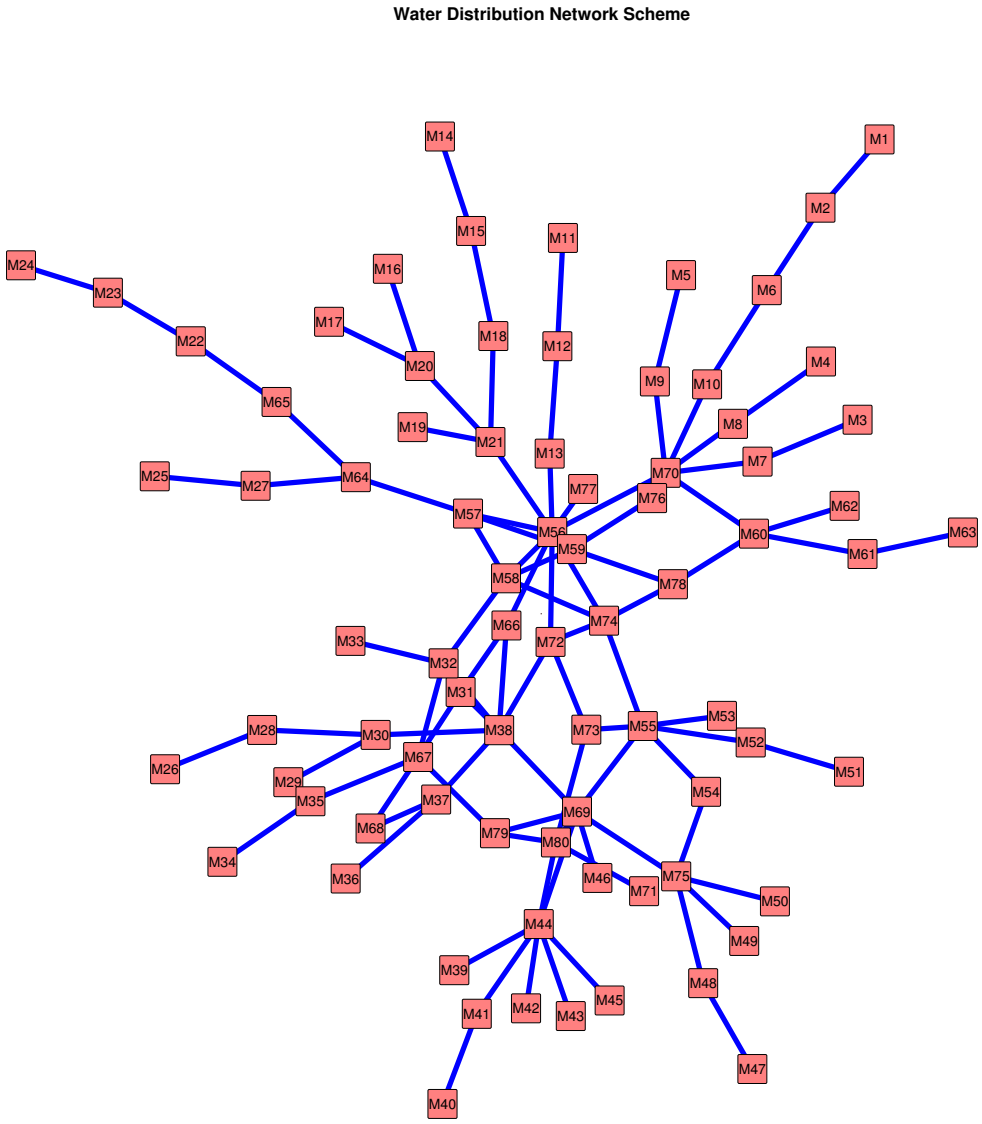
```
   IN190: v70FLL70LLO     MV  (min=0,max=0.159)
   IN191: v70LLO70FLL     MV  (min=0,max=0.104)
   IN192: c135SCG         MD
   IN193: AportLL1        MV
   IN194: AportLL2        MV
   IN195: aPousE1         MV
   IN196: aPousE2         MV
   IN197: c70LLO          MD
   IN198: c250TBA         MD
   IN199: c200BARnord     MD
   IN200: iRoquetes       MV  (min=0,max=0.25)
   IN201: c100BLLnord     MD
   IN202: aPousB          MV
 --- External Outputs ----------
   OUT01: d450BEG         MO  (min=100,max=2900)
   OUT02: d369BEG         MO  (min=400,max=2900)
   OUT03: d175LOR         MO  (min=10,max=80)
   OUT04: d185VIL         MO  (min=20,max=205)
   OUT05: d190SCL         MO  (min=150,max=1000)
   OUT06: d255BEG         MO  (min=400,max=2900)
   OUT07: d150SBO         MO  (min=340,max=2750)
   OUT08: d135VIL         MO  (min=300,max=920)
   OUT09: d114SCL         MO  (min=113,max=480)
   OUT10: d184BEG         MO  (min=400,max=2900)
   OUT11: d263CES         MO  (min=200,max=1600)
   OUT12: d205CES         MO  (min=50,max=300)
   OUT13: d147SCC         MO  (min=32,max=801)
   OUT14: d361CGY         MO  (min=20,max=92)
   OUT15: d268CGY         MO  (min=18,max=82)
   OUT16: d374CGL         MO  (min=100,max=500)
   OUT17: d313CGL         MO  (min=40,max=200)
   OUT18: d200CGY         MO  (min=12,max=100)
   OUT19: d246CGY         MO  (min=7,max=100)
   OUT20: d252CGL         MO  (min=50,max=270)
   OUT21: d195TOR         MO  (min=70,max=1900)
   OUT22: d125PAL         MO  (min=15,max=445)
   OUT23: d205FON         MO  (min=100,max=480)
   OUT24: d320FON         MO  (min=600,max=2000)
   OUT25: d175PAP         MO  (min=600,max=2100)
   OUT26: d400MGB         MO  (min=55,max=450)
   OUT27: d110PAP         MO  (min=375,max=960)
   OUT28: d320MGB         MO  (min=25,max=78)
   OUT29: d437VVI         MO  (min=1003,max=2985)
   OUT30: d300BAR         MO  (min=1050,max=5800)
   OUT31: d176BARsud      MO  (min=200,max=1035)
   OUT32: d200BLL         MO  (min=700,max=7300)
   OUT33: d200BSO         MO  (min=35,max=240)
   OUT34: d328SGE         MO  (min=82,max=1907)
   OUT35: d260SGE         MO  (min=350,max=3072)
   OUT36: d255CAR         MO  (min=34,max=465)
   OUT37: d200ALT         MO  (min=50,max=4240)
   OUT38: d130BAR         MO  (min=3840,max=16000)
   OUT39: d197BET         MO  (min=520,max=2800)
   OUT40: d200FDM         MO  (min=300,max=1000)
```

```
OUT41: d132CMF          MO  (min=500,max=2985)
OUT42: d215VALL         MO  (min=50,max=300)
OUT43: d184SMM          MO  (min=50,max=205)
OUT44: d101MIR          MO  (min=1403,max=4912)
OUT45: d169CME          MO  (min=50,max=3002)
OUT46: d202CRU          MO  (min=40,max=275)
OUT47: d225GUI          MO  (min=40,max=190)
OUT48: d197GUI          MO  (min=100,max=3000)
OUT49: d151BON          MO  (min=10,max=43)
OUT50: d117MTG          MO  (min=1000,max=4500)
OUT51: d190TCA          MO  (min=2,max=32)
OUT52: d171SAM          MO  (min=99,max=1750)
OUT53: d144TPI          MO  (min=447,max=4770)
OUT54: d120POM          MO  (min=150,max=1785)
OUT55: d70BBE           MO  (min=0,max=62750)
OUT56: d100CFE          MO  (min=16500,max=65200)
OUT57: d54REL           MO  (min=800,max=3100)
OUT58: d10COR           MO  (min=0,max=11745)
OUT59: dPLANTA          MO  (min=0,max=14450)
OUT60: d80GAVi80CAS85   MO  (min=480,max=3250)
OUT61: d115CAST         MO  (min=198,max=3870)
OUT62: d130LSE          MO  (min=21,max=130)
OUT63: d145MMA          MO  (min=100,max=480)
OUT64: Node1            X
OUT65: Node2            X
OUT66: Node3            X
OUT67: Node4            X
OUT68: Node5            X
OUT69: Node6            X
OUT70: Node7            X
OUT71: Node8            X
OUT72: Node9            X
OUT73: Node10           X
OUT74: Node11           X
OUT75: Node13           X
OUT76: Node14           X
OUT77: Node15           X
OUT78: Node16           X
OUT79: NodeA            X
OUT80: NodeB            X
```

Plot Water Network Structure

```
figure(1);clf;subfigure(1,1,1,1); %maximize figure
mod = plot(mod);
```

```
Computing vertex positions... Done.
```



Network Decomposition for Control

Decomposition of water network to a set of tanks for distributed control. EPS can be used to find given number of sets by applying graph algorithms of leaf condensation and \epsilon decomposition.

Decomposition to 6 groups

```
figure(2);clf;
subfigure(2,1,1,1);
mod6 = mod.eps(6);
mod6 = plot(mod6);
title('\bfDecomposition to 6 groups');
```

```
Undefined function or method 'components' for input arguments of type 'double'.
```

```
Error in ==> LSmodel.LSmodel>LSmodel.epsdec_groups at 1610
                  [comp, sizes] = components(sparse(A));

Error in ==> WNmodel.WNmodel>WNmodel.eps at 439
      [~,perm,block,~] = LSmodel.epsdec_groups(I,n_target,'inv');

Error in ==> WNmodel_example at 92
mod6 = mod.eps(6);
```

Decomposition to 12 groups

```
figure(3);clf;
subfigure(3,1,1,1);
mod12 = mod.eps(12);
mod12 = plot(mod12);
title('\bfDecomposition to 12 groups');
```

Networks separation

LS model of any group can be obtained by command GROUP.

```
figure(4);clf;
subfigure(4,1,1,1);
for i=1:12,
  subplot(3,4,i);
  plot( mod12.group(i) );
  title(['\bfGroup ' num2str(i)]);
end
```

Compress model

Model modifications using GROUPS, REM_MOD, SELECT, ... keeps unused inputs/outputs and submodels in LS model. SQUEEZE can be used to remove them.

```
mod12_gr9 = mod12.group(9);
mod12_gr9 = mod12_gr9.squeeze;

figure(5);clf;
subfigure(5,1,1,1);
step(mod12_gr9);
```

2.3 acg

Contents

Automatic Code Generation Example

This example shows how to use the ACG for simulating a networked control system.

For a detailed description of ACG please refer to the class reference.

```
close all
clc
```

Define number and positions of sensors and actuators

```
Ns = 1;
Na = 1;
Xpos = rand((Ns+Na)*2,1);
Ypos = rand((Ns+Na)*2,1);
```

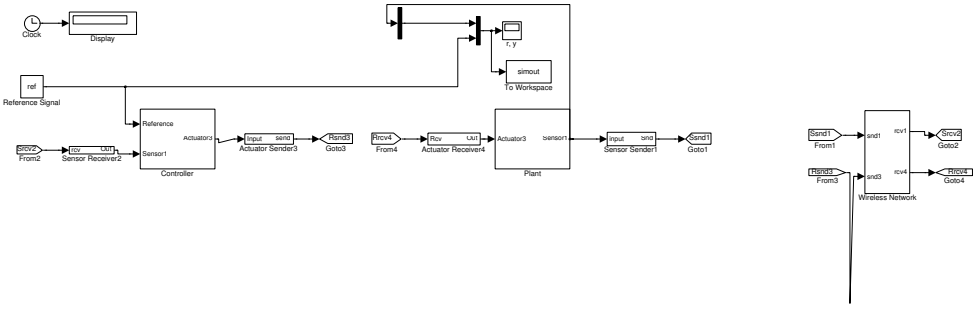
Set the mane to give to the generated .mld file

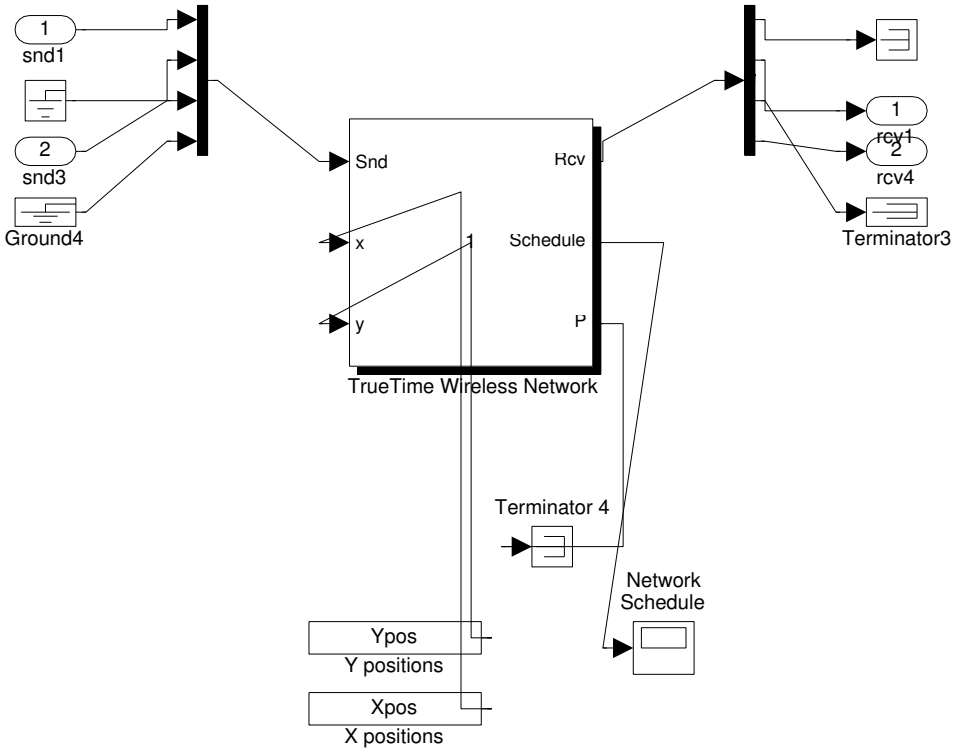
```
name='Example';
```

Create a new instance of ACG obj

```
acg_obj=ACG(Ns,Na,name);
acg_obj.RemoveOldCode;
acg_obj.GenerateCode;
```

```
Warning: Block diagram 'Example' contains disabled library links.  Use Model Advisor to find the
may not contain what you intended.
Warning: Block diagram 'Example' contains disabled library links.  Use Model Advisor to find the
may not contain what you intended.
```





Setup plant parameters

```
A=diag([0.1 .21 .3])+0.01*rand(3);
B=diag([.1 .3 .6]);
C=diag([2 4 1]);
D=zeros(3);
x0=[10 10 40];
```

Create and configure a the Plant and Controller submodels of the generated simulik model. Here the generation is performed in a script fashion for automatization proposes, however the user may use the graphical interface.

```
add_block('simulink/Signal Routing/Mux',[name '/Plant/Mux']);
set_param([name '/Plant/Mux'],'inputs','3');
add_block('simulink/Signal Routing/Demux',[name '/Plant/Demux']);
set_param([name '/Plant/Demux'],'outputs','3');
add_block('simulink/Continuous/State-Space',[name '/Plant/State-Space']);
add_line([name '/Plant'],'Mux/1','State-Space/1');
add_line([name '/Plant'],'State-Space/1','Demux/1');
for i=1:1
    add_line([name '/Plant'],['Actuator' num2str(i*2+Ns) '/1'],['Mux/'...
        num2str(i)]);
    add_line([name '/Plant'],['Demux/' num2str(i)],['Sensor' ...
        num2str(2*i-1) '/1']);
end
set_param([name '/Plant/State-Space'],'A','A','B','B','C','C','D',...
    'D','x0','x0');
```

Configure the controller

```
add_block('simulink/Signal Routing/Demux',[name '/Controller/Demux']);
set_param([name '/Controller/Demux'],'outputs','3');
for i=1:1
    ii=num2str(i);
    add_block('simulink/Math Operations/Sum',[name '/Controller/Sum' ii]);
    set_param([name '/Controller/Sum' ii],'ListOfSigns','+-|');
    add_block('simulink/Math Operations/Gain',[name '/Controller/Gain' ii]);
    add_line([name '/Controller'],['Demux/' ii],['Sum' ii '/1']);
    add_line([name '/Controller'],['Sensor' num2str(2*i-1) ,'/1'],['Sum'...
        ii '/2']);
    add_line([name '/Controller'],['Sum' ii '/1'],['Gain' ii '/1']);
    add_line([name '/Controller'],['Gain' ii '/1'],['Actuator' ...
        num2str(2*i+Ns) '/1']);
end
add_line([name '/Controller'],'Reference/1','Demux/1');
set_param([name '/Controller/Gain1'],'gain','30');
%set_param([name '/Controller/Gain2'],'gain','6');
%set_param([name '/Controller/Gain3'],'gain','60');
```

