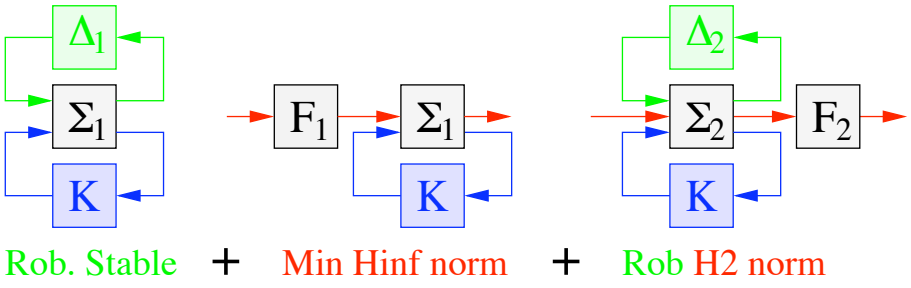
RoMulOC
a YALMIP-MATLAB based
Robust Multi Objective Control Toolbox

Dimitri Peaucelle
LAAS-CNRS
7, avenue du Colonel Roche 31077 Toulouse
peaucelle@laas.fr - tel : (+33)5-61-33-63-09

6th June 2005

Abstract

This is a user's guide for the RoMulOC toolbox. At this date the toolbox performs simple modeling operations for a large class of uncertain systems. LMI based functionalities are provided for robust performance analysis.



1 Introduction

This user's guide for RoMulOC is dedicated to a concise description of the toolbox utilities. The theoretical results on which are built the methods are not discussed in detail and the interested reader is referred to relevant published papers.

The toolbox contains some modeling utilities allowing simple system manipulations and a flexible description of uncertainties.

The manipulated models are in state-space, a dedicated Matlab object called `ssmodel` is defined on the basis of `ss` objects in the Control Toobox of Matlab. The originality of the `ssmodel` object is in explicitly distinguishing between several types of inputs and outputs : for example a disturbance input used to define noise rejection performance is different from a control input.

Two types of uncertain systems are then defined as `ussmodel` objects.

The first one corresponds to affine polytopic descriptions for which the parameters of the state-space representation live in the convex hull of a finite number of vertices. This polytopic modeling encompasses two sub-types : parallelotopic and interval descriptions, which can be manipulated with dedicated functionalities.

The second type of uncertain models is based on an Linear-Fractional Transform (LFT) representation where an uncertain operator is fed back to a Linear Time Invariant (LTI) `ssmodel`. A large variety of uncertain blocks can be defined as `uncertainty` objects, going from norm-bounded uncertainties to interval descriptions. The uncertain blocks can have any diagonal structure.

Based on these modeling functionalities, the toolbox allows to build Linear Matrix Inequality (LMI) optimization formulations for robust performance analysis and in the close future for robust multi objective design. Robust stability is one of the basic analysis tests but pole location, H_∞ cost, H_2 cost and impulse-to-peak performances are also tackled in the robust framework. Depending on the users requirements the LMI formulations can be more or less conservative and computationally demanding. The results are based on theoretical results that borrow from techniques such as quadratic stability, parameter dependent Lyapunov functions, quadratic separation and slack variables.

All analysis and design problems are formulated as LMIs with the YALMIP format, [Löf04a], [Löf04b]. Optimization can therefore be performed with any Semi-Definite Programming (SDP) solver interfaced by YALMIP. The users can therefore, if they desire, fully use the potentialities of the solvers and the YALMIP functionalities.

2 Getting started

2.1 Install ROMULOC, YALMIP and an SDP solver

RoMulOC is entirely based on m-code, and is thus easy to install. Remove any old version of RoMulOC, unzip the file `romuloc.zip` and add the two directories to your MATLAB path as illustrated in see Example 2.1.

Example 2.1 Add the RoMulOC paths in Matlab

```
>> addpath whereIputTHEdirectory/romuloc
>> addpath whereIputTHEdirectory/romuloc/library
```

If you have used RoMulOC before, type `clear classes` or restart MATLAB before using the new version. When this is done, most of the RoMulOC functions should work except for the essential ones that need to install

- YALMIP : `http://control.ee.ethz.ch/~joloef/manual/yalmip.html`
- and at least one SDP solver among those listed in `http://control.ee.ethz.ch/~joloef/manual/htmldata/solvers.htm`.
  The solvers should be installed as described in the solver manuals. Make sure to add required paths.

To test your installation, run the commands `yalmiptest.m` and `romuloc.m`.

2.2 License agreement

The current versions of RoMulOC and YALMIP are free of charge and openly distributed, but note that they are distributed in the hope that they will be useful, but **without any warranty**; without even the implied warranty of **merchantability** or **fitness for a particular purpose** (if your satellite crash or you fail your Phd due to a bug in RoMulOC or YALMIP, your loss!). RoMulOC and YALMIP may not be re-distributed as a part of a commercial product (if you make money from YALMIP, let Yohan in first!). RoMulOC and YALMIP must be referenced when used in a published work (give us some credit for saving your valuable time!).

```
@manual{romuloc,
author = "D. Peaucelle",
title = "{RoMulOC} a YALMIP-MATLAB based Robust Multi Objective Control Toolbox}",
url = {http://www.laas.fr/OLOCEP/romuloc},
year = "2005"}

@manual{yalmiphome,
author = "J. L{\"o}fberg",
title = "{YALMIP} :  A Toolbox for Modeling and Optimization in {MATLAB}",
url = {http://control.ee.ethz.ch/~joloef/yalmip.php},
year = "2004"}
```

or/and

```
@InProceedings{yalmipconf,
author = "J. L{\"o}fberg",
title = "{YALMIP} :  A Toolbox for Modeling and Optimization in {MATLAB}",
booktitle = "Proceedings of the {CACSD} Conference",
year = "2004",
address = "Taipei, Taiwan"}
```

2.3 A short example for a start

The following example is taken from [GdOB02] and [XLZZ04]. It is used here to illustrate robust H_2 performance analysis. Let us follow step by step the example without getting into details of the functions used.

```
>> sys=ssmodel('Geromel et al. 2002')
name: Geromel et al. 2002
empty SSMODEL
```

ROMULOC works along with a new definition of systems in Matlab environment. Here we have declared a new system, with an identifying tag for nice display purpose. The variable contains no data on the system. Now simply append the model

```
>> sys.A=[0.9 0.1; 0.01, 0.9];
>> sys.Bw=[ 1 0 0 ; 0 1 0];
>> sys.Cy=[1 0;1 1];
>> sys.Dyw=[0 0 1.414;0 0 0];
>> sys.T=1
name: Geromel et al. 2002
               n=2    mw=3
 n=2    dx  =   A*x +  Bw*w
py=2     y  =  Cy*x + Dyw*w
discrete time  ( dx : advance operator ) period T=1
```

At this stage we have declared a discrete-time linear system of order 2, with 3 disturbance inputs (represented by the notation w) and 2 measured outputs (represented by the notation y). In [GdOB02] and [XLZZ04], the uncertain system is defined as

$$\begin{pmatrix} x_{k+1} \\ y_k \end{pmatrix} = \left(\left[\begin{array}{cc|ccc} 0.9 & 0.1 & 1 & 0 & 0 \\ 0.01 & 0.9 & 0 & 1 & 0 \\ \hline 1 & 0 & 0 & 0 & 1.414 \\ 1 & 1 & 0 & 0 & 0 \end{array}\right] + \delta_1 \left[\begin{array}{cc|c} 0 & 0.06 & \mathbb{0} \\ 0 & 0 & \\ \hline \mathbb{0} & & \mathbb{0} \end{array}\right] + \delta_2 \left[\begin{array}{cc|c} 0 & 0 & \mathbb{0} \\ 0.05 & 0 & \\ \hline \mathbb{0} & & \mathbb{0} \end{array}\right] \right) \begin{pmatrix} x_k \\ w_k \end{pmatrix}$$

where the uncertainties are norm-bounded $|\delta_i| \leq 1$. This modeling is called parallelotopic. It amounts to adding variations around the 'central' system `sys` along axes that are systems with the same dimensions. Therefore let us define a variable called `axes` that defines an empty system with identical dimensions as `sys`

```
>> axes=ssmodel(0,sys)
               n=2    mw=3
 n=2    dx  =
py=2     y  =
discrete time  ( dx : advance operator ) period T=1
```