Setup simulation paramaters

Reference

```
ref = [8 -5 35]';
```

Antennas transmit power

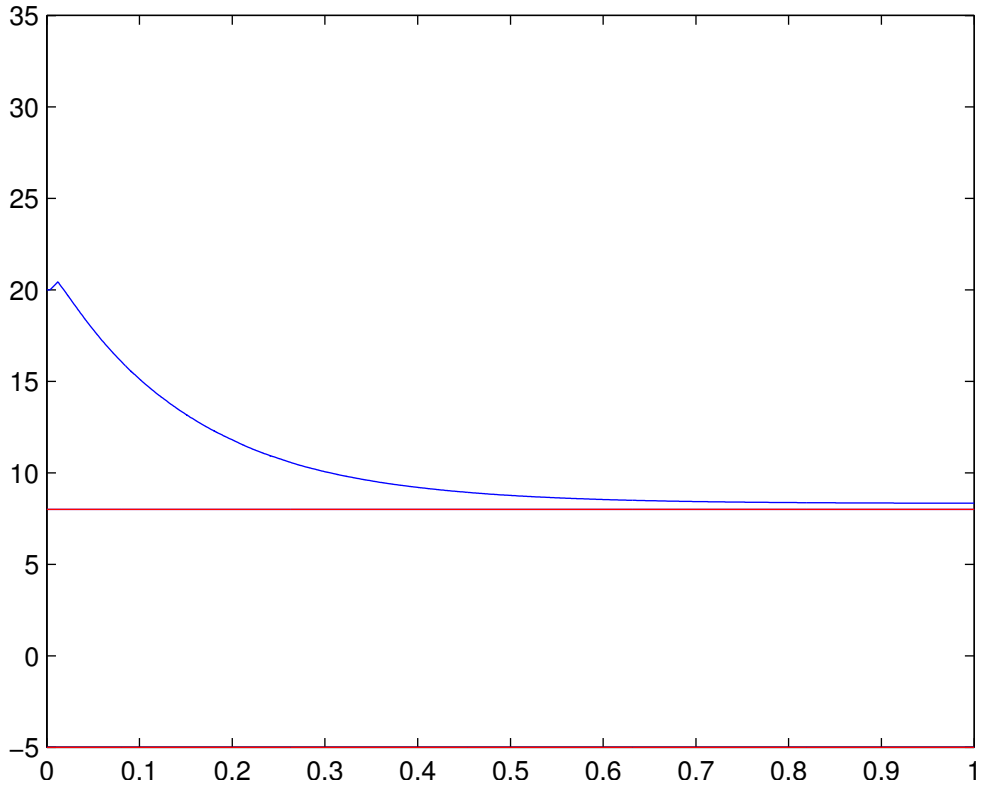
```
set_param([name '/Wireless Network/TrueTime Wireless Network'],...
    'TransPower','-30');
save_system(name)
```

```
Warning: Block diagram 'Example' contains disabled library links.  Use Model Advisor to find the
may not contain what you intended.
```

Start simulation

```
sim(name);

plot(simout.time,simout.signals.values,'b',simout.time,ref*ones(1,...
    length(simout.time)),'r');

disp('Few packets have gone lost using the current transmission power.')
disp(['Press any key for use a lower transmission power and observe'...
    ' the effects.'])
pause
```

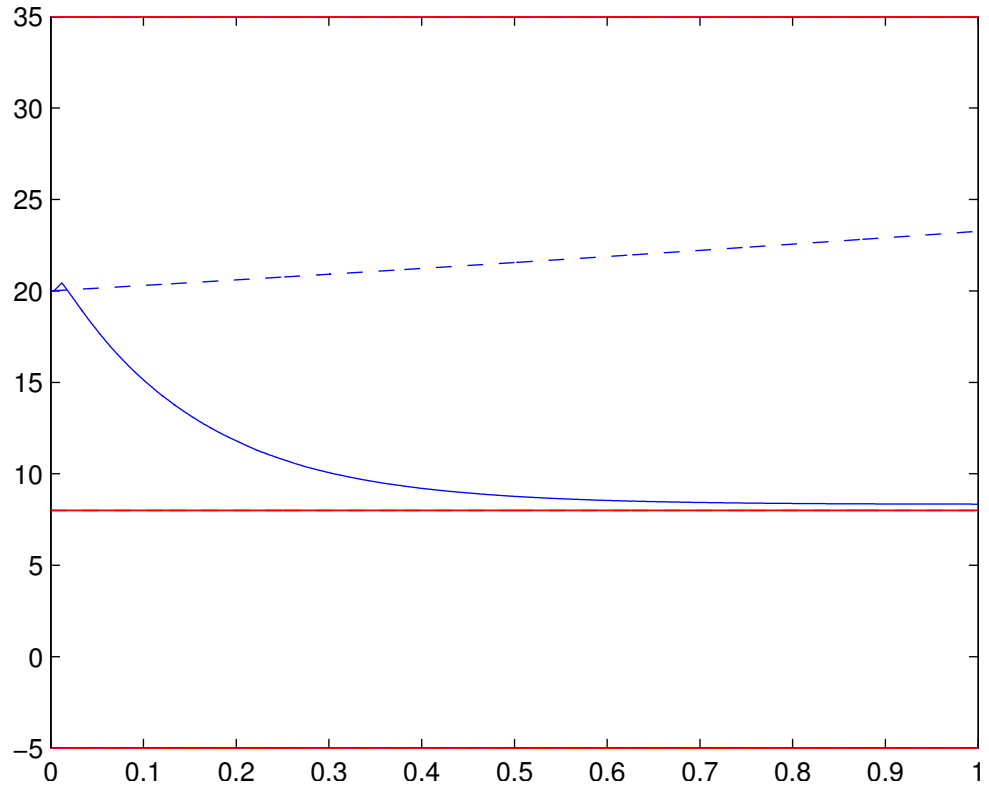
```
-------------------------------------------------------
                TrueTime, Version 1.5
                  Copyright (c) 2007
     Martin Ohlin, Dan Henriksson and Anton Cervin
          Department of Automatic Control LTH
                Lund University, Sweden
-------------------------------------------------------

Wireless network data:
Transmit power is: -30.00 dbm  <==> 0.00 mW
Receiver threshold is: -48.00 dbm  <==> 1.58e-05 mW
Maximum signal reach is calculated to: 62.10 m

Warning: Output port 2 of 'Example/Controller/Demux' is not connected.
Warning: Output port 3 of 'Example/Controller/Demux' is not connected.
Warning: Output port 2 of 'Example/Plant/Demux' is not connected.
Warning: Output port 3 of 'Example/Plant/Demux' is not connected.
Warning: Input port 2 of 'Example/Plant/Mux' is not connected.
Warning: Input port 3 of 'Example/Plant/Mux' is not connected.
Few packets have gone lost using the current transmission power.
Press any key for use a lower transmission power and observe the effects.
```



Decrease antenna transmit power

```
set_param([name '/Wireless Network/TrueTime Wireless Network'],...
    'TransPower','-46');
save_system(name)
```

```
sim(name);

hold on
plot(simout.time,simout.signals.values,'--b',simout.time,...
    ref*ones(1,length(simout.time)),'r');

Warning: Block diagram 'Example' contains disabled library links.  Use Model Advisor to find the
may not contain what you intended.
------------------------------------------------------------
                 TrueTime, Version 1.5
                   Copyright (c) 2007
     Martin Ohlin, Dan Henriksson and Anton Cervin
          Department of Automatic Control LTH
                Lund University, Sweden
------------------------------------------------------------

Wireless network data:
Transmit power is: -46.00 dbm  <==> 0.00 mW
Receiver threshold is: -48.00 dbm  <==> 1.58e-05 mW
Maximum signal reach is calculated to: 0.58 m

Warning: Output port 2 of 'Example/Controller/Demux' is not connected.
Warning: Output port 3 of 'Example/Controller/Demux' is not connected.
Warning: Output port 2 of 'Example/Plant/Demux' is not connected.
Warning: Output port 3 of 'Example/Plant/Mux' is not connected.
Warning: Input port 2 of 'Example/Plant/Mux' is not connected.
Warning: Input port 3 of 'Example/Plant/Mux' is not connected.
```



Conclusions

Figure 1 shows how the transmit power determine the performance of the control actions. In particular for the continous line (-30db transmit power) convergence is achieved, while for the lower trasmit power case (-46 db, depicted in dashed line) is possible to observe divergence of 2 of the 3 states.

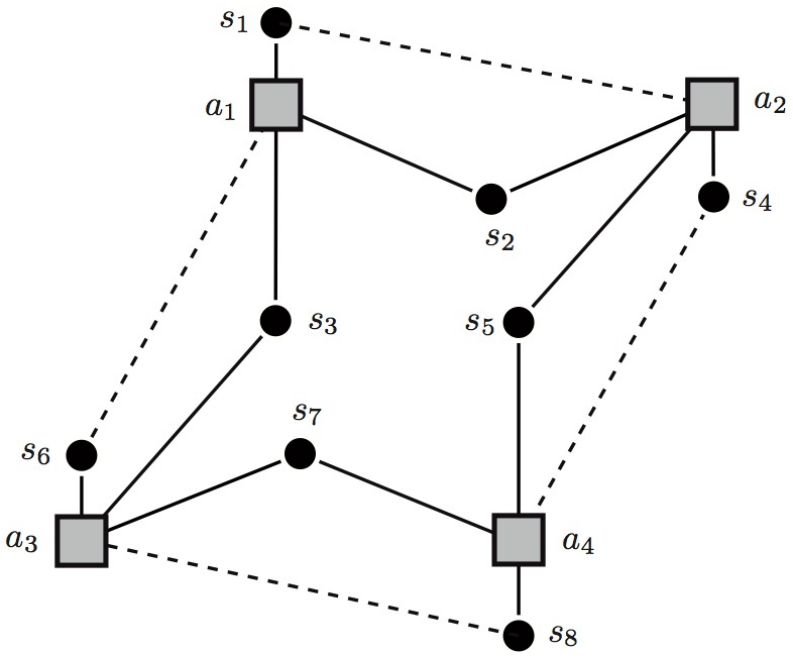
```
acg_obj.RemoveOldCode;
```

2.4 decLMI

Contents

Example of decLMI decentralized controller synthesis

This demo file shows how to use the DECLMI class to synthesize three LTI proportional controllers that guarantees stability while fulfilling constraints for the netwotk depicted below:



Controllers are:

- centralized ideal: each actuator can always exploit the whole state measurements to compute the control action;
- decentralized ideal: each actuator can exploit a subset of state measurements to compute the control action, and that set is time invariant;
- decentralized lossy: each controller can exploit a subset of state measurements, and measruements sent through some network links can be lost. No stochastic model of the dropouts is exploited here. The resulting controller guarantees robust convergence to the origin for any occurrence of packet dropouts.

Moreover, a stochastic controller is also computed, which guarantees closed-loop stability in the mean-square sense. It exploits a model of the packet dropouts as given by a two-state Markov chain. In this case the constraints are not guaranteed to hold at every time step.

```
close all
clc
```

Plant model

Net describes how the network is connected: * 1 = wired (reliable); * -1 = wireless (unreliable, i.e., seubject to dropouts); * 0 = no link.

```
      % e  f  g  h  j  k  l  m
Net = [ 1  1  1  0  0 -1  0  0   % a
       -1  1  0  1  1  0  0  0   % b
        0  0  1  0  0  1  1 -1   % c
        0  0  0 -1  1  0  1  1]; % d
[m,n]=size(Net);
```

State space matrices of the LTI discrete-time model $x(k+1) = A x(k) + B u(k)

```
A =[0.2843   -0.2895    0.1268   -0.0624   -0.2098  0.1882 -0.0061  0.2227;
   -0.2729    0.5052    0.0129   -0.1057   -0.1361  0.2154  0.2799 -0.0183;
    0.1610   -0.1229    0.1829   -0.4267    0.0589 -0.0905 -0.0417 -0.1709;
   -0.1084   -0.0138   -0.4611   -0.0735   -0.2013  0.1463  0.0854 -0.1567;
   -0.0300   -0.1167   -0.0012   -0.1847    0.3547  0.2081  0.0567  0.1566;
    0.2852    0.3036   -0.0994    0.1240    0.1099  0.1262  0.0229 -0.1599
   -0.0032    0.1558   -0.1285    0.1898   -0.0040  0.0017  0.7065 -0.1697;
    0.1961   -0.0173   -0.1023   -0.1888    0.0327 -0.2866 -0.1202  0.3198];
B =[     0   -2.2374   -0.4531         0;
   -0.1794    1.0976    1.3995    0.4287;
   -1.4671         0   -0.4620   -0.7370;
    1.3953   -1.6146    0.0327    0.5649;
    0.4408   -1.2287    0.7988   -1.3842;
    0.5654    0.2074    0.8968    0.4603;
         0         0    0.1379         0;
         0   -1.0061         0    0.3798];
```

Sample time

```
Ts = 1;
```

Constraints on input and state sqrt(x'x) <= xmax, sqrt(u'u)<=umax

```
xmax = 150;
umax = 150;

%Weights
Qx = eye(n);
Qu = 1e-2*eye(m);
```

Vertices of the initial state uncertainty polytope

```
X0 = [
    7.7618    7.7618    8.7618    8.7618
    6.0485    7.0485    6.0485    7.0485
    6.2866    6.2866    6.2866    6.2866
    6.6018    6.6018    6.6018    6.6018
    5.1547    5.1547    5.1547    5.1547
    7.8859    7.8859    7.8859    7.8859
    5.1507    5.1507    5.1507    5.1507
    5.0470    5.0470    5.0470    5.0470];
```

Markov chain description

The two-state Markov chain is described by:

- d : array, where d(i) is the probability of losing a packet being in the i-th state of the Markov chain;
- q : array, where q(i) is the probability of of remaining in the i-th state of the Markov chain.

```
Mc.d = [.1 .5];
Mc.q = [.8 .5];
```

Computation instance creation

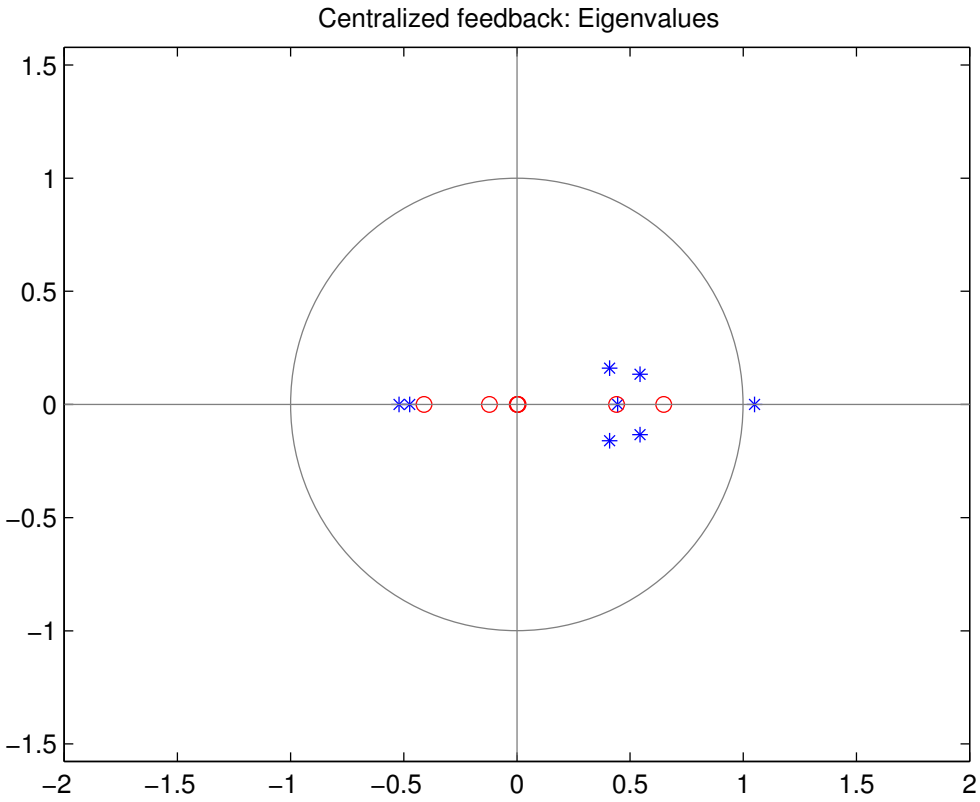
```
obj = decLMI(Net,A,B,Qx,Qu,X0,xmax,umax,Mc);
```

Now all the controller synthesis problems are formulated and solved. The time needed for each computation is stored for comparison purposes.

Centralized ideal

```
tic
obj=obj.solve_centralized_lmi();
CPUtime.centralized = toc;
```

```
Solving centralized SDP problem...
IBM ILOG License Manager: "IBM ILOG Optimization Suite for Academic Initiative" is accessing CPLE
Alg = 2: xz-corrector, theta = 0.250, beta = 0.500
eqs m = 69, order n = 101, dim = 1573, blocks = 9
nnz(A) = 1872 + 0, nnz(ADA) = 4761, nnz(L) = 2415
 it :     b*y       gap    delta  rate   t/tP*  t/tD*   feas cg cg  prec
  0 :            7.53E+04 0.000
  1 :  -1.78E+04 2.49E+04 0.000 0.3311 0.9000 0.9000   1.07  1  1  8.5E+00
  2 :  -1.94E+04 6.62E+03 0.000 0.2654 0.9000 0.9000   1.41  1  1  2.8E+00
  3 :  -5.13E+03 2.66E+03 0.000 0.4016 0.9000 0.9000   2.59  1  1  7.4E-01
  4 :  -4.41E+02 6.11E+02 0.000 0.2299 0.9000 0.9000   2.61  1  1  3.3E-01
  5 :  -1.02E+02 1.50E+02 0.000 0.2449 0.9000 0.9000   1.20  1  1  2.8E-01
  6 :  -7.92E+01 8.63E+01 0.000 0.5769 0.9000 0.9000   0.98  1  1  2.2E-01
  7 :  -1.33E+02 4.53E+01 0.000 0.5247 0.9000 0.9000   0.39  1  1  1.4E-01
  8 :  -1.77E+02 1.70E+01 0.000 0.3752 0.9000 0.9000   0.43  1  1  6.9E-02
  9 :  -1.92E+02 1.18E+01 0.008 0.6939 0.9000 0.9000   0.47  1  1  5.6E-02
 10 :  -2.25E+02 6.00E+00 0.000 0.5095 0.9000 0.9000   0.17  1  1  5.1E-02
 11 :  -2.47E+02 3.89E+00 0.000 0.6483 0.9000 0.9000   0.04  1  1  5.1E-02
 12 :  -3.03E+02 1.59E+00 0.000 0.4079 0.9000 0.9000   0.17  1  1  3.1E-02
 13 :  -3.32E+02 8.80E-01 0.000 0.5545 0.9000 0.9000   0.31  1  1  2.5E-02
 14 :  -3.60E+02 4.66E-01 0.000 0.5296 0.9000 0.9000   0.42  1  1  1.8E-02
 15 :  -3.82E+02 2.60E-01 0.000 0.5568 0.9000 0.9000   0.55  1  1  1.2E-02
 16 :  -3.98E+02 1.47E-01 0.000 0.5652 0.9000 0.9000   0.66  1  1  8.2E-03
 17 :  -4.10E+02 7.85E-02 0.000 0.5347 0.9000 0.9000   0.76  1  1  5.0E-03
 18 :  -4.19E+02 3.92E-02 0.000 0.5001 0.9000 0.9000   0.84  1  1  2.7E-03
 19 :  -4.24E+02 1.71E-02 0.000 0.4348 0.9000 0.9000   0.91  1  1  1.3E-03
 20 :  -4.27E+02 5.79E-03 0.000 0.3394 0.9000 0.9000   0.95  1  1  4.4E-04
 21 :  -4.28E+02 1.32E-03 0.000 0.2271 0.9000 0.9000   0.98  1  1  1.0E-04
 22 :  -4.29E+02 8.26E-05 0.192 0.0628 0.9900 0.9900   0.99  1  1  6.5E-06
 23 :  -4.29E+02 4.75E-06 0.000 0.0575 0.9900 0.9900   1.00  1  1  3.7E-07
 24 :  -4.29E+02 1.56E-08 0.398 0.0033 0.9990 0.9990   1.00  4  4  1.2E-09
Run into numerical problems.

iter seconds digits       c*x               b*y
 24      1.5   9.3 -4.2868543504e+02 -4.2868543527e+02
|Ax-b| =   2.7e-11, [Ay-c]_+ =   1.1E-08, |x|=  4.3e+02, |y|=  1.1e+03

Detailed timing (sec)
   Pre          IPM          Post
2.280E-01    8.476E-01    5.584E-02
Max-norms: ||b||=1, ||c|| = 22500,
Cholesky |add|=0, |skip| = 13, ||L.L|| = 2.09332e+07.
\nCentralized SDP problem solved!\n
```

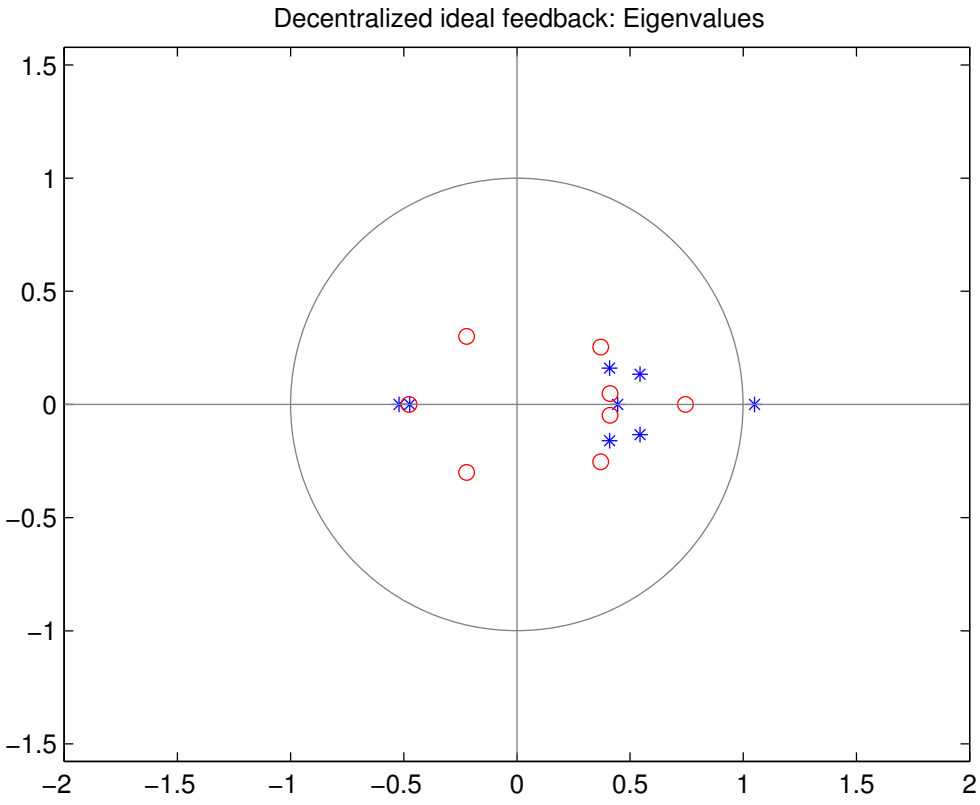


Decentralized ideal

```
tic
obj = obj.solve_dec_ideal_lmi();
CPUtime.dec_ideal = toc;
```

```
Solving decentralized SDP problem...

SeDuMi 1.3 by AdvOL, 2005-2008 and Jos F. Sturm, 1998-2003.
Alg = 2: xz-corrector, theta = 0.250, beta = 0.500
eqs m = 29, order n = 101, dim = 1573, blocks = 9
nnz(A) = 616 + 0, nnz(ADA) = 841, nnz(L) = 435
 it :     b*y       gap    delta  rate   t/tP*  t/tD*   feas cg cg  prec
  0 :            7.53E+04 0.000
  1 :  -1.76E+04 2.43E+04 0.000 0.3227 0.9000 0.9000   1.07  1  1  8.2E+00
  2 :  -1.92E+04 6.63E+03 0.000 0.2727 0.9000 0.9000   1.44  1  1  2.7E+00
  3 :  -5.30E+03 2.85E+03 0.000 0.4299 0.9000 0.9000   2.68  1  1  7.4E-01
  4 :  -4.06E+02 5.81E+02 0.000 0.2038 0.9000 0.9000   2.43  1  1  3.7E-01
  5 :  -1.02E+02 1.51E+02 0.000 0.2593 0.9000 0.9000   1.14  1  1  3.2E-01
  6 :  -8.58E+01 8.72E+01 0.000 0.5790 0.9000 0.9000   0.96  1  1  2.5E-01
  7 :  -1.34E+02 5.06E+01 0.000 0.5807 0.9000 0.9000   0.35  1  1  1.6E-01
  8 :  -1.94E+02 1.77E+01 0.226 0.3498 0.9000 0.9000   0.38  1  1  7.5E-02
  9 :  -2.12E+02 1.25E+01 0.280 0.7077 0.9000 0.9000   0.47  1  1  6.1E-02
 10 :  -2.26E+02 9.76E+00 0.039 0.7784 0.9000 0.9000   0.23  1  1  5.8E-02
 11 :  -2.34E+02 7.31E+00 0.000 0.7492 0.9000 0.9000  -0.31  1  1  7.0E-02
```

```
 12 :  -2.95E+02 3.23E+00 0.000 0.4419 0.9000 0.9000  -0.15  1  1  5.7E-02
 13 :  -3.83E+02 1.02E+00 0.000 0.3148 0.9000 0.9000   0.18  1  1  2.7E-02
 14 :  -4.30E+02 4.25E-01 0.000 0.4176 0.9000 0.9000   0.39  1  1  1.7E-02
 15 :  -4.56E+02 2.25E-01 0.000 0.5300 0.9000 0.9000   0.47  1  1  1.2E-02
 16 :  -4.87E+02 8.19E-02 0.000 0.3637 0.9000 0.9000   0.64  1  1  5.6E-03
 17 :  -5.00E+02 3.65E-02 0.000 0.4455 0.9000 0.9000   0.79  1  1  2.8E-03
 18 :  -5.09E+02 1.11E-02 0.000 0.3053 0.9000 0.9000   0.90  1  1  9.4E-04
 19 :  -5.12E+02 3.43E-03 0.000 0.3078 0.9000 0.9000   0.95  1  1  3.0E-04
 20 :  -5.13E+02 3.06E-04 0.000 0.0894 0.9900 0.9900   0.98  1  1  2.7E-05
 21 :  -5.13E+02 7.36E-05 0.000 0.2402 0.9000 0.9000   1.00  1  1  6.6E-06
 22 :  -5.13E+02 2.61E-07 0.294 0.0035 0.9990 0.9990   1.00  1  1  2.3E-08
 23 :  -5.13E+02 5.16E-08 0.000 0.1978 0.9000 0.9000   1.00  2  2  4.6E-09
 24 :  -5.13E+02 1.39E-09 0.000 0.0269 0.9900 0.9900   1.00  2  2  1.2E-10

iter seconds digits       c*x                b*y
 24      0.8  10.3 -5.1298252214e+02 -5.1298252217e+02
|Ax-b| =   2.8e-12, [Ay-c]_+ =   1.1E-09, |x|=  5.1e+02, |y|=  9.8e+02

Detailed timing (sec)
   Pre          IPM          Post
4.447E-02    5.670E-01    7.121E-03
Max-norms: ||b||=1, ||c|| = 22500,
Cholesky |add|=0, |skip| = 0, ||L.L|| = 362.833.

Decentralized SDP problem solved!
```

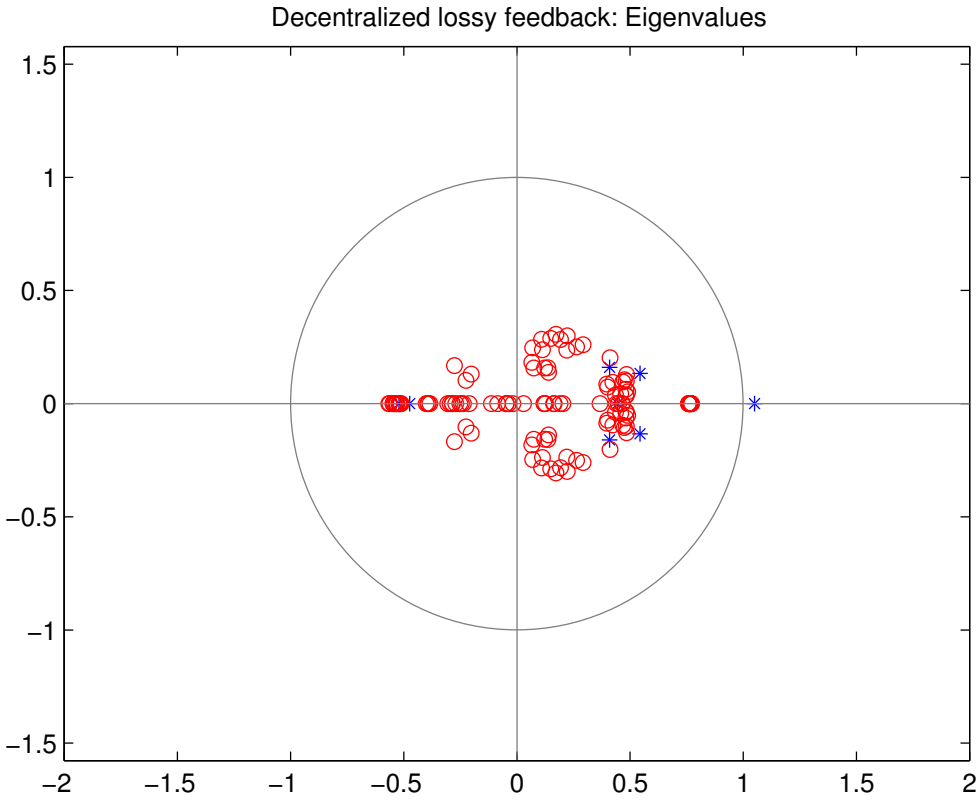


Decentralized lossy (robust)

```
tic
obj = obj.solve_dec_lossy_lmi();
CPUtime.dec_lossy = toc;
```

```
Solving dec. lossy SDP problem...

SeDuMi 1.3 by AdvOL, 2005-2008 and Jos F. Sturm, 1998-2003.
Alg = 2: xz-corrector, theta = 0.250, beta = 0.500
eqs m = 37, order n = 941, dim = 19333, blocks = 54
nnz(A) = 6056 + 0, nnz(ADA) = 1273, nnz(L) = 691
 it :     b*y       gap    delta  rate   t/tP*  t/tD*   feas cg cg  prec
  0 :            1.78E+06 0.000
  1 :  -1.93E+04 4.80E+05 0.000 0.2692 0.9000 0.9000   1.01  1  1  1.0E+02
  2 :  -3.01E+04 4.34E+04 0.000 0.0906 0.9900 0.9900   1.14  1  1  9.3E+00
  3 :  -2.40E+04 1.34E+04 0.000 0.3093 0.9000 0.9000   1.36  1  1  3.3E+00
  4 :  -6.26E+03 4.85E+03 0.000 0.3608 0.9000 0.9000   2.43  1  1  7.9E-01
  5 :  -7.24E+02 1.03E+03 0.000 0.2117 0.9000 0.9000   1.99  1  1  4.1E-01
  6 :  -3.94E+02 5.58E+02 0.000 0.5441 0.9000 0.9000   1.13  1  1  3.7E-01
  7 :  -3.69E+02 4.89E+02 0.000 0.8763 0.9000 0.9000   1.01  1  1  3.5E-01
  8 :  -4.46E+02 3.58E+02 0.000 0.7310 0.9000 0.9000   0.55  1  1  2.7E-01
  9 :  -5.19E+02 2.28E+02 0.000 0.6362 0.9000 0.9000   0.56  1  1  1.9E-01
 10 :  -5.62E+02 1.46E+02 0.000 0.6404 0.9000 0.9000   0.64  1  1  1.3E-01
 11 :  -5.74E+02 1.30E+02 0.133 0.8925 0.9000 0.9000   0.60  1  1  1.2E-01
 12 :  -6.23E+02 8.46E+01 0.000 0.6501 0.9000 0.9000   0.35  1  1  1.1E-01
 13 :  -6.76E+02 5.40E+01 0.000 0.6387 0.9000 0.9000   0.29  1  1  8.7E-02
 14 :  -7.84E+02 1.85E+01 0.000 0.3432 0.9000 0.9000   0.54  1  1  3.5E-02
 15 :  -8.24E+02 8.69E+00 0.000 0.4686 0.9000 0.9000   0.67  1  1  2.0E-02
 16 :  -8.48E+02 4.24E+00 0.000 0.4886 0.9000 0.9000   0.78  1  1  1.1E-02
 17 :  -8.69E+02 9.78E-01 0.000 0.2305 0.9000 0.9000   0.89  1  1  2.7E-03
 18 :  -8.75E+02 2.06E-01 0.000 0.2101 0.9000 0.9000   0.96  1  1  5.8E-04
 19 :  -8.76E+02 5.68E-02 0.000 0.2762 0.9000 0.9000   0.99  1  1  1.6E-04
 20 :  -8.76E+02 1.91E-02 0.000 0.3359 0.9000 0.9000   1.00  1  1  5.5E-05
 21 :  -8.76E+02 4.35E-03 0.000 0.2283 0.9000 0.9000   1.00  1  1  1.2E-05
 22 :  -8.76E+02 8.86E-04 0.000 0.2035 0.9000 0.9000   1.00  1  1  2.5E-06
 23 :  -8.76E+02 6.95E-05 0.403 0.0785 0.9900 0.9900   1.00  1  3  2.0E-07
 24 :  -8.76E+02 1.73E-05 0.000 0.2487 0.9000 0.9000   1.00  8  8  5.0E-08
 25 :  -8.76E+02 1.67E-06 0.434 0.0968 0.9900 0.9900   1.00 15 17  4.8E-09
 26 :  -8.76E+02 3.75E-07 0.000 0.2239 0.9000 0.9000   1.00 26 26  1.1E-09
 27 :  -8.76E+02 3.52E-07 0.000 0.9386 0.9000 0.9000   1.00 51 51  1.0E-09
 28 :  -8.76E+02 1.08E-07 0.000 0.3072 0.9000 0.9000   1.00 51 51  3.1E-10

iter seconds digits       c*x               b*y
 28     13.7   9.4 -8.7629587675e+02 -8.7629587712e+02
|Ax-b| =   2.6e-11, [Ay-c]_+ =   9.6E-10, |x|=  8.8e+02, |y|=  1.4e+03

Detailed timing (sec)
   Pre          IPM          Post
2.930E-02    7.203E+00    8.459E-03
Max-norms: ||b||=1, ||c|| = 22500,
Cholesky |add|=4, |skip| = 6, ||L.L|| = 33929.7.
Closed loop max eig: 0.76382
Closed loop max eig: 0.76244
```

```
Closed loop max eig: 0.76016
Closed loop max eig: 0.75831
Closed loop max eig: 0.76902
Closed loop max eig: 0.76785
Closed loop max eig: 0.76324
Closed loop max eig: 0.76251
Closed loop max eig: 0.76201
Closed loop max eig: 0.76155
Closed loop max eig: 0.7594
Closed loop max eig: 0.75854
Closed loop max eig: 0.77144
Closed loop max eig: 0.77039
Closed loop max eig: 0.76748
Closed loop max eig: 0.76679
```