This variable is used as an array of systems with identical dimensions. Each element of the array contains one axis.

```
>> axes(1).A=[0 0.06;0 0];
>> axes(2).A=[0 0;0.05 0]
Array of 2 systems
               n=2    mw=3
 n=2    dx  =   A*x
py=2     y  =
discrete time  ( dx : advance operator ) period T=1
```

Based on these data, the uncertain parallelotopic system is declared as

```
>> usys=uparal(sys,axes)
Uncertain model : parallelotope 2 param
-------- WITH --------
               n=2    mw=3
 n=2    dx  =   A*x +  Bw*w
py=2     y  =  Cy*x + Dyw*w
discrete time  ( dx : advance operator ) period T=1
```

In [XLZZ04] an output filter is designed for signal reconstruction. The performance of the filter with respect to the disturbance input is measured using the H_2 norm of the transfer between w and z where z is the performance output signal of the filter defined as follows:

```
>> filter=ssmodel('filter Xie et al.2004');
>> filter.A=[0.0705 0.0263;1.2779 0.5492];
>> filter.Bu=[0.9114 0;-0.9972 0];
>> filter.Cz=[-1.2885 -0.2382];
>> filter.Dzu=[0 1];
>> filter.T=1
name: filter Xie et al.2004
               n=2    mu=2
 n=2    dx  =   A*x +  Bu*u
pz=1     z  =  Cz*x + Dzu*u
discrete time  ( dx : advance operator ) period T=1
```

The filtered uncertain system is obtained by plugging in the output of the system `sys` into the control input of `filter`. This corresponds to the usual multiplication of systems

```
>> filtered=filter*usys;
```

```
>> filtered.name = 'Filtered system'
Uncertain model : parallelotope 2 param
-------- WITH --------
name: Filtered system
                n=4    mw=3
 n=4    dx  =   A*x +  Bw*w
pz=1     z  =  Cz*x
discrete time  ( dx : advance operator ) period T=1
```

The robust analysis methods implemented in ROMULOC allow to deal with polytopic models which are a more general representation than parallelotopes. Polytopes describe uncertain sets as the convex combination of vertices. It is tedious to transform parallelotopes into polytopes but ROMULOC does it for you

```
>> filtered=u2poly(filtered)
Uncertain model : polytope 4 vertices
-------- WITH --------
name: Filtered system
                n=4    mw=3
 n=4    dx  =   A*x +  Bw*w
pz=1     z  =  Cz*x
discrete time  ( dx : advance operator ) period T=1
```

Now that the uncertain system is declared, the ROMULOC functions allow to test two methods for computing its robust H_2 cost. This is done in three steps. First declare the type of problem you want to solve and the methodology to be used.

```
>> quiz=ctrpb
 CHOOSE A CONTROL PROBLEM
(a) Analysis
(b) State-Feedback design
(c) Full-Order Dynamic Output-Feedback
choice > a
 CHOOSE A TYPE OF LYAPUNOV FUNCTION
(a) Unique over all uncertainties (quadratic stability)
(b) Quadratic w.r.t. del*(I-Ddd*del)^-1*Cd (for LFT SSMODEL objects)
(c) Polytopic (for polytopic USSMODEL objects)
choice > a

control problem: ANALYSIS
Lyapunov function: UNIQUE (quadratic stability)
No specified performance
```

Second, append the `quiz` variable to indicate that H_2 performance of `filtered` is required

```
>> quiz=h2(quiz,filtered)
control problem: ANALYSIS
Lyapunov function: UNIQUE (quadratic stability)
Specified performances / systems:
# minimize H2 / Filtered system
```

At this stage an LMI problem has been declared and stored in YALMIP format in the variable `quiz`. The analysis problem can be solved as a third and last step, using whatever `sdpsettings` of YALMIP.

```
>> solvesdp( quiz , sdpsettings('solver','sdpt3') )

 num. of constraints = 11
 dim. of sdp    var  = 20,   num. of sdp  blk  =  5
 dim. of linear var  =  4
   SDPT3: Infeasible path-following algorithms
************************************************************************
 it  pstep dstep p_infeas d_infeas  gap     mean(obj)    cputime
------------------------------------------------------------------------
 0  0.000 0.000 4.4e+01 2.0e+00  2.8e+02 -4.205273e+01  0.0 spchol  1  1
....
 14  0.998 1.000 2.3e-09 2.1e-16  6.9e-07 -2.212779e+01  1.8
 Stop: max(relative gap, infeasibilities) < 1.00e-07
...
------------------------------------------------------------------------
H2 norm = 4.70402 is guaranteed for all uncertainties
ans =
    4.7040
```

Note that the optimization has been done here using SDPT3 and the results are satisfying. This may not be the case for all examples and for all solvers. A significant issue of YALMIP (and therefore of ROMULOC) is to change solvers easily. For example, for the same problem perform the optimization using SeDuMi gives:

```
>> solvesdp( quiz , sdpsettings('solver','sedumi') )
  SeDuMi 1.05R5 by Jos F. Sturm, 1998, 2001-2003.
Alg = 2: xz-corrector, theta = 0.250, beta = 0.500
eqs m = 11, order n = 25, dim = 85, blocks = 6
nnz(A) = 238 + 0, nnz(ADA) = 111, nnz(L) = 61
 it :     b*y       gap    delta  rate   t/tP*  t/tD*   feas cg cg
  0 :            1.13E+01 0.000
...
  20 :  -2.21E+01 1.26E-11 0.000 0.0759 0.9900 0.9900   1.00  2  2
iter seconds digits       c*x               b*y
 20      0.8   Inf -2.2127786147e+01 -2.2127786146e+01
|Ax-b| =   6.6e-10, [Ay-c]_+ =   4.1E-11, |x|=  4.6e+02, |y|=  2.8e+01
Max-norms: ||b||=1, ||c|| = 2.577000e+00,
Cholesky |add|=0, |skip| = 0, ||L.L|| = 940.335.

Feasibility is not determined
Worst constraint residual is -4.14989e-11 < 0
4.70402 (=sqrt(double(CTRPB.vars{2}))) may be a guaranteed H2 norm
ans =
   NaN
```

The two results are close except that SeDuMi converged to a non strictly feasible point. The

distance to feasibility is evaluated as equal to `4.14989e-11`.

At this stage we have obtained that the H_2 norm of the uncertain system is less than 4.70402 whatever the uncertainties. Now let test the second method that involves parameter-dependent Lyapunov functions.

```
>> quiz=ctrpb;
 CHOOSE A CONTROL PROBLEM
(a) Analysis
(b) State-Feedback design
(c) Full-Order Dynamic Output-Feedback
choice > a
 CHOOSE A TYPE OF LYAPUNOV FUNCTION
(a) Unique over all uncertainties (quadratic stability)
(b) Quadratic w.r.t. del*(I-Ddd*del)^-1*Cd (for LFT SSMODEL objects)
(c) Polytopic (for polytopic USSMODEL objects)
choice > c

>> quiz=h2(quiz,filtered)

control problem: ANALYSIS
Lyapunov function: POLYTOPIC (only for polytopic USSMODEL)
Specified performances / systems:
# minimize H2 / Filtered system

>> solvesdp(quiz,sdpsettings('solver','sdpt3'))

 num. of constraints = 73
 dim. of sdp    var  = 36,   num. of sdp  blk  =  5
 dim. of linear var  =  4
*****************************************************************
   SDPT3: Infeasible path-following algorithms
*****************************************************************
it  pstep dstep p_infeas d_infeas  gap     mean(obj)    cputime
-----------------------------------------------------------------
 0  0.000 0.000 1.4e+02 2.5e+00  1.1e+03 -8.410546e+01  0.1 spchol  2  2
...
 13  0.355 1.000 9.3e-07 1.5e-13  5.4e-06 -1.539477e+01  2.4
 Stop: relative gap < infeasibility.
...
H2 norm = 3.92362 is guaranteed for all uncertainties
```

For analysis purpose, parameter-dependent Lyapunov function based methods are all proved to be less conservative than quadratic stability. The conservatism is reduced at the expense of larger LMI problems to solve. Here the number of constraints grow from 11 to 73 and the number of decision variables from 20 to 36. Meanwhile the computation time grows (on the SunBlade 150 computer I used) from 1.8 to 2.4. In terms of conservatism, the computed upper bound on the worst-case H_2 norm is reduced from 4.70402 to 3.92362.