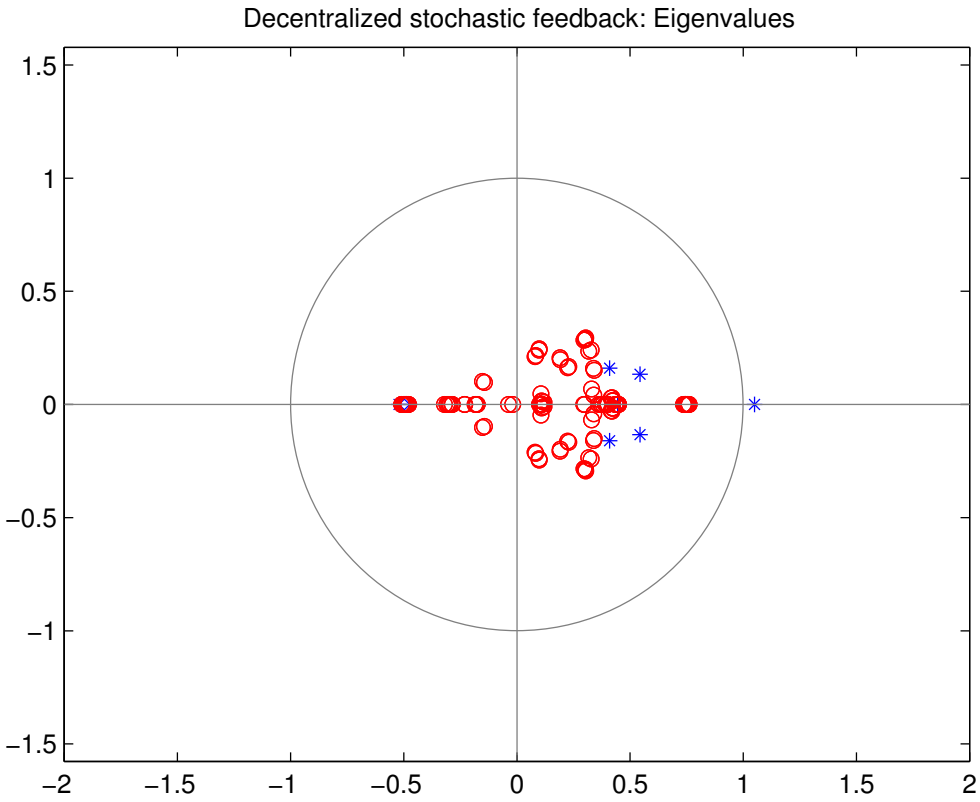


Decentralized stochastic

```
tic
obj = obj.solve_dec_stoch_lmi();
CPUtime.dec_stoch = toc;
```

```
Solving decentralized stoch. SDP problem...

SeDuMi 1.3 by AdvOL, 2005-2008 and Jos F. Sturm, 1998-2003.
```

```
Alg = 2: xz-corrector, theta = 0.250, beta = 0.500
Put 2 free variables in a quadratic cone
eqs m = 47, order n = 758, dim = 226511, blocks = 13
nnz(A) = 10685 + 0, nnz(ADA) = 2029, nnz(L) = 1038
 it :     b*y       gap    delta  rate   t/tP*  t/tD*   feas cg cg  prec
  0 :            5.13E+01 0.000
  1 :  -9.89E+03 1.44E+01 0.000 0.2810 0.9000 0.9000  -0.72  1  1  2.4E+02
  2 :  -8.93E+03 7.30E+00 0.000 0.5066 0.9000 0.9000  -0.60  1  1  2.1E+02
  3 :  -8.68E+03 5.00E+00 0.000 0.6848 0.9000 0.9000  -0.49  1  1  1.9E+02
  4 :  -7.37E+03 2.50E+00 0.000 0.4997 0.9000 0.9000  -0.42  1  1  1.5E+02
  5 :  -5.18E+03 8.68E-01 0.000 0.3475 0.9000 0.9000  -0.25  1  1  9.3E+01
  6 :  -3.00E+03 2.08E-01 0.000 0.2397 0.9000 0.9000   0.05  1  1  3.7E+01
  7 :  -1.72E+03 6.69E-03 0.000 0.0322 0.9900 0.9900   0.46  1  1  1.7E+00
  8 :  -1.64E+03 1.37E-04 0.000 0.0204 0.9900 0.9900   0.98  1  1  3.4E-02
  9 :  -1.10E+03 3.39E-05 0.000 0.2479 0.9000 0.9000   1.68  1  1  5.7E-03
 10 :  -9.14E+02 1.48E-05 0.000 0.4378 0.9000 0.9000   1.61  1  1  2.1E-03
 11 :  -8.84E+02 6.51E-06 0.000 0.4386 0.9000 0.9000   1.15  1  1  8.9E-04
 12 :  -8.79E+02 2.31E-06 0.000 0.3544 0.9000 0.9000   1.03  1  1  3.1E-04
 13 :  -8.76E+02 1.95E-07 0.000 0.0846 0.9900 0.9900   1.01  1  1  2.6E-05
 14 :  -8.76E+02 9.14E-09 0.000 0.0468 0.9900 0.9900   1.00  1  1  1.2E-06
 15 :  -8.76E+02 7.45E-10 0.000 0.0815 0.9900 0.9900   1.00  1  1  1.0E-07
 16 :  -8.76E+02 6.33E-11 0.000 0.0850 0.9900 0.9900   1.00  2  2  8.6E-09
 17 :  -8.76E+02 3.35E-12 0.142 0.0530 0.9900 0.9900   1.00  3  3  4.5E-10

iter seconds digits       c*x               b*y
 17     50.6   8.9 -8.7629588173e+02 -8.7629588070e+02
|Ax-b| =   3.2e-07, [Ay-c]_+ =   1.4E-11, |x|=  5.3e+04, |y|=  1.7e+03

Detailed timing (sec)
   Pre          IPM          Post
6.146E-02    2.680E+01    1.172E-01
Max-norms: ||b||=1000, ||c|| = 1.752360e+01,
Cholesky |add|=0, |skip| = 0, ||L.L|| = 26679.7.

Decentralized stoch. SDP problem solved!
```



Simulation

Random initial state in X0

```
x0 =[
    8.3272
    5.7953
    7.2282
    8.6640
    5.8317
    6.6488
    8.1784
    6.5376];
```

Simulation length

```
Tsim = 50;
```

Number of simulations to be carried out

```
Nsim = 50;
```

For a proper comparison, a sequence of states and emissions of the Markov chain model is stored.

```
load Statechange_paper
load Randvals_paper

Sim = [];
for iter=1:Nsim
```

Generate realizations of Markov chain

```
    statechange = Statechange(1,:,iter);
    randvals = Randvals(1,:,iter);
```

Initial Markov chain state

```
    initialstate = 1;
```

Compute sequence of Markov chain states

```
    trc = cumsum(obj.Mc.T,2);
    ec = cumsum(obj.Mc.E,2);
    currentstate = initialstate;
    numStates = size(obj.Mc.T,1);
    numEmissions = size(obj.Mc.E,2);
    for i = 1:Tsim
```

Calculate state transition

```
        stateVal = statechange(i);
        state = 1;
        for innerState = numStates-1:-1:1
            if stateVal > trc(currentstate,innerState)
                state = innerState + 1;
                break
            end
        end
```

Calculate emission

```
        val = randvals(i);
        emit = 1;
        for inner = numEmissions-1:-1:1
            if val > ec(state,inner)
                emit = inner + 1;
                break
            end
        end
```

Add values and states to output

```
        seq(i) = emit;
        states(i) = state;
        currentstate = state;

    end
```

Get controllers gains from decLMI obj

```
Kc=obj.K.ci;
Kd=obj.K.di;
Kl=obj.K.dl;
Ks=obj.K.ds;
```

Init states, inputs, state norms and input norms

```
Xc = x0;
Xd = x0;
Xn = x0;
Xl = x0;
Xs = x0;

Uc = [];
Ud = [];
Un = [];
Ul = [];
Us = [];

Xcnorm = [];
Xdnorm = [];
Xnnorm = [];
Xlnorm = [];
Xsnorm = [];

Ucnorm = [];
Udnorm = [];
Unnorm = [];
Ulnorm = [];
Usnorm = [];
```

Perform current simulation

```
for i=1:Tsim
```

Centralized

```
    Uc(:,i) = Kc*Xc(:,i);
    Xc(:,i+1) = A*Xc(:,i) + B*Uc(:,i);
    Xcnorm(i) = norm(Xc(:,i));
    Ucnorm(i) = norm(Uc(:,i));
```

Decentralized with ideal network

```
    Ud(:,i) = Kd*Xd(:,i);
    Xd(:,i+1) = A*Xd(:,i) + B*Ud(:,i);
    Xdnorm(i) = norm(Xd(:,i));
    Udnorm(i) = norm(Ud(:,i));
```

Decentralized with lossy network and robust stability

```
        Ul(:,i) = Kl{seq(i)}*Xl(:,i);
        Xl(:,i+1) = A*Xl(:,i) + B*Ul(:,i);
        Xlnorm(i) = norm(Xl(:,i));
        Ulnorm(i) = norm(Ul(:,i));
```

Decentralized with lossy network and stoch. stability

```
        Us(:,i) = Ks{seq(i),states(i)}*Xs(:,i);
        Xs(:,i+1) = A*Xs(:,i) + B*Us(:,i);
        Xsnorm(i) = norm(Xs(:,i));
        Usnorm(i) = norm(Us(:,i));

    end
```

Performance indices initialization

```
    Jc = [];
    Jd = [];
    Jl = [];
    Js = [];
```

The used performance index is a sum over the entire simulation horizon of the state and input norms weighted by the Riccati equation weights.

```
    for i=1:Tsim
        Jc(i) = sqrt(Xc(:,i)'*Qx*Xc(:,i) + Uc(:,i)'*Qu*Uc(:,i));
        Jd(i) = sqrt(Xd(:,i)'*Qx*Xd(:,i) + Ud(:,i)'*Qu*Ud(:,i));
        Jl(i) = sqrt(Xl(:,i)'*Qx*Xl(:,i) + Ul(:,i)'*Qu*Ul(:,i));
        Js(i) = sqrt(Xs(:,i)'*Qx*Xs(:,i) + Us(:,i)'*Qu*Us(:,i));
    end
```

store the current simulation results

```
    Sim.Jc(iter) = sum(Jc);
    Sim.Jd(iter) = sum(Jd);
    Sim.Jl(iter) = sum(Jl);
    Sim.Js(iter) = sum(Js);

end

disp(['Global results over ' num2str(Nsim) ' simulations:   ']);
disp(['Central. ideal performance: ',num2str(sum(Sim.Jc))]);
disp(['Decentr. ideal performance: ',num2str(sum(Sim.Jd))]);
disp(['Decentr. lossy performance: ',num2str(sum(Sim.Jl))]);
disp(['Decentr. stoch performance: ',num2str(sum(Sim.Js))]);

disp(['Central. ideal performance AVG: ',num2str(mean(Sim.Jc))]);
disp(['Decentr. ideal performance AVG: ',num2str(mean(Sim.Jd))]);
disp(['Decentr. lossy performance AVG: ',num2str(mean(Sim.Jl))]);
disp(['Decentr. stoch performance AVG: ',num2str(mean(Sim.Js))]);

disp(['Central. ideal performance STD: ',num2str(std(Sim.Jc))]);
disp(['Decentr. ideal performance STD: ',num2str(std(Sim.Jd))]);
disp(['Decentr. lossy performance STD: ',num2str(std(Sim.Jl))]);
```

```
disp(['Decentr. stoch performance STD: ',num2str(std(Sim.Js))]);

CPUtime
```

```
Global results over 50 simulations:
Central. ideal performance: 2048.6692
Decentr. ideal performance: 2251.9299
Decentr. lossy performance: 2487.4078
Decentr. stoch performance: 2351.7918
Central. ideal performance AVG: 40.9734
Decentr. ideal performance AVG: 45.0386
Decentr. lossy performance AVG: 49.7482
Decentr. stoch performance AVG: 47.0358
Central. ideal performance STD: 7.1776e-15
Decentr. ideal performance STD: 4.3065e-14
Decentr. lossy performance STD: 1.5547
Decentr. stoch performance STD: 1.4408

CPUtime =

    centralized: 4.2254
      dec_ideal: 1.1811
      dec_lossy: 10.3414
      dec_stoch: 32.3619
```

Conclusions

Performance evaluation

As expected, the best performance is achieved by the centralized ideal controller, as it can use all the state measurements for all actuators at all time steps.

However, the decentralized ideal controller achieves a performance which is only 10% higher than the ideal centralized one, while using much less communications between sensors and actuators.

A robustly stabilizing controller has been found also for the case where some of the links are subject to possible packet dropouts. In this case constraints are fulfilled at every time step, regardless of the occurrence of the dropouts in the network.

Exploiting a stochastic model of the packet dropouts, a stochastic controller has been computed which obtaines an improvement on the performance index of around 6% with respect to the robust control scheme.

2.5 dlincon

DMPC usage example
The main features of the Decentralized MPC class are showed in this file.

Contents

- Example description
- Example setup
- Comment to the results

Example description

The following example aims at showing the usage of the class dlincon in a simple but complete control problem in which the plant has some a relevant degree of decoupling between its states.

```
close all
clc
%
% The plant we are about to control is the LTI discrete-time system below
Ts=.1;
sys=ss([1.1 .1 0;.2 .3 0;0 .2 1.3],[1 0;0 0.1;0 1],eye(3),zeros(3,2),Ts);
%
% Nx, Nu and Ny are number of states, input and outputs, respectively
[Nx,Nu]=size(sys.B);
Ny=size(sys.c,1);
%
% Engineering insight suggest the following decentralization
dec(1).u=[1 2];
dec(1).x=[1 2];
dec(1).y=[1 2];
dec(1).applied=1;
dec(2).u=[1 2];
dec(2).x=[2 3];
dec(2).y=[2 3];
dec(2).applied=2;
```

Example setup

Below 4 simulations will be performed to show the potential of the class. In order:

- regulator with fixed bounds
- tracking with fixed bounds
- regulator with variant bounds
- tracking with variant bounds

Each simulation shows the behavior of the set of DMPC against the centralized controller that is used as reference for comparison. Both states and inputs are plotted.

Setup simulation parameters

```
x_c0=[ -0.4286;0.2182;-0.3596];
u_c0=zeros(Nu);
Tsim=1;
for j=1:4
```

```
Steup controllers parameters

if mod(j,2)==0
type='track'; % even values of j
else
type='reg';   % odd values of j
end
if strcmp(type,'reg')
    cost.Q=1e1*eye(Nx);
    cost.R=eye(Nu);
else
    cost.S=1e1*eye(Nx);
    cost.T=eye(Nu);
end
interval.N=10;
interval.Nu=5;
[A,B,C,D]=ssdata(sys);
var_bounds=(j>2);
if var_bounds
```

Nullify influence of static constraint on case of variant bounds. This is not mandatory, it is possible to have some bounds to be variant and some others to be fixed, and that is currently supported

```
k=inf;

else
k=.5;
end
limits.umin=-k*ones(Nu,1);
limits.umax=k*ones(Nu,1);
limits.ymin=-k*ones(Ny,1);
limits.ymax=k*ones(Ny,1);

    cost.rho=inf;

Dcon = dlincon(sys,type,cost,interval,limits,0,dec,var_bounds);

    xx=.1;
    range.xmin=-xx*ones(Nx,1);
    range.xmax=-range.xmin;

    if ~Dcon.var_bounds
        if Dcon.stability_test(range,cost)
            disp('Stability test succeded');
        else
            disp('Stability test failed');
        end
    end

    x_c=x_c0;
    x_d=x_c;
    u_c=u_c0;
    u_d=u_c;
disp('Start simulation')
```

```
Index exceeds matrix dimensions.

Error in ==> dlincon_example at 51
if mod(j,2)==0


% Setup variant bounds
bounds=[-.2;.2];
for i=2:Dcon.M
     bounds=[bounds; [-.2;.2]];
    end

% Reference signal
r=[3,3]';

for i=1:Tsim/Ts
    x_c(:,i+1) = A*x_c(:,i) + B*u_c(:,i);
    x_d(:,i+1) = A*x_d(:,i) + B*u_d(:,i);

    if var_bounds
            if mod(j,2)==1
                r=zeros(Nx+2*Nu,1);
            else
```

There are only two inputs, hence only 1st and 3rd state components can achieve zero-error convergence. 2*Nu zeros are added because of the variant bound that are mapped as fake outputs.

```
                r=[-.1;0;-.1;zeros(2*Nu,1)];

            end
            [u_c(:,i+1) Dcon] = Deval(Dcon,'global',[x_c(:,i+1) ; bounds],r);
            [u_d(:,i+1) Dcon] = Deval(Dcon,'Dglobal',[x_d(:,i+1) ; bounds],r);
  if  i==ceil(Tsim/Ts/2)

  % Vary the variant bounds
      bounds=bounds+.1;

  end
        else
            if mod(j,2)==1
                r=zeros(Nx,1);
            else
```

There are only two inputs, hence only 1st and 3rd state components can achieve zero-error convergence.

```
                r=[-.1;0;-.1];

            end
        [u_c(:,i+1) Dcon] = Deval(Dcon,'global',x_c(:,i+1),r);
        [u_d(:,i+1) Dcon] = Deval(Dcon,'Dglobal',x_d(:,i+1),r);
    end
    end
```

```
% Plot showing in red, green and blue color the first, second and third
% state components, respectively. Continuous lines depicts centralized
% controller trajectories while dash ones show decentralized set of
    % controllers behavior.
    figure;
t=0:Ts:Tsim;
subplot(2,1,1)
plot(t,x_c);
hold on
plot(t,x_d,'--');
hold off
title('States: cent(-), dec(--)');
subplot(2,1,2)
plot(t,u_c);
hold on
plot(t,u_d,'--');
hold off
title('Inputs: cent(-), dec(--)');

if j<4
disp('Press any key to test next configuration');
pause();
end

end
```

Comment to the results

Figures 1 and 3 depicts the regulator configuration with fixed and time varying bounds, respectively. Convergence in the second case is slower due to the more restrictive bound imposed. However in both cases convergence is achieved by both the centralized and set of decentralized controllers.

Figure 2 and 4 depicts the reference tracking configuration with fixed and time varying bounds, respectively. The more restrictive variant bounds make the latter case to need more time to achieve convergence with respect to the fixed bounds case. In particular figure 4 show the main difference between the centralized and decentralized controllers, that is the longer transient in the decentralized case.

2.6 eampc

Contents

- Energy-Aware MPC Demo
- Plant model
- Network model
- Controller design
- Simulation: energy-aware MPC vs standard MPC
- Results

- Conclusions

Energy-Aware MPC Demo

This demo illustrates the usage of the EAMPC class on a 2-states, 1-input linear system subject to state and output noise. Several sensor nodes are used for disturbance rejection purposes. We assume that every node measures the full state vector, corrupted by an additive disturbance. At every time step, once every node got a measurement, an estimated state value is obtained by taking the mean value of the outputs. This estimation is transmitted to the controller in accordance with a threshold-based policy, where the estimated state is compared with a predicted value which has been precomputed by the controller and transmitted beforehand to the sensors. This policy is inteded to reduce the number of communications between controller and sensors, hence saving sensor nodes battery. In this example the energy-aware MPC is compared with a traditional MPC scheme, where the measurements are simply transmitted to the controller at every time step, and no predictions are computed and transmitted by the controller.

by D. Bernardini, 2010.

```
close all
clc
```

Plant model

State-space matrices of the discrete-time LTI system

$$x(k+1) = Ax(k) + Bu(k)$$

```
Plant.A = [.21 -.39; -.39 .82];
Plant.B = [0 1]';
```

Network model

specify parameters for sensor nodes

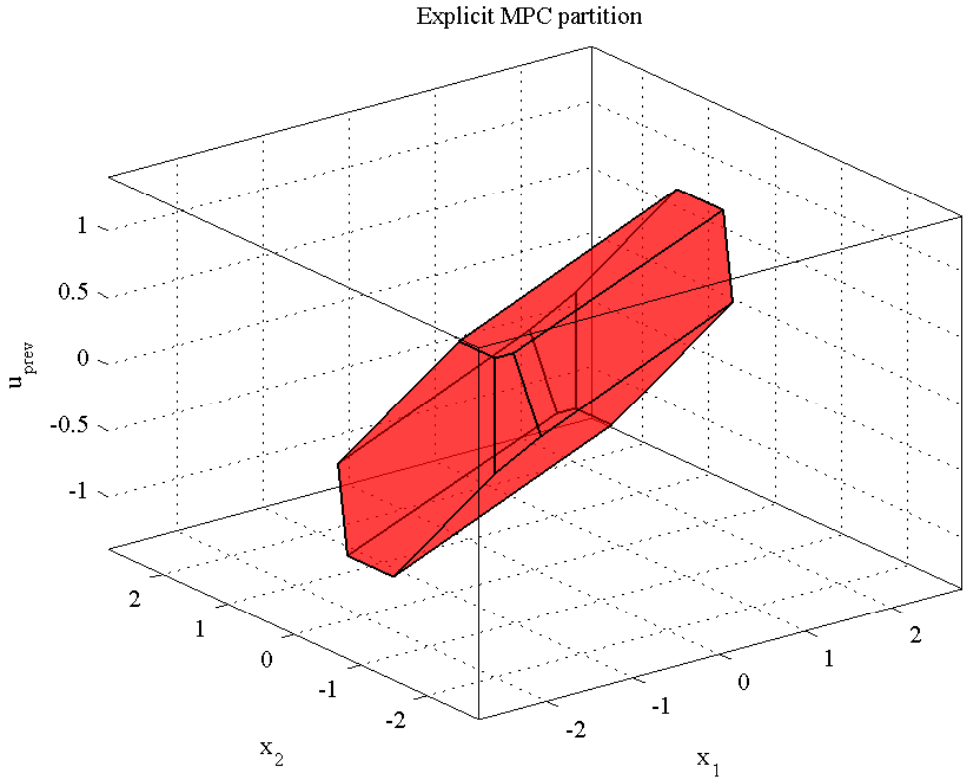
```
Net.nodes = 3; % number of wireless sensor nodes
Net.th = .05*ones(2,1); % transmission thresholds
Net.Ne = 10; % estimation horizon
```

Controller design

```
% element-wise constraints on output, input, and input rate
Limits.ymin = [-2 -2]'; % min output
Limits.ymax = [2 2]'; % max output
Limits.umin = -1;  % min input
```

```
Limits.umax = 1; % max input
Limits.dumin = -.6; % min input rate
Limits.dumax = .6; % max input rate

% weights matrices for MPC objective function
Weights.Qu = .1; % input weight
Weights.Qy = eye(size(Plant.A,1)); % output weight
Weights.Qn = Weights.Qy; % terminal weight
Weights.rho = Inf; % positive weight for soft output constraints
% (if rho=Inf then hard constraints are imposed)

% other parameters
Params.pnorm = 2; % norm used in the objective function (1, 2 or Inf).
Params.N = 10; % prediction horizon
Params.Nc = Params.N; % control horizon

% build EAMPC object
EAobj = eampc(Plant,Net,Limits,Weights,Params);

% plot explicit MPC partition
Options = mpt_options;
Options.shade = .5;
Options.samecolors = 1;
figure
plot(EAobj.Ctrl.Pn,Options);
xlabel 'x_1'
ylabel 'x_2'
zlabel 'u_{prev}'
axis(1.4*[Limits.ymin(1) Limits.ymax(1) ...
    Limits.ymin(2) Limits.ymax(2) Limits.umin Limits.umax]);
title('Explicit MPC partition')
box on


IBM ILOG License Manager: "IBM ILOG Optimization Suite for Academic Initiative" is accessing CPLE
Warning: mpt_init: No supported MIQP solver available on your system.

MPT toolbox 2.6.3 initialized...
Copyright (C) 2003-2006 by M. Kvasnica, P. Grieder and M. Baotic

Send bug reports, questions or comments to mpt@control.ee.ethz.ch
For news, visit the MPT web page at http://control.ee.ethz.ch/~mpt/
         LP solver: CPLEX-IBM
         QP solver: CPLEX-IBM
       MILP solver: CPLEX-IBM
       MIQP solver: CPLEX-IBM
Vertex enumeration: CDD

 Run 'mpt_studio' to start the GUI. Run 'mpt_setup' to set global parameters.

mpt_mpqp: 1 regions
-> Generated 1 partition.

Solution consists of 1 regions
```

```
1 regions with 1 different control laws
control law # 1, 1 region --> 1 region
controller partition reduced to 1 regions
Plotting...
```



Simulation: energy-aware MPC vs standard MPC

The energy-aware MPC controller is compared with a standard MPC control scheme, where at every time step measurements are transmitted to the controller regardless to the threshold logic, and no predictions are transmitted to the sensors.

The two controllers are simulated in closed-loop with the LTI system

$$x(k+1) = Ax(k) + Bu(k) + w(k)$$

where $|w(k)| \leq w_{max},\ \forall k$. Moreover, measurements are affected by an additive error, i.e.,

$$|y^i(k) = x(k) + v^i(k)|$$

for all $i = 1, 2, ...,$`Net.nodes`, where $|y^i|$ is the output measured by the `i`-th node, and $|v^i(k)| \leq v_{max}$, for all k, i.

```
% build EAMPC object for standard MPC
MPCobj = EAobj; % copy EAMPC object
MPCobj.Net.th = zeros(MPCobj.ny,1); % set thresholds to zero

% simulation length
Tsteps = 20;

% disturbance realizations
loadNoise = 1; % 0: pick random noise, 1: load noise data form Noise.mat
if loadNoise==0

    % additive disturbance bounds
    wmax = [.03 .03]';

    % output noise bounds for each output and each node
    vmax = kron([.06 .06]',ones(1,Net.nodes));

    % additive disturbance realizations
    W = diag(2*wmax)*rand(EAobj.ny,Tsteps)-wmax*ones(1,Tsteps);

    % output noise realizations
    V = zeros(EAobj.ny,EAobj.Net.nodes,Tsteps);
    for j=1:EAobj.Net.nodes
        V(:,j,:) =  diag(2*vmax(:,j))*rand(EAobj.ny,Tsteps) ...
            -kron(ones(1,Tsteps),vmax(:,j));
    end

    % initial state (randomly picked in the feasible state set)
    vert = extreme(projection(EAobj.Ctrl.Pfinal,1:EAobj.ny));
    comb = rand(size(vert,1),1);
    comb = comb/sum(comb);
    x0 = (comb'*vert)';
    save Noise wmax vmax W V x0
else
    load Noise.mat
end

% init energy-aware MPC simluation variables
X = x0; % state
Xestim = []; % estimated state
U = []; % input
Uprev = zeros(EAobj.nu,1); % previous input
J = []; % experimental cost function
EAobj = EAobj.init_sim(); % init simulation data

% init standard MPC simluation variables
XX = x0; % state
XXestim = []; % estimated state
UU = []; % input
UUprev = zeros(MPCobj.nu,1); % previous input
JJ = []; % experimental cost function
MPCobj = MPCobj.init_sim(); % init simulation data

% compute first sequence of output predictions and transmit them to sensors
```

```
% (we assume that the initial state is known)
EAobj = EAobj.send_predictions(X,Uprev);
MPCobj = MPCobj.send_predictions(XX,UUprev);
% note: since MPCobj.Net.th = 0, predictions are not actually transmitted here

% run simulation for k=1,2,...,Tsteps
for k=1:Tsteps

    % measurements acquisition
    [xestim,EAobj] = EAobj.get_measurements(X(:,end),V(:,:,k));
    Xestim(:,end+1) = xestim;
    [xestim,MPCobj] = MPCobj.get_measurements(XX(:,end),V(:,:,k));
    XXestim(:,end+1) = xestim;

    % compute the control law
    U(:,end+1) = EAobj.get_input(Xestim(:,end),Uprev);
    UU(:,end+1) = MPCobj.get_input(XXestim(:,end),UUprev);

    % if needed, compute new predictions
    EAobj = EAobj.send_predictions(Xestim(:,end),Uprev);
    MPCobj = MPCobj.send_predictions(XXestim(:,end),UUprev);

    % evaluate evperimental costs
    if EAobj.Params.pnorm == 2
        J(k) = X(:,end)'*Weights.Qy*X(:,end) + ...
            U(:,end)'*Weights.Qu*U(:,end);
    else
        J(k) = norm(Weights.Qy*X(:,end),EAobj.Params.pnorm) + ...
            norm(Weights.Qu*U(:,end),EAobj.Params.pnorm);
    end
    if MPCobj.Params.pnorm == 2
        JJ(k) = XX(:,end)'*Weights.Qy*XX(:,end) + ...
            UU(:,end)'*Weights.Qu*UU(:,end);
    else
        JJ(k) = norm(Weights.Qy*XX(:,end),MPCobj.Params.pnorm) + ...
            norm(Weights.Qu*UU(:,end),MPCobj.Params.pnorm);
    end

    % state evolution
X(:,end+1) = Plant.A*X(:,end) + Plant.B*U(:,end) +  W(:,k);
    XX(:,end+1) = Plant.A*XX(:,end) + Plant.B*UU(:,end) +  W(:,k);

    % update previous input
    Uprev = U(:,end);
    UUprev = UU(:,end);

end
```