To finish with this experiment, compute the H_2 norm of some specific systems within the uncertain set. First test the central 'nominal' system (calculated as the center of the polytope):

```
>> norm(filtered,2)
Nominal
ans =
    2.9636
```

Second test any of the four vertices of the polytope, for example the fourth:

```
>> norm(filtered(4),2)
System is stable
ans =
    3.9236
```

The H_2 norm on this vertex is equal to the upper bound found by the second method. Therefore we may conclude that it is the worst-case H_2 norm for the filtered system. This is also attested using a gridding in [XLZZ04].

3 Modeling of LTI state-space systems : `ssmodel` object

3.1 Three types of inputs and outputs

To cope simultaneously with controller design requirements, input/output performances and uncertainties, the `ssmodel` object explicitly distinguishes between three pairs of outputs/inputs:

- MEASUREMENTS/CONTROL INPUTS. Denoted y/u, these are the actual vector signals accessible for control purpose.
- CONTROL OUTPUTS/PERTURBATIONS. Denoted z/w, these vector signals are used to define input/output performance specifications such as H_∞ norm, H_2 norm or impulse-to-peak.
- EXOGENOUS. Denoted z_Δ/w_Δ, these are fictitious vector signals used for LFT modeling of uncertainties.

Both continuous-time and discrete-time LTI systems can be defined. For the discrete-time case $\delta[x(t)] = x(t+T)$ where T is the sampling time. For the continuous-time case $\delta[x(t)] = \dot{x}(t)$. The matrices of the state-space model are as follows:

$$
\begin{array}{rrrrrr}
 & & x \in \mathbb{R}^n & w_\Delta \in \mathbb{R}^{m_\Delta} & w \in \mathbb{R}^{m_w} & u \in \mathbb{R}^{m_u} \\
 & \delta[x(t)] = & Ax(t)+ & B_\Delta w_\Delta(t)+ & B_w w(t)+ & B_u u(t) \\
z_\Delta \in \mathbb{R}^{p_\Delta} & z_\Delta(t) = & C_\Delta x(t)+ & D_{\Delta\Delta} w_\Delta(t)+ & D_{\Delta w} w(t)+ & D_{\Delta u} u(t) \\
z \in \mathbb{R}^{p_z} & z(t) = & C_z x(t)+ & D_{z\Delta} w_\Delta(t)+ & D_{zw} w(t)+ & D_{zu} u(t) \\
y \in \mathbb{R}^{p_y} & y(t) = & C_y x(t)+ & D_{y\Delta} w_\Delta(t)+ & D_{yv} v(t)+ & D_{yu} u(t)
\end{array}
$$

Any of the seven dimensions $n, m_\Delta, m_w, m_u, p_\Delta, p_z, p_y$ may be equal to zero.

3.2 Extract data from an `ssmodel`

3.2.1 Display an `ssmodel`

The output display of `ssmodel` objects shows explicitly the relations (if any) between the various input/outputs, without giving numerical values of the various matrices. The display insists on the defined dimensions and non zero matrices. Examples 3.1, 3.2 and 3.3 illustrate three possible displays.

Example 3.1 First example of `ssmodel` display

```
>> sys1
name: example1
              n=4    md=2     mw=2    mu=1
 n=4    dx  =  A*x +  Bd*wd +  Bw*w +  Bu*u
pd=3    zd  = Cd*x + Ddd*wd + Ddw*w + Ddu*u
pz=1     z  = Cz*x + Dzd*wd + Dzw*w
py=1     y  = Cy*x
continuous time system ( dx : derivative operator )
```

3.2.2 Extract data of an `ssmodel`

All the data can be extracted using structured array referencing, see Example 3.4.

Example 3.2 Second example of `ssmodel` display

```
>> sys2
Array of 4 systems
name: example2
static gain   md=1     mw=2    mu=2
pd=1    zd  =          Ddw*w + Ddu*u
pz=1     z  = Dzd*wd
py=3     y  = Dyd*wd + Dyw*w + Dyu*u
```

Example 3.3 Third example of `ssmodel` display

```
>> sys3
name: example3
              n=7    mw=1    mu=3
 n=7    dx  =  A*x +  Bw*w +  Bu*u
pz=1     z  = Cz*x          + Dzu*u
py=3     y  = Cy*x          + Dyu*u
discrete time LTI system  ( dx : advance operator ) period T=1
```

Example 3.4 Extract data from `ssmodel` objects

```
>> sys3.Dzu
ans =
     0     0     1
>> sys3.name
ans =
example3
>> sys3.Ts
ans =
     1
>> sys3.n
ans =
     7
>> sys2.nb
ans =
     4
```

3.2.3 Extract size of `ssmodel`

All system dimensions can be either display with the previous subsreference controls or else with the `size` function, see Example 3.5.

Example 3.5 Extract size of `ssmodel` objects

```
>> size(sys1)
 8 states
 3/2 exogenous outputs/inputs
 1 performance outputs / 2 disturbance inputs
 1 measure outputs / 1 control inputs

>> [ qu, py ] = size(sys1)
ans =
     1     1

>> [ n, pz, pg, py, mw, mv, mu ] = size( sys1, 'all' )
ans =
     4     3     1     1     2     2     1
```

3.3 Define an `ssmodel`

Defining an `ssmodel` implies declaring quite a number of matrices, which is quite tricky to do at once because of possible confusions and errors in the dimensions. Therefore several alternative methods are available for the user.

3.3.1 Indirect method

The procedure is to define an empty object and then set each field of the structured array. The assignment is done as for a structured array, see Example 3.6.

3.3.2 Partitioning method

This method is based on a description where all the matrices are concatenated into a unique matrix M representing the linear transformation:

$$\begin{pmatrix} \delta[x] \\ z_\Delta \\ z \\ y \end{pmatrix} = M \begin{pmatrix} x \\ w_\Delta \\ w \\ u \end{pmatrix}$$

The blocks corresponding to each transfer matrix may not be ordered as in this equation. They are specified using sets of row indices $\mathcal{I}_{xr}$, $\mathcal{I}_{z\Delta}$, $\mathcal{I}_z$, $\mathcal{I}_y$ and column indices $\mathcal{I}_{xc}$, $\mathcal{I}_{w\Delta}$, $\mathcal{I}_w$, $\mathcal{I}_u$. These sets are defined in Matlab as vectors `ixr`, `izd`, `iz`, `iy`, `ixc`, `iwd`, `iw` and `iu` respectively. The various matrices of the LTI system are obtained from the $M = (m_{ij})$ matrix as follows:

$$A = (m_{ij})_{i\in\mathcal{I}_{xr}, j\in\mathcal{I}_{xc}} \quad , \quad B_\Delta = (m_{ij})_{i\in\mathcal{I}_{xr}, j\in\mathcal{I}_{w\Delta}} \quad , \quad D_{yw} = (m_{ij})_{i\in\mathcal{I}_y, j\in\mathcal{I}_w}$$

This partitioning method is described in Example 3.7.