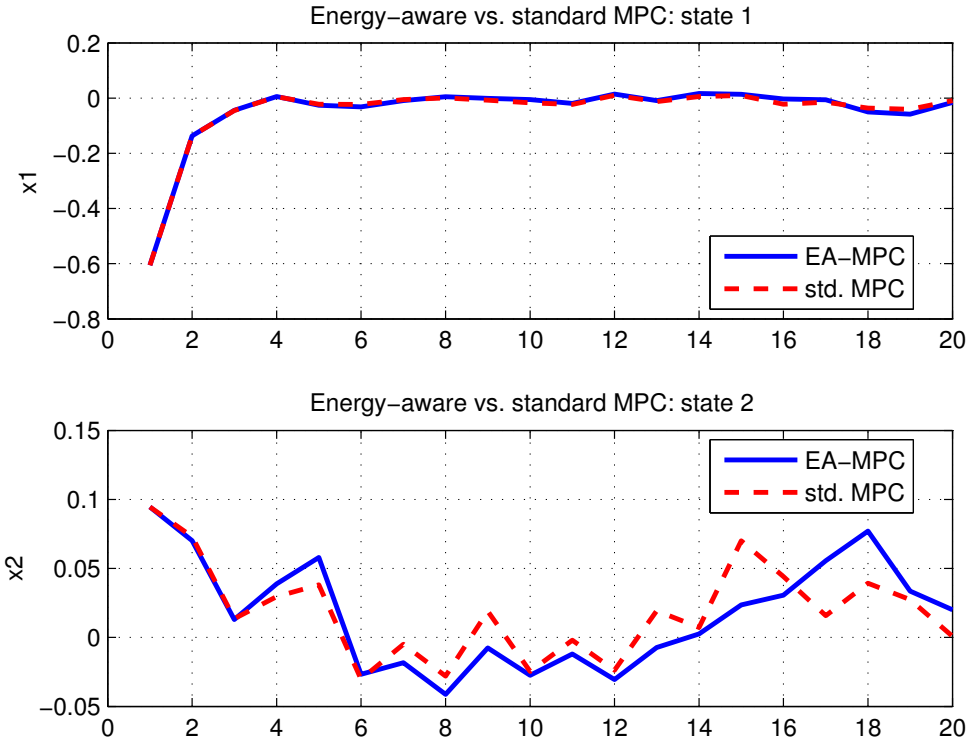
Results

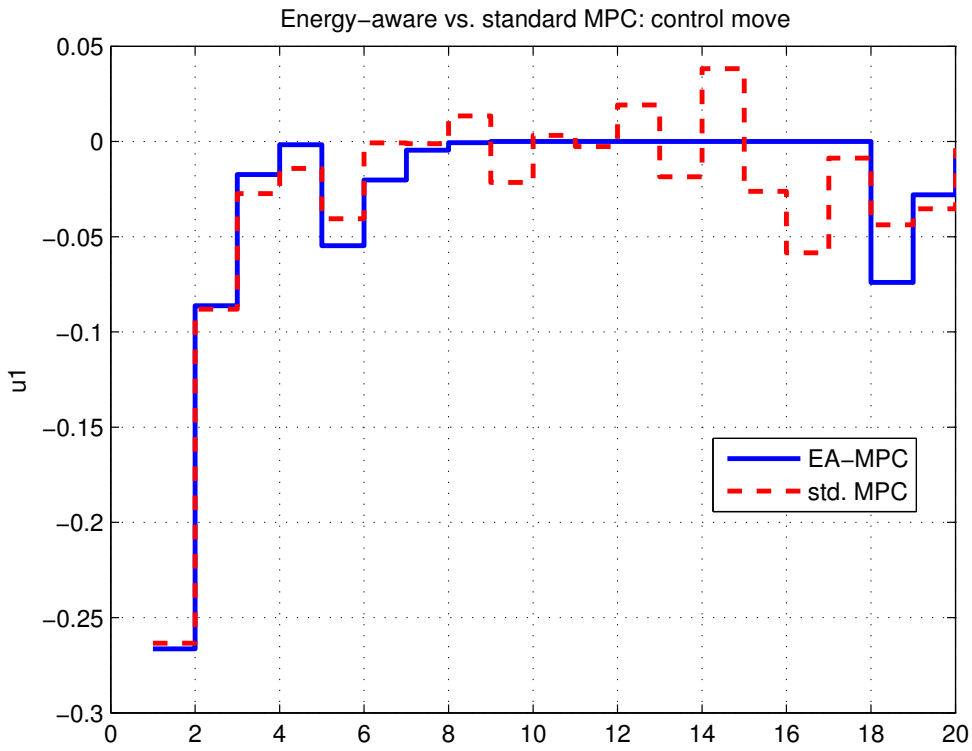
```
% plot state trajectory
figure;
```

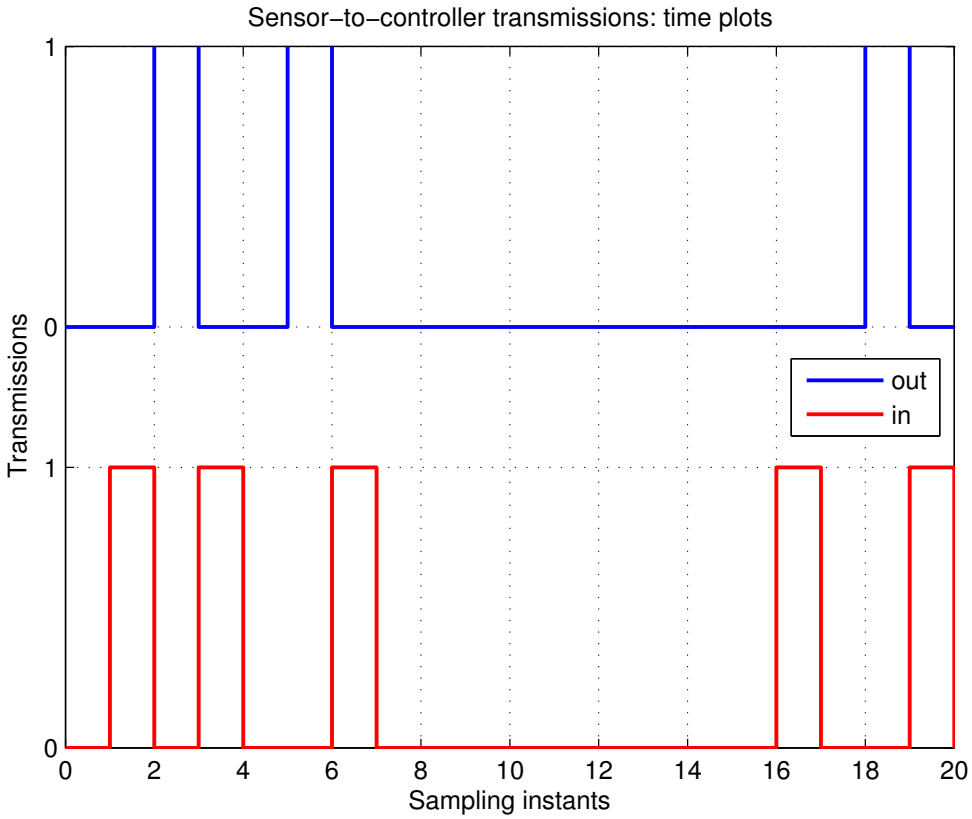
```
subplot(2,1,1);
plot(1:Tsteps,X(1,1:Tsteps),'b',1:Tsteps,XX(1,1:Tsteps),'r--','LineWidth',2);
title('Energy-aware vs. standard MPC: state 1')
ylabel 'x1'
legend('EA-MPC','std. MPC','Location','Best')
grid on
subplot(2,1,2);
plot(1:Tsteps,X(2,1:Tsteps),'b',1:Tsteps,XX(2,1:Tsteps),'r--','LineWidth',2);
title('Energy-aware vs. standard MPC: state 2')
ylabel 'x2'
legend('EA-MPC','std. MPC','Location','Best')
grid on

% plot input trajectory
figure
stairs(1:Tsteps,U(1,1:Tsteps),'b','LineWidth',2)
hold on
stairs(1:Tsteps,UU(1,1:Tsteps),'r--','LineWidth',2)
title('Energy-aware vs. standard MPC: control move')
ylabel 'u1'
legend('EA-MPC','std. MPC','Location','Best')
grid on

% plot WSN activity
figure
stairs(0:Tsteps,[0 EAobj.Sim.tx(1:Tsteps)]+1.5,'b','LineWidth',1.5);
hold on
stairs(0:Tsteps,[0 EAobj.Sim.rx(1:Tsteps)],'r','LineWidth',1.5);
set(gca,'Ytick',[0 1 1.5 2.5],'YTickLabel',{'0' '1' '0' '1'});
title('Sensor-to-controller transmissions: time plots')
xlabel 'Sampling instants'
ylabel 'Transmissions'
legend('out','in','Location','Best')
grid on

% plot pie graph for transmission data
figure
tx_perc = 100*sum(EAobj.Sim.tx)/Tsteps;
rx_perc = 100*sum(EAobj.Sim.rx)/Tsteps;
free_perc = 100-tx_perc-rx_perc;
pie([tx_perc free_perc rx_perc],[0 1 0], ...
    {['Outgoing: ' num2str(tx_perc,'%3.1f') '%'], ...
    ['No transmissions: ' num2str(free_perc,'%3.1f') '%'], ...
    ['Incoming: ' num2str(rx_perc,'%3.1f') '%']});
title('Sensor-to-controller transmissions: overall statistics')

% print experimental costs
sumJ = sum(J);
sumJJ = sum(JJ);
fprintf('\n== Results for Energy-Aware MPC:\n');
fprintf('threshold TH = [%.2f %.2f]'', experimental cost J = %.3f.\n', ...
    EAobj.Net.th',sumJ);
fprintf('\n== Results for standard MPC:\n');
fprintf('threshold TH = [%.2f %.2f]'', experimental cost J = %.3f.\n', ...
```

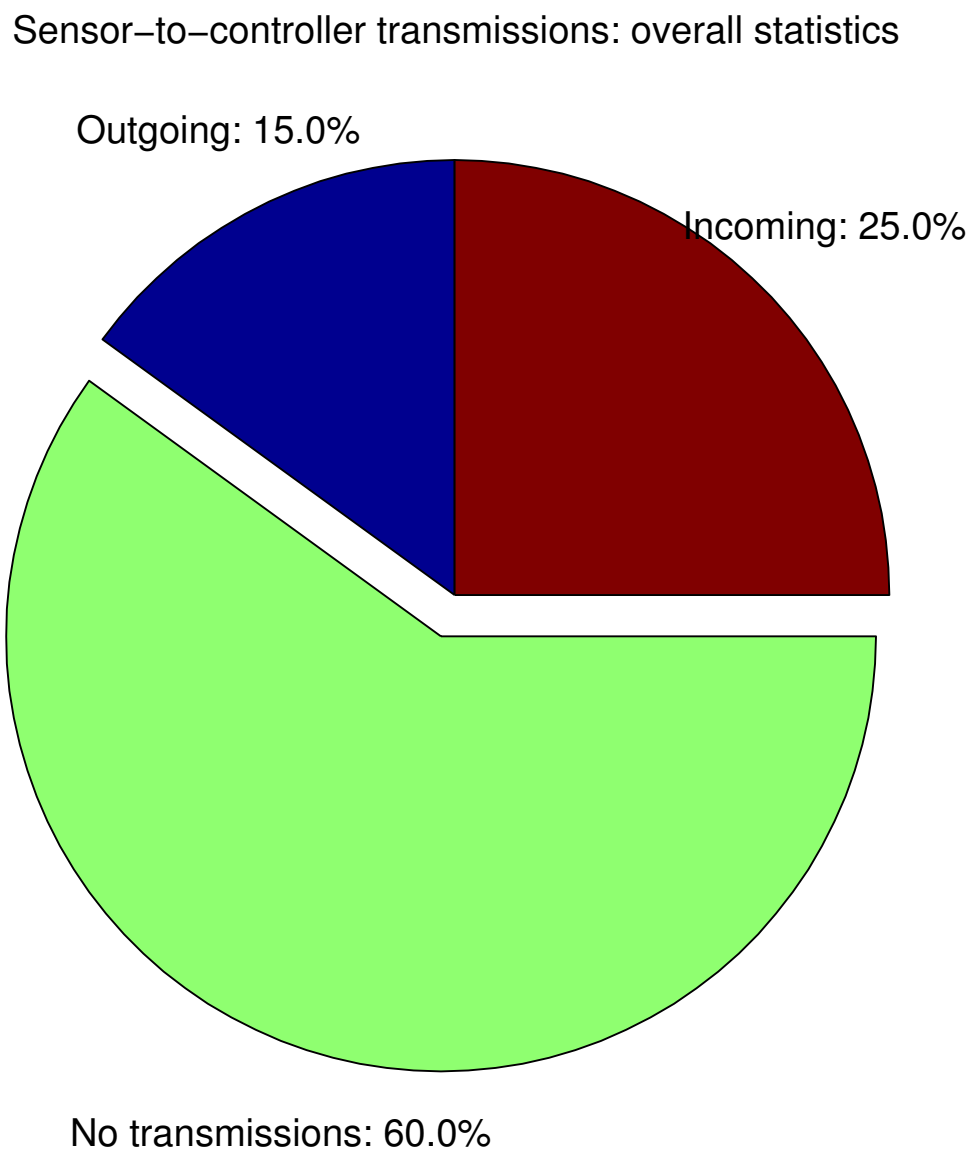
```
    MPCobj.Net.th',sumJJ);
fprintf('\n== Energy-Aware vs standard MPC: transmission savings = %.2f%%\n', ...
    free_perc);
fprintf('\n== Energy-Aware vs standard MPC: performance ratio = %.2f%%\n', ...
    (sumJ/sumJJ-1)*100);
```

```
== Results for Energy-Aware MPC:
threshold TH = [0.05 0.05]', experimental cost J = 0.441.

== Results for standard MPC:
threshold TH = [0.00 0.00]', experimental cost J = 0.432.

== Energy-Aware vs standard MPC: transmission savings = 60.00%

== Energy-Aware vs standard MPC: performance ratio = 1.98%
```




Energy−aware vs. standard MPC: control move
u1
0.05
0
−0.05
−0.1
−0.15
−0.2
−0.25
−0.3
0
2
4
6
8
10
12
14
16
18
20
EA−MPC
std. MPC


Sensor−to−controller transmissions: time plots
Transmissions
1
0
1
0
0
2
4
6
8
10
12
14
16
18
20
Sampling instants
out
in



Conclusions

In the proposed example, the energy-aware MPC scheme provided a reduction of 60% in the number of transmissions between the controller and the sensor nodes. This energy savings have been obtained with a small loss in the performance index (around 2%), with respect to a traditional MPC scheme. Indeed, the threshold values `Net.th` need to be properly tuned, especially in function of the magnitude of the disturbances, in order to obtain a valuable reduction in the transmission rate and also achieve a satisfying performance.

2.7 himpc

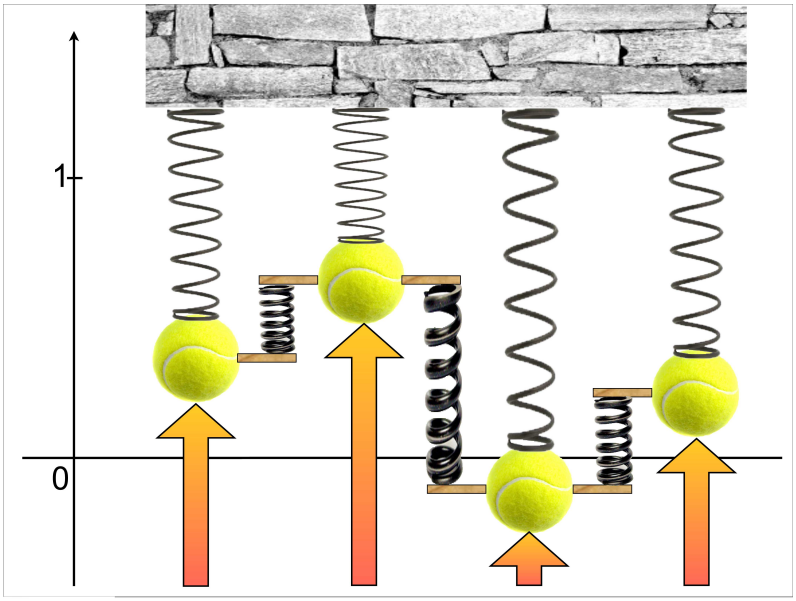
Example file for class HiMPC

In this demo we use the HiMPC toolbox to show how to build a decentralized hierarchical controller for linear systems subject to linear constraints on input and output variables.

Contents

Description

For this demo we use the following system consting of four balls and seven springs.



The process is composed by four masses moving vertically, each one connected by a spring to a fixed ceiling, subject to damping due to viscous friction with the environment, and connected to its neighbor mass by another spring. An input force u [Nm] can be applied to each mass by the lower-level controller. The state of the system is the vector x collecting the vertical positions and accelerations of the masses, while the output y is the vector that collects the positions. The constrained masses-springs system is an extension of the example presented [1]. However in this

demo both layers have a decentralized structure, in particular four controllers at the higher level independently process the user provided references and command their respective actuators.

The final objective of this class functions is to compute the maximum reference variation, element-wise, that the higher controller can apply in order to guarantee the system stability while fulfilling the constraints. Such goal is achieved by ensuring that the system state vector starting in a Maximal Output Admissible Set (MOAS) will be in a new MOAS at the next execution of the higher level controller. The MOAS is obtained by tightening the output constraints by a fixed value, Δy that is a tuning knob of the approach. The computation of maximum reference variation is an optimization problem, which can be conveniently casted as a Mixed Integer Linear Program (MILP), and solved via suitable solvers (we strongly suggest IBM, former ILOG, Cplex).

The simulations are performed using at the higher layer a set of independent hierarchical MPC (HiMPC) which adopt a linear formulation. The upper-layer controllers must embed constraints on the generated references, to ensure stability and constraint satisfaction. In this example we impose constraints on plant outputs, namely the positions of the balls depicted in figure.

[1] D. Barcelli, A. Bemporad and G. Ripaccioli, *Hierarchical MPC controller design for LTI systems*, 49th Control and Decision Conference, Atlanta, Georgia, USA, 2010.

Simulation Example

```
close all
clc

% Computing data is very time consiming, set the variable below to 0 in
% case you want to recompute
precomputed_data=0;
showPlot=1;
```

Plant model

Sample time

```
TL=.25;
```

Creates the closed-loop model of the plant with the lower layer controller

```
[sys N]=buildModelFriction(TL);
[A B C D] = ssdata(sys);
```

Number of states and inputs respectively

```
[Nx Ny]=size(B);
%
```

Element-wise state constraints

The output constraints have to be expressed in state coordinates in order to compute the MOAS. To avoid numerical problems the state constraints polytope is bounded, hence the speed is limited to 10 [m/s]

```
Xcon.min=[];
for i=1:N,Xcon.min=[Xcon.min;-.3;-10];end
Xcon.max=[];
for i=1:N,Xcon.max=[Xcon.max;1;10];end
%
```

Coupled constraints

mass #2 position is forbidden to go below mass #1 position more than 0.3

```
coupledCons(1).H=[-1 0 1 0];
coupledCons(1).K=[0.3];
coupledCons(2).H=[];
coupledCons(2).K=[];
coupledCons(3).H=[];
coupledCons(3).K=[];
%
```

The decentralization of the model is defined as:

```
dec(1).x=1:4;
dec(1).y=1:2;
dec(1).u=1:2;
dec(1).applied=1:2;
dec(2).x=[5 6];
dec(2).y=3;
dec(2).u=3;
dec(2).applied=3;
dec(3).x=[7 8];
dec(3).y=4;
dec(3).u=4;
dec(3).applied=4;
```

Δk : Tightening of the output admissible set. Tuning knob.

```
DeltaK{1}=[.3;2;.3;2;.3;2;.3;2;0.12];
DeltaK{2}=[.3;2;.3;2];
DeltaK{3}=[.3;2;.3;2];
```

Truncated DCgain

In order to have a decentralized structure stabilizing feedback must retain the structure, hence we simply nulify out-of-structure entries

```
E=inv(dcgain(sys));
nd=length(dec);
for i=1:nd
    % for each subsystem
    for h=dec(i).u
        for j=setdiff(1:4,dec(i).u)
            E(h,j)=0;
        end
    end
end
sys.B=sys.B*E;
```

```
if ~precomputed_data
```

Create the object, i.e. the HiMPC calss instance

```
    dobj = HiMPC(sys,dec,Xcon,DeltaK,coupledCons);
```

Compute the MOAS. For this purpose each sub-model is considered independent and subject to an unknown but bounded disturbance. The disturbance models the influence of the unmodeled dynamics, i.e. the state components that are not present in that sub-system. The uncertainty polytope is computed in an exact manner by considering the range of each unmodeled state component and its influence with respect to state components which belong to the current sub-model. It follows that the invariant set is actually a Maximal Output Admissible Robust Set.

```
    dobj=dobj.computeMOARS();
```

```
Computing disturbance polytope for subsystem 1
Done
Pnoise=

Normalized, minimal representation polytope in R^4
         H: [12x4 double]
         K: [12x1 double]
    normal: 1
    minrep: 1
     xCheb: [4x1 double]
     RCheb: 1.0013e-06

[        -0          -0           1   -0.002004]      [1.002e-06]
[        -0           1          -0   -0.002004]      [1.002e-06]
[         1          -0          -0   -0.002004]      [1.002e-06]
[        -0          -0          -1   -0.002004]      [1.002e-06]
[        -0          -1          -0   -0.002004]      [1.002e-06]
[        -1          -0          -0   -0.002004]      [1.002e-06]
[        -1 -4.3455e-19  -8.691e-19    0.002004] x <= [1.002e-06]
```

```
[4.5566e-19          -1  -8.691e-19    0.002004]      [1.002e-06]
[9.1132e-19 -4.3455e-19          -1    0.002004]      [1.002e-06]
[4.5566e-19           1  -8.691e-19    0.002004]      [1.002e-06]
[         1  4.5566e-19  -8.691e-19    0.002004]      [1.002e-06]
[9.1132e-19 -4.3455e-19           1    0.002004]      [1.002e-06]

Computing disturbance polytope for subsystem 2
Done
Pnoise=

Normalized, minimal representation polytope in R^2
         H: [4x2 double]
         K: [4x1 double]
    normal: 1
    minrep: 1
     xCheb: [2x1 double]
     RCheb: 1.0005e-06

[ 1   -0.001001]      [1.001e-06]
[-1   -0.001001]      [1.001e-06]
[-1    0.001001] x <= [1.001e-06]
[ 1    0.001001]      [1.001e-06]

Computing disturbance polytope for subsystem 3
Done
Pnoise=

Normalized, minimal representation polytope in R^2
         H: [4x2 double]
         K: [4x1 double]
    normal: 1
    minrep: 1
     xCheb: [2x1 double]
     RCheb: 1.0010e-06

[ 1   -0.002004]      [1.002e-06]
[-1   -0.002004]      [1.002e-06]
[-1    0.002004] x <= [1.002e-06]
[ 1    0.002004]      [1.002e-06]

Computing the MOARS for subsystem 1
Oinf=

Normalized, minimal representation polytope in R^4
         H: [93x4 double]
         K: [93x1 double]
    normal: 1
    minrep: 1
     xCheb: [4x1 double]
     RCheb: 0.1511


Done
Computing the MOARS for subsystem 2
```

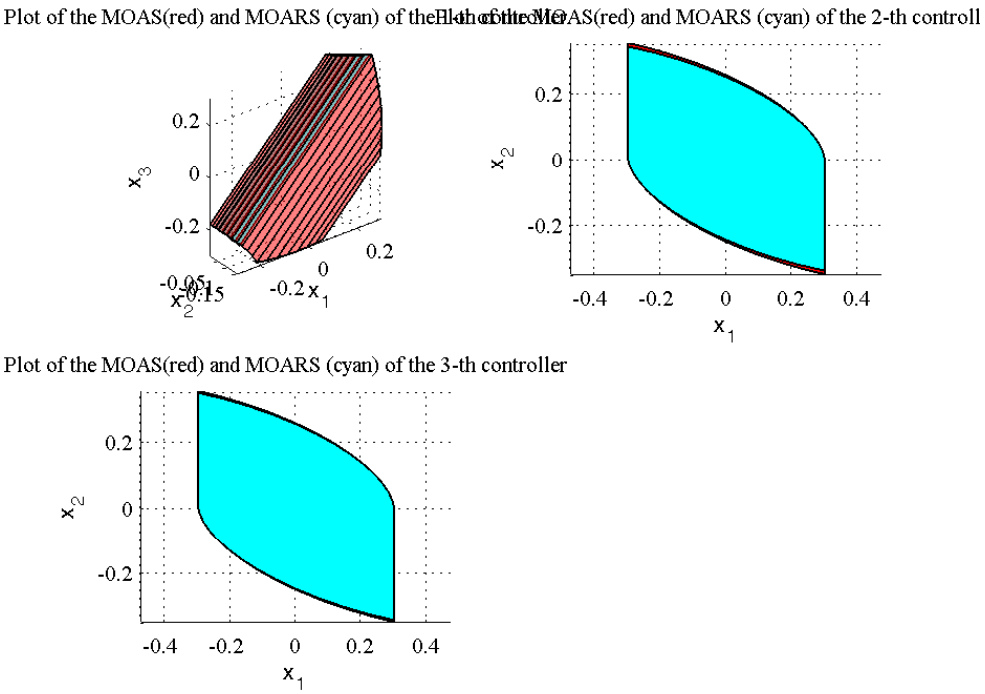
```
Oinf=

Normalized, minimal representation polytope in R^2
         H: [28x2 double]
         K: [28x1 double]
    normal: 1
    minrep: 1
     xCheb: [2x1 double]
     RCheb: 0.2231

[-0.22586    -0.97416]      [0.26619]
[ 0.22586     0.97416]      [0.26619]
[ -0.2774    -0.96075]      [0.25064]
[  0.2774     0.96075]      [0.25064]
[-0.32786    -0.94473]      [0.23894]
[ 0.32786     0.94473]      [0.23894]
[-0.37832    -0.92567]      [0.23061]
[ 0.37832     0.92567]      [0.23061]
[-0.42985     -0.9029]      [0.22536]
[ 0.42985      0.9029]      [0.22536]
[-0.48352    -0.87533]      [0.22307]
[ 0.48352     0.87533]      [0.22307]
[ -0.5405    -0.84135]      [0.22371]
[  0.5405     0.84135]      [0.22371]
[  -0.602    -0.79849] x <= [0.22742]
[   0.602     0.79849]      [0.22742]
[ -0.6692    -0.74308]      [0.23439]
[  0.6692     0.74308]      [0.23439]
[-0.74282    -0.66949]      [0.24485]
[ 0.74282     0.66949]      [0.24485]
[-0.82209    -0.56936]      [0.25874]
[ 0.82209     0.56936]      [0.25874]
[-0.90227    -0.43117]      [0.27515]
[ 0.90227     0.43117]      [0.27515]
[-0.97014    -0.24254]      [0.29104]
[ 0.97014     0.24254]      [0.29104]
[      -1           -0]      [    0.3]
[       1            0]      [    0.3]

Done
Computing the MOARS for subsystem 3
Oinf=

Normalized, minimal representation polytope in R^2
         H: [28x2 double]
         K: [28x1 double]
    normal: 1
    minrep: 1
     xCheb: [2x1 double]
     RCheb: 0.2251

[-0.22586    -0.97416]      [0.27129]
[ 0.22586     0.97416]      [0.27129]
[ -0.2774    -0.96075]      [0.25494]
```

```
[  0.2774     0.96075]       [0.25494]
[-0.32786    -0.94473]       [0.24255]
[ 0.32786     0.94473]       [0.24255]
[-0.37832    -0.92567]       [0.23361]
[ 0.37832     0.92567]       [0.23361]
[-0.42984     -0.9029]       [0.22784]
[ 0.42984      0.9029]       [0.22784]
[-0.48352    -0.87533]       [0.22507]
[ 0.48352     0.87533]       [0.22507]
[ -0.5405    -0.84135]       [0.22529]
[  0.5405     0.84135]       [0.22529]
[  -0.602    -0.79849] x <= [0.22862]
[   0.602     0.79849]       [0.22862]
[ -0.6692    -0.74308]       [0.23526]
[  0.6692     0.74308]       [0.23526]
[-0.74282    -0.66949]       [0.24541]
[ 0.74282     0.66949]       [0.24541]
[-0.82209    -0.56936]       [0.25905]
[ 0.82209     0.56936]       [0.25905]
[-0.90227    -0.43117]       [0.27526]
[ 0.90227     0.43117]       [0.27526]
[-0.97014    -0.24254]       [0.29104]
[ 0.97014     0.24254]       [0.29104]
[      -1          -0]       [    0.3]
[       1           0]       [    0.3]

Done
```

Plot both the MOAS with (blue) and without (red) the influence of the other sub-models for each sub-model

```
dobj.plotMOARS();
```

```
Plotting...
PLOT: Problem detected. Try to change value of abs_tol in mpt_init. 1e-7 should be a good value.
PLOT: Problem detected. Most probably extreme point enumeration failed for 76 polytopes...
Try to change value for extreme_solver.
```



Solve the MILP for values of sample-time ratio from 1 to 50 and, accordingly to the ratio, compute the maximal allowable Δr

```
dobj=dobj.computeDeltaR();
```

```
Time needed for the 1-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=7.1713e-05
Time needed for the 2-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=7.2957e-05
Time needed for the 3-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=7.5416e-05
Time needed for the 4-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0
Time needed for the 5-th problem of the 1 subsystem
```

```
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=8.434e-05
Time needed for the 6-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0
Time needed for the 7-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00010032
Time needed for the 8-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00011223
Time needed for the 9-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0
Time needed for the 10-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0
Time needed for the 11-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0
Time needed for the 12-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0
Time needed for the 13-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0
Time needed for the 14-th problem of the 1 subsystem
```

```
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00041293
Time needed for the 15-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.0006755
Time needed for the 16-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.0013508
Time needed for the 17-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.0015762
Time needed for the 18-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.0011374
Time needed for the 19-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.0006694
Time needed for the 20-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00049531
Time needed for the 21-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00040737
Time needed for the 22-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00035672
Time needed for the 23-th problem of the 1 subsystem
```

```
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00032598
Time needed for the 24-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00030749
Time needed for the 25-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00029746
Time needed for the 26-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00029388
Time needed for the 27-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00029574
Time needed for the 28-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00030261
Time needed for the 29-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00031453
Time needed for the 30-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00033189
Time needed for the 31-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00035554
Time needed for the 32-th problem of the 1 subsystem
```

```
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00038682
Time needed for the 33-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00042784
Time needed for the 34-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00048182
Time needed for the 35-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00055385
Time needed for the 36-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00065234
Time needed for the 37-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.0039051
Time needed for the 38-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.02138
Time needed for the 39-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.065297
Time needed for the 40-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.16856
Time needed for the 41-th problem of the 1 subsystem
```

```
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
...DeltaR=0

----------


Time needed for the 1-th problem of the 2 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.000101
Time needed for the 2-th problem of the 2 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0
Time needed for the 3-th problem of the 2 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0
Time needed for the 4-th problem of the 2 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0
Time needed for the 5-th problem of the 2 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0
Time needed for the 6-th problem of the 2 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00012853
Time needed for the 7-th problem of the 2 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0
Time needed for the 8-th problem of the 2 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
```

```
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0
Time needed for the 9-th problem of the 2 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0
Time needed for the 10-th problem of the 2 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0
Time needed for the 11-th problem of the 2 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0
Time needed for the 12-th problem of the 2 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0
Time needed for the 13-th problem of the 2 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0
Time needed for the 14-th problem of the 2 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0
Time needed for the 15-th problem of the 2 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.050564
Time needed for the 16-th problem of the 2 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.3395
Time needed for the 17-th problem of the 2 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
```

```
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.56935
Time needed for the 18-th problem of the 2 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.56801
Time needed for the 19-th problem of the 2 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.56647
Time needed for the 20-th problem of the 2 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.56474
Time needed for the 21-th problem of the 2 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.58746
Time needed for the 22-th problem of the 2 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.62624
Time needed for the 23-th problem of the 2 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.6761
Time needed for the 24-th problem of the 2 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
...DeltaR=0

----------


Time needed for the 1-th problem of the 3 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
```

```
DeltaR=0.000101
Time needed for the 2-th problem of the 3 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0
Time needed for the 3-th problem of the 3 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0
Time needed for the 4-th problem of the 3 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0
Time needed for the 5-th problem of the 3 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0
Time needed for the 6-th problem of the 3 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00012853
Time needed for the 7-th problem of the 3 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00014132
Time needed for the 8-th problem of the 3 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00015811
Time needed for the 9-th problem of the 3 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0
Time needed for the 10-th problem of the 3 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
```

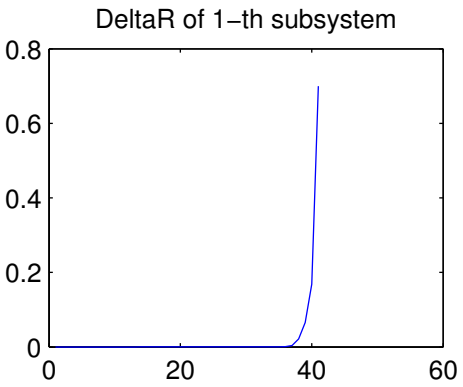
```
DeltaR=0
Time needed for the 11-th problem of the 3 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0
Time needed for the 12-th problem of the 3 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0
Time needed for the 13-th problem of the 3 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0
Time needed for the 14-th problem of the 3 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.006773
Time needed for the 15-th problem of the 3 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.067398
Time needed for the 16-th problem of the 3 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.40157
Time needed for the 17-th problem of the 3 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.60822
Time needed for the 18-th problem of the 3 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.60752
Time needed for the 19-th problem of the 3 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
```

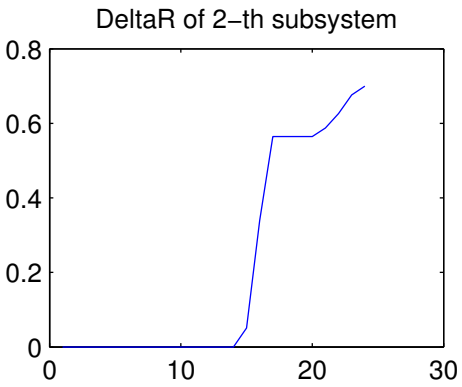
```
DeltaR=0.60673
Time needed for the 20-th problem of the 3 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.60586
Time needed for the 21-th problem of the 3 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.62533
Time needed for the 22-th problem of the 3 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.66263
Time needed for the 23-th problem of the 3 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
...DeltaR=0

----------
```

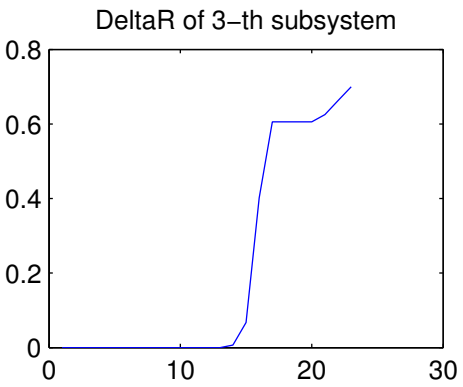
Plot both $\Delta r(N)$ as a function of N, that is the higher-lower sample times ratio, and $\Delta r(N)/N$ for each sub-model

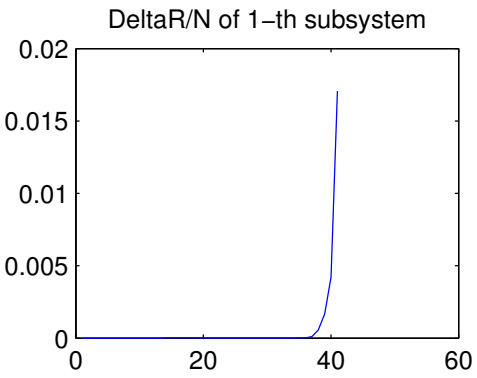
```
    dobj.plotDeltaR();
```

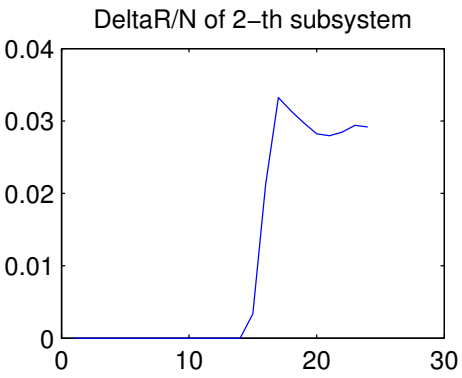
```
else
   load dobj_data;
end
```

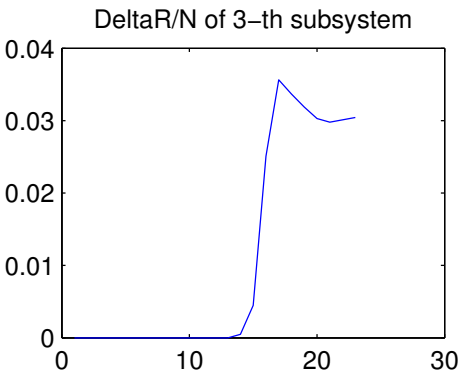

DeltaR of 1−th subsystem
0.8
0.6
0.4
0.2
0
0
20
40
60


DeltaR of 2−th subsystem
0.8
0.6
0.4
0.2
0
0
10
20
30


DeltaR of 3−th subsystem
0.8
0.6
0.4
0.2
0
0
10
20
30


DeltaR/N of 1−th subsystem
0.02
0.015
0.01
0.005
0
0
20
40
60


DeltaR/N of 2−th subsystem
0.04
0.03
0.02
0.01
0
0
10
20
30


DeltaR/N of 3−th subsystem
0.04
0.03
0.02
0.01
0
0
10
20
30

Pre-simulation considerations

The first and the forth sub-models MOAS are completely equivalent as well as the corresponding $\Delta r(N)$ and $\Delta r(N)/N$. The same consideration can be extended to the second and third sub-models. This is motivated by the internal symmetry of both the plant and the decentralization.