Example 3.6 Indirect method to declare an `ssmodel`

```
>> sys4 = ssmodel
empty SSMODEL
>> sys4.Bw = rand(4,2)
              mw=2
 n=2    dx  =  Bw*w
continuous time ( dx : derivative operator )
>> sys4.name='helicopter'
name: helicopter
              mw=2
 n=2    dx  =  Bw*w
continuous time system ( dx : derivative operator )
>> sys4.A = rand(2,2)
??? Error using ==> ssmodel/set
dimensions do not fit with specified number of states
>> sys4.A(1:2,1:2) = rand(2,2)
name: helicopter
              n=4    mw=2
 n=4    dx  =  A*x +  Bw*w
continuous time system ( dx : derivative operator )
```

Example 3.7 Example of partitioning method to declare an `ssmodel`

```
>> ixr = 1:3; izd = 4:5; iz = 5; iy = 5:6;
>> ixc = 1:3; iwd = 4; iw = 4:5; iu = 5:7;
>> ssmodel(M, ixr, izd, iz, iy,  ixc, iwd , iw, iu,  0.1)
               n=3     md=1    mw=2    mu=3
 n=3    dx  =   A*x +  Bd*wd +  Bw*w +  Bu*u
pd=2    zd  =  Cd*x          + Ddw*w + Ddu*u
pz=1     z  =  Cz*x          + Dzw*w + Dzu*u
py=2     y  =  Cy*x          + Dyw*w + Dyu*u
discrete time  ( dx : advance operator ) period T=0.1
```

3.3.3 Compatibility with the Control System Toolbox:

Our toolbox is compatible with the CST and a `ssmodel` can de defined from a given `lti` object of the CST. The idea is similar to the partitioning method, see Example 3.8.

Example 3.8 Compatibility with `lti` objects of the Control Toolbox

```
>> sysControl = ss(A,B,C,D);
>> izd = 1; iz = 2:3; iy = 4:6;
>> iwd = 1:2; iw = 3:4; iu = 5:6;
>> heli = ssmodel(sysControl,  izd,iz,iy, iwd,iw,iu, 'helicopter')
name: helicopter
              n=3     md=1    mw=2    mu=3
 n=3    dx  =  A*x +  Bd*wd +  Bw*w +  Bu*u
pd=2    zd  = Cd*x + Ddd*wd + Ddw*w + Ddu*u
pz=2     z  = Cz*x           + Dzw*w + Dzu*u
py=2     y  = Cy*x + Dyd*wd + Dyw*w + Dyu*u
continuous time system ( dx : derivative operator )
```

Two abbreviated notations are also available to specify directly that all the inputs and outputs of the `lti` system describe either the y/u couple of control signals or the z/w couple of performance signals, see Example 3.9.

Example 3.9 Compatibility with `lti` objects of the Control Toolbox

```
>> pidControl = tf( ... );
>> pid = ssmodel(pidControl , 'control')
              n=3    mu=1
 n=3    dx  =  A*x +  Bu*u
py=1     y  = Cy*x + Dyu*u
continuous time system ( dx : derivative operator )

>> weightControl = zpk( ... );
>> weight = ssmodel(weightControl , 'filter')
              n=2    mw=1
 n=2    dx  =  A*x +  Bw*w
pz=1     z  = Cz*x
continuous time system ( dx : derivative operator )
```

3.3.4 Compatibility with the LFR Toolbox

We plan in the close future to implement the compatibility with the LFR toolbox of J.-F. Magni which provides functions to build minimal, well conditioned system LFT representations.

3.4 Arrays of `ssmodel` objects

It is possible to define arrays of models with the same number of state/input/output signals and the same sampling time. The usefulness is illustrated in the section devoted to `ussmodel` objects that describe uncertain LTI systems.

Only one dimensional `ssmodel` arrays are implemented at this time. The subs-referencing `sys2(4)` (to access the fourth element of the array of `ssmodel` stored in `sys2`) is valid while

`sys(1,2)` will not work.

Arrays of `ssmodel` can be build using subs-referencing or else can be obtained:
- with the partitioning method as in table 3.7, in that case `M` should be a three dimensional matrix;
- or using the compatibility with CST as in table 3.8, in that case `sysControl` should be a one dimensional array of `lti` objects.

In case `M` or `sysControl` have more dimension, then the resulting `ssmodel` has the same number of elements but reshaped to a one dimensional array.

To access and modify the data of an array of `ssmodel` objects see Example 3.10.

Example 3.10 Manipulating arrays of `ssmodel` objects

```
>> size(sys2)
Array of 4 systems
Each model has:
 0 states
 1/1 exogenous outputs/inputs
 1 performance outputs / 2 disturbance inputs
 3 measure outputs / 2 control inputs

>> sys2(1)
name: example2
static gain   md=1     mw=2    mu=2
pd=1    zd  =          Ddw*w + Ddu*u
pz=1     z  = Dzd*wd
py=3     y  = Dyd*wd + Dyw*w + Dyu*u

>> sys2(1).Ddd = 1
Array of 4 systems
name: example2
static gain   md=1     mw=2    mu=2
pd=1    zd  = Ddd*wd + Ddw*w + Ddu*u
pz=1     z  = Dzd*wd
py=3     y  = Dyd*wd + Dyw*w + Dyu*u
```

All the operators defined for `ssmodel` objects that are described in the following sub sections for the case of a single `ssmodel`, can also handle arrays of such objects. See the inline `help` for details.

3.5 Specific functions for `ssmodel` objects

3.5.1 Connect systems

The following elementary operations between two systems are available only for systems of the same class of `ssmodel` objects. To perform these operations with other objects such as `lti` objects, it is required to convert first to the `ssmodel` class using any method illustrated in Examples 3.8 and 3.9.

Multiply two systems: This operation corresponds to building a model for two models in series where the measured output is applied to the control input. The obtained exogenous signals

are concatenations of those of the original systems. Multiplication is not valid if both systems have controlled outputs z or disturbance inputs w. The global scheme is given on figure 1.

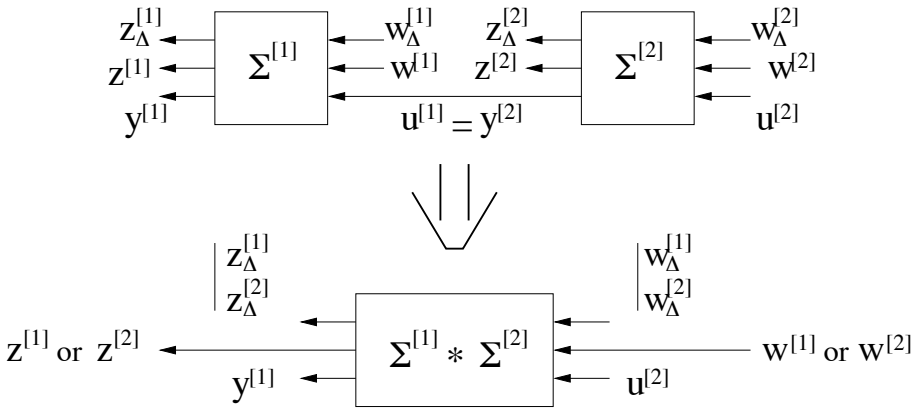


Figure 1: Multiplication

It is assumed that the measured output of the first system has the same dimension as the input of the second system. See Example 3.11 for notations.

Example 3.11 Multiplication of two `ssmodel` objects

```
>> sysMULT = mtimes( sys , pid );
>> sysMULT = sys * pid ;
```

Performance/Disturbance shaping: In case of loop-shaping as well as for many other industrial specifications, it may be needed to weight the input disturbance or output performance signals. This amounts to introducing some LTI model in series with the input w or the output z. This other model may as well have z_Δ/w_Δ exogenous signals but it does not have control/measure signals since it is only defined for performance purpose. The global scheme is that of figure 2. The global scheme is defined for both cases where the shaping operates on the input w or on the output z.

It is assumed that the performance output of the first system has the same dimension as the disturbance of the second system. Moreover, one of the two systems must have zero measured output / control input dimensions. See Example 3.12 for notations.

Example 3.12 Shaping the disturbance input

```
>> shape(sys, weight, 'helicopter with shaped input')
name: helicopter with shaped input
              n=5    md=1     mw=2    mu=3
 n=5    dx  =  A*x +  Bd*wd +  Bw*w +  Bu*u
pd=2    zd  = Cd*x + Ddd*wd + Ddw*w + Ddu*u
pz=1     z  = Cz*x          + Dzw*w + Dzu*u
py=2     y  = Cy*x + Dyd*wd + Dyw*w + Dyu*u
continuous time system ( dx : derivative operator )
```

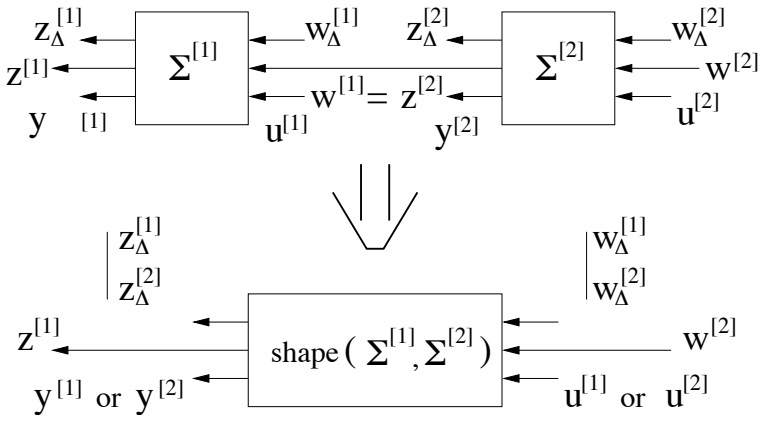


Figure 2: Disturbance/Performance shaping

Uncertainty feedback: Two types of feedback loops are considered. The first one is an "upper" feedback that connects the output/input signals z_Δ/w_Δ with some constant gain matrix. This operation amounts to chosing a value for the uncertain parameters of the system. It can be used when the user wants to test the system properties for given values of the uncertainty. The global scheme is that of figure 3. Example 3.13, where `heli` is the variable defined on page 14, illustrates the use of `certain`.

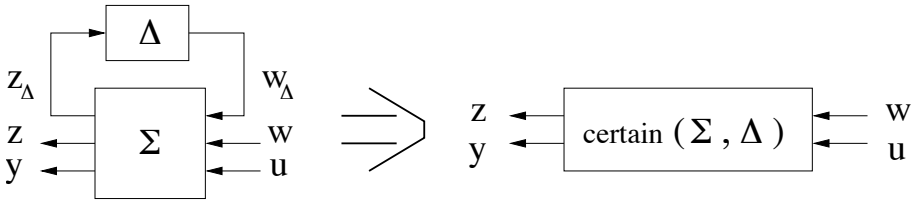


Figure 3: Uncertainty feedback

Example 3.13 Fixing the uncertainty value

```
>> Delta = [ 0.1 , 3.45 ];
>> certain( heli , Delta )
              n=3    mw=2    mu=3
 n=3    dx  =  A*x +  Bw*w + Bu *u
pz=2     z  = Cz*x + Dzw*w + Dzu*u
py=2     y  = Cy*x + Dyw*w + Dyu*u
continuous time system ( dx : derivative operator )
```

Control feedback: The second feedback loop is a "lower" feedback that connects the output/input signals y/u with some control law. The global scheme is that of figure 4. Note that in order to analyse the closed loop fragility, the controller may be uncertain. Therefore, exogenous signals may be defined for the controller and the closed loop model concatenates these with the exogenous signals of the process model. The same rule applies on the performance signals z/w

of the two systems as for the multiplication operator. Example reft-fb-ssmodel-2, where `heli` is the variable defined on page 14, illustrates the use of `feedback`.

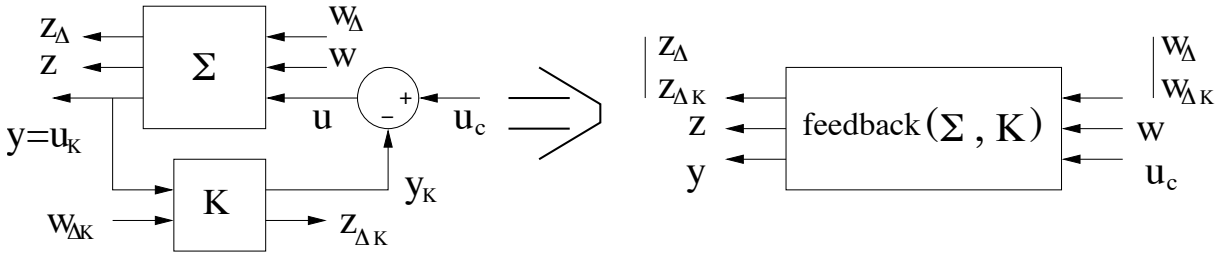


Figure 4: Control feedback

Example 3.14 Controlled helicopter

```
>> K = [ 0.1 , 3.45 ];
>> feedback( heli, K, 'closed loop helicopter')
name: closed loop helicopter
              n=3    md=1    mw=2    mu=3
 n=3    dx  =  A*x +  Bd*wd +  Bw*w +  Bu*u
pd=2    zd  = Cd*x + Ddd*wd + Ddw*w + Ddu*u
pz=2     z  = Cz*x + Dzd*wd + Dzw*w + Dzu*u
py=2     y  = Cy*x + Dyd*wd + Dyw*w + Dyu*u
continuous time system ( dx : derivative operator )
```

Note that a function `sfeedback` is available for state-feedback ($u_K = x$).

3.5.2 Standard analysis functions

Standard analysis tools can be developped for the `ssmodel` objects. The point of the toolbox being LMI-based techniques for robust and multi objective control, the developed functions are trivial interfaces with existing Matlab (Control Toolbox) functions.

Here is a list of such functions:

- Compute poles: `poles`
- Compute the norm of the performance/disturbance transfer: `norm`
- Plot impulse response for the performance/disturbance transfer and get the maximal norm of the output z: `impulse`

4 Modeling of uncertain operators: `uncertainty` object

Real valued parametric uncertainties are considered. Extensions to complex valued uncertainties, non-linear uncertain blocks and others, may be considered in the future. At this stage, the uncertainties are assumed to be real and constant.

4.1 Two types of uncertainties and sub-types

Block-diagonal structured uncertainties are assumed. Each block may appear several times on the diagonal at arbitrary locations.An example of uncertainties with three independent blocks is:

$$\Delta = \left[\begin{array}{cccccc} \Delta_1 & \mathbb{0} & \mathbb{0} & \mathbb{0} & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \Delta_2 & \mathbb{0} & \mathbb{0} & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \Delta_3 & \mathbb{0} & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \mathbb{0} & \Delta_2 & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \mathbb{0} & \mathbb{0} & \Delta_1 & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \mathbb{0} & \mathbb{0} & \mathbb{0} & \Delta_2 \end{array} \right] \quad .$$

Two main types of blocks are considered: $\{X, Y, Z\}$-dissipative uncertainties and polytopic uncertainties.

4.1.1 $\{X, Y, Z\}$-dissipative uncertainties

$\{X, Y, Z\}$-dissipative uncertainties are constrained by a quadratic matrix inequality such as:

$$\left[\begin{array}{cc} \mathbb{1} & \Delta \end{array} \right] \left[\begin{array}{cc} X & Y \\ Y' & Z \end{array} \right] \left[\begin{array}{c} \mathbb{1} \\ \Delta \end{array} \right] \leq \mathbb{0} \quad , \quad Z \geq \mathbb{0} \quad .$$

This modeling of constraints on uncertain blocks is quite general. It allows the users to avoid tedious manipulations on models and uncertainties for the cases when the uncertainties are not in norm-bounded format. Indeed, if the uncertainty is defined by the following norm-bounded constraint (centered at Δ_o, scalled by T_1 and T_2):

$$\|T_1(\Delta - \Delta_o)T_2\| \leq \rho$$

without modifying the LFT between the uncertainty and the LTI system it may be defined as a $\{X, Y, Z\}$-dissipative uncertainty where:

$$X = \Delta_o' T_1' T_1 \Delta_o - \rho^2 T_2^{-T} T_2^{-1} \quad , \quad Y = -\Delta_o' T_1' T_1 \quad , \quad Z = T_1' T_1 \quad .$$

The advantage of such modeling is to keep the knowledge of the actual uncertain parameters.