Higher level controller setup

In this section we define the higher layer controllers. We propose a model predictive control (MPC) design strategy for such a layer. The function lincon from the hybrid toolbox is used to perform that task.

```
warning off
```

Centralized HiMPC for comparison

```
cdec(1).x=1:8;
cdec(1).y=1:4;
cdec(1).u=1:4;
cdec(1).applied=1:4;
ccoupledCons(1).H=[-1 0 1 0 0 0 0 0];
ccoupledCons(1).K=[.3];
cDeltaK{1}=[.3;2;.3;2;.3;2;.3;2;.3;2;.3;2;.3;2;.3;2;0.12];
csys=ss(A,B,C,D,TL);
cdcg=dcgain(csys);
csys.B=csys.B*inv(cdcg);
if ~precomputed_data
    cobj = HiMPC(csys,cdec,Xcon,cDeltaK,ccoupledCons);
    cobj=cobj.computeMOARS();
    cobj=cobj.computeDeltaR();
else
    load cobj_data;
end
[sup ind] = max(cobj.DrN{1});
cdrOpt=cobj.Dr{1}(ind);
cTH = ind*TL;
cc=[csys.c;zeros(17,8)];
dd=[csys.d;cobj.Hr{1}*cdcg];
cmodel=ss(A,B,cc,dd,TL);

nDec=length(dec);
dcg={};
for i=1:nDec
```

High level sample time is chosen guaranteeing the maximal allowable slope for the references, as the maximum of the ratio $\Delta r(N)/N$

```
    [sup ind] = max(dobj.DrN{i});
    drOpt=dobj.Dr{i}(ind);
```

compute the Higher level sample-time

```
TH(i) = ind*TL;
```

Sub-models definition

```
Ai=A(dec(i).x,dec(i).x);
Bi=B(dec(i).x,dec(i).u);
Ci=C(dec(i).y,dec(i).x);
Di=D(dec(i).y,dec(i).u);

%[Ai Bi Ci Di]=ssdata(d2d(ss(Ai,Bi,Ci,Di,TL),TH(i)));
%Ai=round(Ai*1e3)/1e3;
%Bi=round(Bi*1e3)/1e3;

nx=length(dec(i).x);
nu=length(dec(i).u);
ny=length(dec(i).y);
```

State and reference polytopes (MOAS) to be imposed in order to guarantee stability

```
Ko=dobj.Ko;
Ho=dobj.Ho;
Hx=dobj.Hx;
K=dobj.K;
Hr=dobj.Hr;
Kr=dobj.Kr;
nco(i)=length(Ko{i});
ncu(i)=length(Kr{i});
ncx(i)=length(K{i});

% Ci=[Ci;Ho{i};zeros(ncu(i),nx)];
% Di=[Di;-Ho{i}*Ci(1:ny,:)'*0;Hy{i}];
Ci=[Ci;zeros(ncu(i),nx)];
Di=[Di;Hr{i}];
```

Controller tuning

```
sysOld=ss(Ai,Bi,Ci,Di,TL);
model{i}=d2d(sysOld,TH(i));
dcg{i}=dcgain(model{i});
%keyboard
model{i}.d=[D(dec(i).y,dec(i).u);Hr{i}*dcg{i}(nu,nu)];
type='track';
```

MPC setup, 'help lincon' for more details

```
cost.S=blkdiag(1e2*eye(ny),1e-3*eye(ncu(i)));
cost.T=1e0*eye(nu);
cost.rho=1e6;
interval.N=10;
interval.Nu=5;
% Compute output constraints (only mass positions)
```

```
    toOut=[1 0 0 0;0 0 0 0;0 1 0 0;0 0 0 0;0 0 1 0;0 0 0 0;0 0 0 1;0 0 0 0]';
    Ycon.min=toOut*Xcon.min;
    Ycon.max=toOut*Xcon.max;
    limits.ymin=[Ycon.min(dec(i).y);-inf*ones(ncu(i),1)];
    limits.ymax=[Ycon.max(dec(i).y);Kr{i}];
    limits.dumin=-drOpt*ones(nu,1);
    limits.dumax=drOpt*ones(nu,1);
```

Controller object creation

```
    L{i}=lincon(model{i},type,cost,interval,limits,'cplex',1);
```

```
Error using ==> chkcost>chkwght at 117
Invalid dimensions of weight COST.Q (required dimension: 4-by-4)

Error in ==> chkcost at 58
                    w=chkwght(w,n,Name,wdef);

Error in ==> lincon.lincon at 188
    [cost,soft]=chkcost(cost,type,nx,nu,ny,costdef,A,B);

Error in ==> himpc_example at 279
    L{i}=lincon(model{i},type,cost,interval,limits,'cplex',1);


end

% Centralized Controller
cmodel=d2d(cmodel,cTH);
type='track';
cost.S=blkdiag(1e2*eye(4),1e-3*eye(17));
cost.T=1e0*eye(4);
cost.rho=1e6;
interval.N=10;
interval.Nu=5;
limits.ymin=[Ycon.min;-inf*ones(17,1)];
limits.ymax=[Ycon.max;cobj.Kr{1}];
limits.dumin=-cdrOpt*ones(4,1);
limits.dumax=cdrOpt*ones(4,1);
cL=lincon(cmodel,type,cost,interval,limits,'cplex',1);


Computing the MOARS for subsystem 1
Oinf=

Normalized, minimal representation polytope in R^8
         H: [149x8 double]
         K: [149x1 double]
    normal: 1
    minrep: 1
     xCheb: [8x1 double]
     RCheb: 0.1582
```

```
Done
Time needed for the 1-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=7.0711e-05
Time needed for the 2-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=7.1269e-05
Time needed for the 3-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=7.2933e-05
Time needed for the 4-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=7.5811e-05
Time needed for the 5-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=8.0004e-05
Time needed for the 6-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=8.4995e-05
Time needed for the 7-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=9.106e-05
Time needed for the 8-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00010262
Time needed for the 9-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
```

```
DeltaR=0.00010961
Time needed for the 10-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00012345
Time needed for the 11-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00014194
Time needed for the 12-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00016273
Time needed for the 13-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00018709
Time needed for the 14-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00022042
Time needed for the 15-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00026122
Time needed for the 16-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.0003067
Time needed for the 17-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00034993
Time needed for the 18-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
```

```
DeltaR=0.00037695
Time needed for the 19-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00038188
Time needed for the 20-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00036886
Time needed for the 21-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00034928
Time needed for the 22-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00032952
Time needed for the 23-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.0003118
Time needed for the 24-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00030166
Time needed for the 25-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00029265
Time needed for the 26-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00028916
Time needed for the 27-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
```

```
DeltaR=0.00029102
Time needed for the 28-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00029781
Time needed for the 29-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00030956
Time needed for the 30-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00032668
Time needed for the 31-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00035
Time needed for the 32-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00038084
Time needed for the 33-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00042127
Time needed for the 34-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.0004745
Time needed for the 35-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.00054556
Time needed for the 36-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
```

```
DeltaR=0.00064275
Time needed for the 37-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.0058935
Time needed for the 38-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.027401
Time needed for the 39-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.079081
Time needed for the 40-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
DeltaR=0.19829
Time needed for the 41-th problem of the 1 subsystem
+ Solver chosen : CPLEX-IBM
+ Processing objective h(x)
+ Processing F(x)
+ CPLEX-IBM
...DeltaR=0

----------
```

Simulation

Simulation time

```
Tsim = 3*60;
```

```
t=0:TL:Tsim-TL;
```

Number of iteration of the discrete time loop

```
TT=Tsim/TL;
```

Define references

```
r=[ .65*ones(1,TT/2-15) .2*ones(1,TT/2+15);
```

```
    .8*ones(1,TT/4+10) .7*ones(1,TT*2/4+10) .4*ones(1,TT*1/4-20);
    0.0*ones(1,TT/2-120) 0.7*ones(1,TT/2+120);
    0.0*ones(1,TT)];
```

Initial condition

```
x0=[0 0 0 0 0 0 0 0]';
```

Initial output

```
u0=zeros(4,1);
x=[x0 x0];
cx=x;
u=u0;
cu=u;
```

Simulation loop

```
for k=2:TT-1
```

For each controller

```
    for i=1:nDec
```

Determine which states are used by the current controller

```
        xx=x(dec(i).x,k);
        %yy=[xx;himpc_obj.Ho{i}*xx;himpc_obj.Hy{i}*u(dec(i).u,k-1)];
```

Determine the references that applies to the current sub-system

```
        rr=[r(dec(i).y,k);zeros(ncu(i),1)];
```

If it is not time to change the reference (input given by the current controller) then use the previous one

```
        u(dec(i).u,k) = u(dec(i).u,k-1);
```

Check if it is time for the higher level controller to execute

```
        if mod(k,TH(i)/TL)==2
```

The controller type is 'track' the MPC will return the input increment

```
            u(dec(i).u,k)=u(dec(i).u,k-1)+eval(L{i},xx,rr,u(dec(i).u,k-1));
        end
    end
```

Update the state

```
    x(:,k+1)=A*x(:,k)+B*u(:,k);
```

Assume the next input to be the same as current one. That is done for plot reasons which request x and u to have the same length

```
    u(:,k+1)=u(:,k);
```

Centralized

```
    if mod(k,cTH/TL)==2
        cu(:,k)=cu(:,k-1)+eval(cL,cx(:,k),[r(:,k);zeros(17,1)],cu(:,k-1));
    end
    cx(:,k+1)=A*cx(:,k)+B*cu(:,k);
    cu(:,k+1)=cu(:,k);

end

if showPlot
```

Plot the first (continuous blue) and second (dashed blue) mass positions against their references (continuous red and dash red), respectively

```
    figure(4);
    plot(t,x(1,:)','b',t,x(3,:)','--b',t,r(1,:)','r',t,r(2,:)','--r',t,cx(1,:)','k',t,cx(3,:)','-
    title('First and second mass positions')
```

Plot first and second inputs against their references

```
    figure(5);
    plot(t,u(1,:)','b',t,u(2,:)','--b',t,r(1,:)','r',t,r(2,:)','--r')
    hold on
    plot(t,cu(1,:)','k',t,cu(2,:)','--k')
    title('First and second inputs')
```

Plot showing all the mass trajectories against their references

```
    figure(6);
    plot(t,x(1:2:7,:)','b',t,cx(1:2:7,:)','k',t,r,'r')
    title('All mass positions')
```

Plot showing all inputs (applied references) against their references

```
    figure(7);
    plot(t,u','b',t,cu','k',t,r,'r')
    title('All inputs')

end
```

Computation of performance indices

Compute the MPC cost over the whole simulation horizon

```
index_d=0;
index_c=0;
for i=TT
    index_c=index_c+cx(:,i)'*C'*cost.S(1:4,1:4)*C*cx(:,i)+cu(:,i)'*cost.T*cu(:,i);
    index_d=index_d+x(:,i)'*C'*cost.S(1:4,1:4)*C*x(:,i)+u(:,i)'*cost.T*u(:,i);
end
disp(['Comulated const with centralized feedback is ' num2str(index_c) ...
    ' and with decentralized feedback is ' num2str(index_d)]);
disp(['Performance improvement is ' num2str(100*(index_c-index_d)/index_d) '%'])
```

Considerations

Figures 4 and 6 show that the hard constraint that bounds the mass upper position holds for both masses. Moreover the tracking is effective with limited overshoot. Figure 4 also show that the system is actually coupled since both masses 3 and 4, whose reference is stationary after time instant 20, have some fluctuation due to the interaction with neighbors.

Figures 5 and 7 show the references actually applied to the lower level controller. It is interesting to see how limits imposed have the effect of limiting the overshoot on the mass position. Moreover, it should be observed that the reference on the second mass (dashed blue line) does not reach the user defined reference (red continuous line) which is not in the admissible set for references (we recall that the admissible reference set is the output set tighter by the tuning know factor Δy for allowing the determination of a MOAS).

2.8 dncs

Approximating the Convergence Rate of a NCS - Example
This example shows how to find an upperbound on the convergence rate of a given NCS modeled as discrete-time linear parameter varying (DLPV) system. Specifically we mean solving for an lower bound on

$$\gamma$$

such that the state of the NCS

$$\bar{x}$$

satisfies

$$||\bar{x}_k|| \leq c||\bar{x}_0||(1-\gamma)^k$$

when the NCS has delays and transmission intervals bounded in a continuous range.

Contents

Define the Network Control System

The plant is a divided into two subsystems which are given as

$$\dot{x}_1 = A_1 x_1 + B_1 p u_1 + A_{c1} x_2 + B_{c1} u_2$$

$$\dot{x}_2 = A_2 x_2 + B_2 p u_2 + A_{c2} x_1 + B_{c2} u_1$$

with decentralized state feedback

$$u_1 = K_1 x_1$$

$$u_2 = K_2 x_2.$$

The plant subsystem matrices with corresponding feedback gains are defined as

```
clc

A1=[0.6  -4.2;
    0.1  -2.1];
B1=[0.7   1.9;
    0     1];

K1= [1.94 -1.40;
   -0.56 -0.86];

A2=[-3.2  0.2;
    5.3  -0.2];
B2=[0.8;
   -0.4];
K2 = [1.36 0.81];
```

and the coupling matrices are given as

```
Ac1 = [0.1   2.1;
       0.01   0];
Ac2 = [0     0;
       0  -0.03];

Bc1 = [-0.02;
       -0.01];
Bc2=[0 0;
     0 0];
```

We can express these two coupled systems as one system with the expression

$$\dot{x} = Ax + Bu$$

$$u = Kx$$

where

```
A=[A1 Ac1;
   Ac2 A2];
B=[B1 Bc1;
   Bc2 B2];
K= [K1  zeros(2);
    0 0    K2];
```

where `K` is a decentralized state feedback gain.

The network that the decentralized NCS is operating on has the following characteristics

```
h=[0.9, 1.1];     % [hmin,hmax] bounds on the sampling time
tau=[0, 1e-3];    % [taumin,taumax] bounds on the delay
delta = 0;        % integer bound on the number of subsequent dropouts
```

Create 'ncs' Object

Now we are ready to create a `ncs` class variables. This is done using the ncsEditor. To open the ncsEditor simply type 'ncsEditor' into the command prompt. Since the matrix variables are defined in the workspace, we can directly input the matrix names into the fields of the ncsEditor GUI. We do not consider communication constraints in this example so leave that box unchecked.

```
load exampleNcs
```

Generate Stability Data

Finally, to determine if out system is stable with a given `gamma`, we simply plug the `ncs` object into the following function:

```
gamma = 0;
stable = dncs.isNcsStable('JNF','explicit','pardep',gamma);

% where |explicit| is the dropout modeling approach we will use.
% Alternatively one could use the |lngtrans| dropout modeling approach.
% Also, 'CH' or 'GNB' are alternative overapproximation choices.

Undefined function or method 'dncs_JNF_CH' for input arguments of type 'ncs'.

Error in ==> ncs.ncs>ncs.isNcsStable at 154
               stable = dncs_JNF_CH(obj,ovraprx,arg1,arg2,arg3);

Error in ==> dncs_example at 96
stable = dncs.isNcsStable('JNF','explicit','pardep',gamma);
```

We chose `gamma=0` since it is the slowest decay rate and it is the most general way to verify system stability. We also chose the Lyapunov function to be parameter dependent (`pardep`) and the overapproximation to be the Jordan Normal Form (`JNF`).

Now since the above command results in a stable system, we can simply write a loop around this function to gradually increase `gamma` until stability cannot be gauranteed, which will provide an

upper bound on the convervence rate when the loop terminates. A simple example of this loop is the following:

```
if stable == 1
    while stable == 1 && gamma <= 1
        stable = dncs.isNcsStable('JNF','explicit','pardep',gamma);
        gamma = gamma +0.1;
        if stable == 1
            disp(['lower bound on gamma is ' num2str(gamma)])
            disp(' ')
        end
    end
end
```

Therefore our NCS is stable for `h=[0.9, 1.1]` and `tau=[0, 1e-3]`. Furthermore, decay of the state of our discrete-time system is upperbounded by the expression

$$||\bar{x}_k|| \leq c||\bar{x}_0||(1-0.3)^k$$

2.9 hncs

Contents

Comparing the TOD and RR Protocol Stability Regions - Example

This example shows how to compare the robustness of a given control setup which operates under either the Try-Once-Discard (TOD) network protocol or the Round Robin (RR) network protocol. The robustness is compared by means of generating a tradeoff plot between the maximally allowable transmission interval (MATI) and maximally allowbale delay (MAD).

```
clc
```

Define the Network Control System

The plant is a model of a batch reactor, which the dynamics are linearized and given in continuous-time as

$$\dot{x}_p = A_p x_p + B_p \hat{u}$$

$$y = C_p x_p$$

where

```
Ap=[1.38     -0.2077     6.715   -5.676;
    -0.5814  -4.29         0      0.675;
    1.067     4.273     -6.654    5.893;
    0.048     4.273      1.343   -2.104];

Bp=[0        0;
    5.679    0 ;
    1.136   -3.146;
    1.136    0 ];

Cp=[1 0 1 -1;
    0 1 0  0];
```

Next we define the controller, which is given as

$$\dot{x}_c = A_c x_c + B_c \hat{y}$$

$$u = C_c x_c + D_c \hat{y}$$

where

```
Ac=zeros(2);

Bc=[0 1;
    1 0];

Cc=[-2 0 ;
    0  8];

Dc=[0 -2;
    5  0];
```

Where `sy=[1 1]` indicates that both outputs of the plant are wired directly to the input of the controller and `Su=[0 0]` indicates that both outputs of the controller are shared on the network. Lastly, `l=2` means that the two outputs which are shared on the network are divided into two nodes.

Create 'ncs' Object

Now we are ready to create a `ncs` object. Since we want to compare two different protocols, we must create two seperate `ncs` objects. It is easiest to use the 'ncsEditor' to create these objects. To open the ncsEditor simply type 'ncsEditor' into the command prompt.

Once the 'ncsEditor' is opened, first input Ap,Bp and Cp into the plant parameters and then select 'C-LTI Dynamic Feedback' in the drop down menu Next input Ac,Bc,Cc,and Dc into the controller parameters. The transmission interval, delay and dropout boxes can be filled with 0. Check the box next to Comm Constraints an click 'Edit Nodes' and define two nodes where node1 has u01,u02 and y01 and node2 has u01,u02 and y02 then click 'save'. Chose the protocol as RR and click 'File > Export' and name the `ncs` as hncs_RR and change the protocol to TOD and click 'File > Export' and name this `ncs` as hncs_TOD.

```
load exampleNcs
```

Generate Stability Data

Finally, to generate the data for plotting the stability regions we simply plug each of the `ncs` variables into the following function

```
Sy = [1 1];      %[y1 y2 ... yN] where yi is 1 for networked, 0 for wired
Su = [0 0];      %[u1 u2 ... uM] where yi is 1 for networked, 0 for wired

[Hmati1,Tmad1] = hncs_RR.findNcsStablilityTradeoff([0 0],[1 1]);
[Hmati2,Tmad2] = hncs_TOD.findNcsStablilityTradeoff([0 0],[1 1]);
```

```
Undefined function or method 'ncs_hyb_outputfb' for input arguments of type 'ncs'.

Error in ==> ncs.ncs>ncs.findNcsStablilityTradeoff at 189
                [Hmati,Tmad] = ncs_hyb_outputfb(obj,Su,Sy);

Error in ==> hncs_example at 77
[Hmati1,Tmad1] = hncs_RR.findNcsStablilityTradeoff([0 0],[1 1]);
```

Plot Stability Region Comparision

Now we can plot the resulting data to see the comparision between the two protocols

```
plot(Hmati1,Tmad1,'r');
hold on;
plot(Hmati2,Tmad2,'b');
legend('TOD - Output FB','RR - Output FB')
title('Tradeoff Curves')
xlabel('MATI','interpreter', 'latex')
ylabel('MAD', 'interpreter', 'latex')
```

This plot indicates that the NCS is robustly stable in the region lying below the line drawn in the graph. From this comparision it is clear that the TOD protocol is more robust than the RR protocol.