Well-known sub-cases of $\{X, Y, Z\}$-dissipative uncertainties are:

- **norm-bounded** uncertainties ($\|\Delta\| \leq \rho$) which are $\{-\rho^2\mathbb{1}, \mathbb{0}, \mathbb{1}\}$-dissipative uncertainties,
- **positive-real** uncertainties ($\Delta' + \Delta \geq \mathbb{0}$) which are $\{\mathbb{0}, -\mathbb{1}, \mathbb{0}\}$-dissipative uncertainties.

These two important sub-types are included explicitely in the toolbox.

4.1.2 Polytopic uncertainties

These uncertainties lie in the convex hull of a finite number of vertices:

$$co\left\{\Delta^{[1]},\Delta^{[2]},\ldots,\Delta^{[N]}\right\}=\left\{\begin{array}{ccccc}\Delta=\sum_{i=1}^{N}\zeta_i\Delta^{[i]} & : & \zeta_i\geq 0 & , & \sum_{i=1}^{N}\zeta_i=1\end{array}\right\} .$$

Such modeling describes structured uncertainties with N independent parameters: the barycentric coordinates ζ.

Two sub-types of polytopic uncertainties are:

- **Parallelotopes:** These polytopes are built starting from an initial nominal matrix and describing unitary variations around this point along several axes. The formulas are:

$$\left\{\begin{array}{ccc}\Delta=\Delta^{|0|}+\sum_{i=1}^{N_p}\delta_i\Delta^{|i|} & : & |\delta_i|\leq 1\end{array}\right\} .$$

  The parallelotopes defined by N_p axes are polytopes with $N=2^{N_p}$ vertices built out from all extremal points such that $|\delta_i|=1$.

- **Intervals:** A sub-case of parallelotopes where the $\Delta^{|i|}$ matrices are of rank one and belong to the standard Euclidian basis of matrices. The simplest notation is:

$$\left\{\begin{array}{ccccc}\underline{\Delta} & \preceq & \Delta & \preceq & \bar{\Delta}\end{array}\right\}$$

  where the $\preceq$ symbol stands for entry-wise inequalities ($\underline{\Delta}_{i,j}\leq\Delta_{i,j}\leq\bar{\Delta}_{i,j}$). Intervals are polytopes with 2^{N_I} vertices where N_I is the number of distinct elements in $\underline{\Delta}$ and $\bar{\Delta}$ matrices. The vertices are built out from all the combinations of elements from $\underline{\Delta}$ and $\bar{\Delta}$.

Again, these two important sub-types are included explicitely in the toolbox.

4.2 Extract data from an `uncertainty`

4.2.1 Display an `uncertainty`

The display gives the block structure of the uncertainty and the type of constraints of each block. The chosen display does not insist numerical values but gives a short description of each block, see Example 4.1.

Example 4.1 Example of `uncertainty` display

```
>> uD
diagonal structured uncertainty
size: 7x9 | nb blocks: 5 | independent blocks: 4
 wd = diag( #2 #4 #5 #6 #4 ) * zd
index   size    constraint              name
#2      2x1     norm-bounded by 1.235   gain of filters
#4      1x2     polytope 3 vertices     coupled parameters m1 and m2
#5      1x2     interval 2 param        independent parameters n1 and n2
#6      2x2     parallelotope 3 param   independent parameters p1, p2, p3
```

4.2.2 Extract uncertainty block from a structured `uncertainty`

A specific block of a structured uncertainty may be extracted using curly braces, see Example 4.2.

Example 4.2 Extract a block from an `uncertainty`

```
>> a=uD{4}
 wd = #4 * zd
index   size    constraint              name
#4      1x2     polytope 3 vertices     coupled parameters m1 and m2
```

4.2.3 Extract data of an `uncertainty` block

The data on constraints on each block can be extracted with the `udata` function, see Example 4.3. The first input argument is the `uncertainty` and the second one indicates the block for which the data are required. For uncertainties with single block, the second input argument is optionnal.

Example 4.3 Extract data on constraints from `uncertainty` objects

```
>> [valmin,valmax,name] = udata(uD,5)
valmin =
    -2    -3
valmax =
     4     3
name =
independent parameters n1 and n2
```

4.2.4 Extract size of an `uncertainty`

See Example 4.4.

Example 4.4 Extract size of `uncertainty` object

```
>> size(uD)
ans =
     7     9
>> a = uD{2}
 wd = #2 * zd
index   size    constraint              name
#2      2x1     norm-bounded by 1.235   gain of filters

>> size(a,2)
ans =
     1
```

4.3 Define an `uncertainty`

There is a single Matlab object to store all possible uncertainties. But to make it simpler for the user, specific functions are developed for each type of constraints on uncertainty blocks. To

produce a structured uncertainty, it is first necessary to define each individual block.

4.3.1 {X, Y, Z}-dissipative (udiss), Norm-bounded (unb), Positive-real (upos)

A function `udiss` allows to declare any $\{X, Y, Z\}$-dissipative uncertainty. The input arguments are the X, Y, Z matrices as well as an optional string `name` (used for nice display). As exposed earlier, the most common $\{X, Y, Z\}$-dissipative uncertainties are the norm-bounded uncertainties and the positive-real uncertainties. For these two sub-cases, dedicated functions (`unb` and `upos`) are exposed. Example 4.5 illustrates the use of these functions.

Example 4.5 Define dissipative, norm-bounded and positive-real `uncertainty`

```
>> uiner = udiss( -5*eye(2), [1 0 0;0 1 0], eye(3), 'uncertainties on inertia')
 wd = #1 * zd
index   size    constraint              name
#1      3x2     {X,Y,Z}-dissipative     uncertainties on inertia

>> ufilt = unb( 2, 1, 1.234567, 'gain of filters');

>> ufric = upos( 3, 'passive friction')
 wd = #3 * zd
index   size    constraint              name
#3      3x3     positive-real           passive friction
```

4.3.2 Polytopic (upoly), Interval (uinter), Parallelotopic (uparal)

A function `upoly` allows to declare any polytopic uncertainty. The input argument is a three-dimensional array, the third dimension spanning the vertices, and an optional string `name` (used for nice display). As exposed earlier, two sub-cases are frequently encountered in the litterature. Dedicated functions, `uinter` for interval uncertainties and `uparal` for parallelotopic uncertainties, are implemented. Example 4.6 illustrates the use of these functions.

Example 4.6 Define polytopic, interval, parallelotopic `uncertainty`

```
>> v(:,:,1)=[-1  1];
>> v(:,:,2)=[-1 -1];
>> v(:,:,3)=[ 1  0];
>> uM = upoly( v, 'coupled parameters m1 and m2');

>> uN = uinter( [-2 -3], [4 3], 'independent parameters n1 and n2');

>> cntr=eye(2);
>> ax(:,:,1)=[0 1;1 0];
>> ax(:,:,2)=[1 0;0 0];
>> ax(:,:,3)=[0 0;0 1];
>> uP = uparal( cntr, ax, 'independent parameters p1, p2, p3');
```

4.3.3 Diagonal structured: `diag`

Block diagonal structured uncertainties are declared using the overloaded function `diag`, see Example 4.7. An important feature here is that blocks can be repeated times on the diagonal

Example 4.7 Define diagonal structured `uncertainty`

```
>> uD = diag( ufilt, uM, uN, uP, uM);
```

4.4 Modify `uncertainty` constraints

4.4.1 Convert parallelotopes and intervals to polytopes

Most of the robust analysis methods implemented at this time in the toolbox are dedicated to polytopic uncertainties rather than parallelotopic or interval. Therefore a simple function is developed to convert uncertainties to polytopic modeling when it is possible, see Example 4.8.

Example 4.8 Convert to polytopic modeling

```
>> uP
 wd = #5 * zd
index   size    constraint              name
#5      2x2     parallelotope 3 param   independent parameters p1, p2, p3

>> u2poly(uP)
 wd = #7 * zd
index   size    constraint              name
#7      2x2     polytope 8 vertices     independent parameters p1, p2, p3
```

5 Modeling of uncertain systems: `ussmodel` object

5.1 LFT type uncertain systems

A `ussmodel` object describes an uncertain LTI model. It is composed of an `ssmodel` (see section 3) and an `uncertainty` (section 4). This object inherits of all properties and functions associated to these two objects. In particular functionalities such as getting data, display, multiplication, etc. that apply to `ssmodel` and `uncertainty` object apply exactly the same for `ussmodel` objects.

Example 5.1 shows the declaration of an LFT type uncertain model such as described on figure 5.

Example 5.1 Define `ussmodel` object

```
>> usys = ussmodel( sys , uD );
```

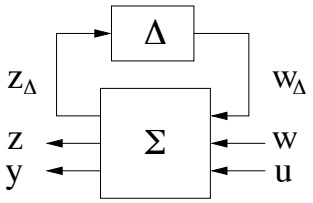


Figure 5: Uncertain LTI system

5.2 Polytopic type uncertain systems

Not all the uncertain LTI systems are described by LFTs. A common modeling is to define the system matrices as elements of a polytope. In that case there is no z_Δ/w_Δ signals. A simple manner to describe this type of modeling is to declare the system vertices of the polytope. All vertices are LTI systems and must have same output/input and state dimensions. If each vertex ($\Sigma^{[i]}$) is defined as:

$$\begin{array}{rrlll} & & x \in \mathbb{R}^n & v \in \mathbb{R}^{m_w} & u \in \mathbb{R}^{m_u} \\ & \delta[x(t)] = & A^{[i]}x(t)+ & B_w^{[i]}w(t)+ & B_u^{[i]}u(t) \\ z \in \mathbb{R}^{p_z} & z(t) = & C_z^{[i]}x(t)+ & D_{zw}^{[i]}w(t)+ & D_{zu}^{[i]}u(t) \\ y \in \mathbb{R}^{p_y} & y(t) = & C_y^{[i]}x(t)+ & D_{yw}^{[i]}w(t)+ & D_{yu}^{[i]}u(t) \end{array}$$

then the uncertain polytopic system with vertices $\Sigma^{[1]}$, $\Sigma^{[2]}$, $\ldots$ $\Sigma^{[N]}$ is described by the set :

$$\left\{ \quad \Sigma(\zeta) \quad : \quad \zeta_i \geq 0 \quad , \quad \sum_{i=1}^{N} \zeta_i = 1 \quad \right\}$$

where matrices of the systems $\Sigma(\zeta)$ are such that:

$$A(\zeta) = \sum_{i=1}^{N} \zeta_i A^{[i]} \qquad B_w(\zeta) = \sum_{i=1}^{N} \zeta_i B_w^{[i]} \qquad B_u(\zeta) = \sum_{i=1}^{N} \zeta_i B_u^{[i]}$$
$$C_z(\zeta) = \sum_{i=1}^{N} \zeta_i C_z^{[i]} \qquad D_{zw}(\zeta) = \sum_{i=1}^{N} \zeta_i D_{zw}^{[i]} \qquad D_{gu}(\zeta) = \sum_{i=1}^{N} \zeta_i D_{gu}^{[i]}$$
$$C_y(\zeta) = \sum_{i=1}^{N} \zeta_i C_y^{[i]} \qquad D_{yw}(\zeta) = \sum_{i=1}^{N} \zeta_i D_{yw}^{[i]} \qquad D_{yu}(\zeta) = \sum_{i=1}^{N} \zeta_i D_{yu}^{[i]}$$

Similarly as for matrices of polytopic uncertain operators described in section 4, two sub types of polytopic systems are: parallelotopic and interval systems. Their definition follows the same rules as for uncertainties, see Examples 5.2, 5.3 and 5.4. As for the `uncertainty` objects, the function `u2poly` converts all parallelotopic and interval systems into polytopic ones, see Example 5.5.

Example 5.2 Define polytopic uncertain system

```
>> Svtx
Array of 4 systems
                n=2    mw=2    mu=1
 n=2    dx  =   A*x +  Bw*w +  Bu*u
pz=1     z  =  Cz*x + Dzw*w + Dzu*u
py=2     y  =  Cy*x + Dyw*w + Dyu*u
continuous time ( dx : derivative operator )

>> usPo=upoly(Svtx)
Uncertain model : polytope 4 vertices
-------- WITH --------
                n=2    mw=2    mu=1
 n=2    dx  =   A*x +  Bw*w +  Bu*u
pz=1     z  =  Cz*x + Dzw*w + Dzu*u
py=2     y  =  Cy*x + Dyw*w + Dyu*u
continuous time ( dx : derivative operator )
```

Example 5.3 Define parallelotopic uncertain system

```
>> whos Scntr Sax
  Name        Size                            Bytes  Class

  Sax         1x2x2                            3870  ssmodel object
  Scntr       1x2                              3670  ssmodel object

Grand total is 317 elements using 7540 bytes

>> usPa=uparal(Scntr,Sax)
Uncertain model : parallelotope 2 param
-------- WITH --------
               n=2    mu=2
 n=2    dx  =   A*x +  Bu*u
py=1     y  =  Cy*x + Dyu*u
continuous time ( dx : derivative operator )
```

Example 5.4 Define interval uncertain system

```
>> usIn = uinter(Smin,Smax)
Uncertain model : interval 3 param
-------- WITH --------
               n=4    mu=2
 n=4    dx  =   A*x +  Bu*u
py=3     y  =  Cy*x
continuous time ( dx : derivative operator )
```

Example 5.5 Convert to polytopic system

```
>> u2poly(usIn)
Uncertain model : polytope 8 vertices
-------- WITH --------
               n=4    mu=2
 n=4    dx  =   A*x +  Bu*u
py=3     y  =  Cy*x
continuous time ( dx : derivative operator )
```

6 Analysis and design problems: `ctrpb` object

The toolbox is focused on LMI methods for the robust performance analysis of certain and uncertain systems defined in sections 3 and 5. In the future versions of RoMulOC robust controller design with several performance specifications will be implemented.

6.1 Stability and performance

All results are Lyapunov-type and based on the following definitions for an LTI system without uncertainties:

$$\begin{aligned} \delta[x(t)] = &\quad Ax(t) + \quad B_w w(t) \\ z(t) = &\quad C_z x(t) + \quad D_{zw} w(t) \end{aligned}$$

6.1.1 Stability

First define the matrix R_s as either

$$R_{sc} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \qquad \text{or} \qquad R_{sd} = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

respectively for continuous-time and discrete-time systems. Stability is proved by the existence of a solution $\mathbf{P}$ to the LMI problem:

$$\mathbf{P} > \mathbb{0} \qquad\qquad \begin{bmatrix} \mathbb{1} & A^T \end{bmatrix} R_s \otimes \mathbf{P} \begin{bmatrix} \mathbb{1} \\ A \end{bmatrix} < \mathbb{0}$$

For stability analysis or stabilizing controller design a single function is defined: `stability`. Its usage is such as of Example 6.1 where `quiz` is a `ctrpb` object described later in sub-section 6.4.

Example 6.1 Define a stability analysis/design problem

```
>> quiz = stability( quiz, sys );
```

6.1.2 D-Stability

Pole location is considered for half planes and discs of the complex plane. To declare such regions a `region` object is available for the user, see Example 6.2.

Remark: In the case of real-valued matrices, pole location in a sector of the complex plane is equivalent to pole location in the inclined half plane with same angle as illustrated on Figure 6. Specifications on the time response, damping, natural frequency can be easily defined with the considered regions.

All the considered regions are in fact described by a quadratic constraint on the poles p such as:

$$\begin{pmatrix} 1 & p^* \end{pmatrix} R \begin{pmatrix} 1 \\ p \end{pmatrix} \leq 0$$

and the pole location is proved by the existence of a solution $\mathbf{P}$ to the LMI problem:

$$\mathbf{P} > \mathbb{0} \qquad\qquad \begin{bmatrix} \mathbb{1} & A^T \end{bmatrix} R \otimes \mathbf{P} \begin{bmatrix} \mathbb{1} \\ A \end{bmatrix} < \mathbb{0} \quad .$$

For all pole location problems the usage is such as of Example 6.3 where `reg` is a `region` object.

Example 6.2 Regions of the complex plane

```
>> help region
 --- help for region/region.m ---

  REGION - Define a region of the complex plane

  r = region('plane',a)     : half-plane such that Re(z)<=Re(a)
  r = region('plane',a,pi)  : half-plane such that Re(z)>=Re(a)
  r = region('plane',a,psi) : half-plane below the inclined line
                              that crosses the point 'a'
                              and makes an angle of 'psi' with the imag. axis

  r = region('disk',c,rad)  : disk centered at 'c' with radius 'rad'
                              default 'rad=1'
                              if 'rad<0' the region is the exterior of the disk.

  SEE ALSO dstability
```

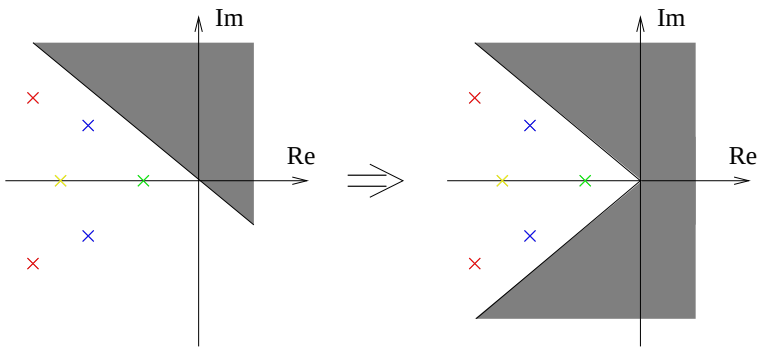


Figure 6: Sector pole location for real-valued matrices

Example 6.3 Define a D-stability analysis/design problem

```
>> quiz = dstability( quiz, sys, reg );
```

6.1.3 H_∞ performance

The H_∞ norm of the system is proved to be less than γ if there exists a solution $\mathbf{P}$ to the LMI problem:

$$\mathbf{P} > \mathbb{0} \qquad \begin{bmatrix} \mathbb{1} & A^T \\ \mathbb{0} & B_w^T \end{bmatrix} R_s \otimes \mathbf{P} \begin{bmatrix} \mathbb{1} & \mathbb{0} \\ A & B_w \end{bmatrix} + \begin{bmatrix} C_z^T C_z & C_z^T D_{zw} \\ D_{zw}^T C_z & D_{zw}^T D_{zw} - \gamma^2 \mathbb{1} \end{bmatrix} < \mathbb{0} \quad .$$

To define H_∞ problems the usage is such as of Example 6.4 where `perf` is the γ to test. If the argument `perf` is empty or omitted, γ is minimized.

Example 6.4 Define an H_∞ analysis/design problem

```
>> quiz = hinfty( quiz, sys, perf );
```

6.1.4 H_2 performance

The H_2 norm of the system is proved to be less than γ by the existence of a solution $\mathbf{P}$ to the LMI problem (D_{zw} should be zero for continuous-time systems):

$$\mathbf{P} > \mathbb{0} \qquad \begin{bmatrix} \mathbb{1} & A^T \end{bmatrix} R_s \otimes \mathbf{P} \begin{bmatrix} \mathbb{1} \\ A \end{bmatrix} + C_z^T C_z < \mathbb{0} \qquad \mathrm{Trace}(B_w^T \mathbf{P} B_w + D_{zw}^T D_{zw}) \leq \gamma^2 \quad .$$

To define H_2 problems the usage is such as of Example 6.5 where `perf` is the γ to test. If the argument `perf` is empty or omitted, γ is minimized.

Example 6.5 Define an H_2 analysis/design problem

```
>> quiz = h2( quiz, sys, perf );
```

6.1.5 Impulse-to-peak performance

The peak norm of the impulse response of the system is proved to be less than γ by the existence of a solution $\mathbf{P}$ to the LMI problem:

$$\mathbf{P} > \mathbb{0} \qquad \begin{bmatrix} \mathbb{1} & A^T \end{bmatrix} R_s \otimes \mathbf{P} \begin{bmatrix} \mathbb{1} \\ A \end{bmatrix} < \mathbb{0} \qquad B_w^T \mathbf{P} B_w \leq \gamma^2 \mathbb{1} \qquad C_z^T C_z \leq \mathbf{P} \qquad D_{zw}^T D_{zw} \leq \gamma^2 \mathbb{1} \quad .$$

To define impulse-to-peak problems the usage is such as of Example 6.6 where `perf` is the γ to test. If the argument `perf` is empty or omitted, γ is minimized.

Example 6.6 Define an impulse-to-peak analysis/design problem

```
>> quiz = i2p( quiz, sys, perf );
```

6.2 Robust methods

For each type of uncertain system (LFT and polytopic) the toolbox allows to build conservative LMI results that prove the stability or the specified performance. These LMI results are now briefly presented for the case of robust stability analysis.

6.2.1 Quadratic stability and quadratic separators

In the case of LFT uncertain systems robust stability can be proved by the existence of a unique quadratic Lyapunov function for all values of the uncertain parameters, $V(x) = x^T P x$, and a quadratic separator Θ such that the following constraints hold:

$$\mathbf{P} > \mathbb{0} \qquad \begin{bmatrix} \mathbb{1} & A^T \\ \mathbb{0} & B_\Delta^T \end{bmatrix} R_s \otimes \mathbf{P} \begin{bmatrix} \mathbb{1} & \mathbb{0} \\ A & B_\Delta \end{bmatrix} < \begin{bmatrix} C_\Delta^T & \mathbb{0} \\ D_{\Delta\Delta}^T & \mathbb{1} \end{bmatrix} \mathbf{\Theta} \begin{bmatrix} C_\Delta & D_{\Delta\Delta} \\ \mathbb{0} & \mathbb{1} \end{bmatrix}$$

$$\begin{bmatrix} \mathbb{1} & \Delta^T \end{bmatrix} \mathbf{\Theta} \begin{bmatrix} \mathbb{1} \\ \Delta \end{bmatrix} \leq \mathbb{0} \quad \forall \Delta \in \mathbb{\Delta}$$

where $\mathbb{\Delta}$ is the set of uncertainties. This set has infinite number of elements. To make the problem tractable one therefor needs an (eventually conservative) relaxation to describe the admissible separators with a finite number of constraints. The implemented relaxations are based on, [IH98]:

- vertex separators for polytopic uncertainties,
- D-scaling for full block dissipative uncertainties,
- DG-scaling for repeated scalar dissipative uncertainties.

The construction of these separators is not discussed here. To generate examples of separators, know that separators are built in a YALMIP format using a single internal function `separator`.

6.2.2 Quadratic-LFT Lyapunov function and quadratic separators

The previous result is conservative due to the choice of a unique Lyapunov function for all uncertainties. A second implemented result attenuates the conservatism with the use of a so called quadratic-LFT Lyapunov function:

$$V(x, \Delta) = x^T \begin{bmatrix} \mathbb{1} & \Delta_c^T \end{bmatrix} \mathbf{P} \begin{bmatrix} \mathbb{1} \\ \Delta_c \end{bmatrix} x \qquad \Delta_c = \Delta(\mathbb{1} - D_{\Delta\Delta}\Delta)^{-1} C_\Delta$$

The LMI results are then as follows:

$$\begin{bmatrix} \mathbb{1} & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \mathbb{1} & \mathbb{0} \\ A & B_\Delta & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \mathbb{1} \end{bmatrix}^T R_s \otimes \mathbf{P} \begin{bmatrix} \mathbb{1} & \mathbb{0} & \mathbb{0} \\ \mathbb{0} & \mathbb{1} & \mathbb{0} \\ A & B_\Delta & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \mathbb{1} \end{bmatrix} < \begin{bmatrix} C_\Delta & D_{\Delta\Delta} & \mathbb{0} \\ C_\Delta A & C_\Delta B_\Delta & D_{\Delta\Delta} \\ \mathbb{0} & \mathbb{1} & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \mathbb{1} \end{bmatrix}^T \mathbf{\Theta} \begin{bmatrix} C_\Delta & D_{\Delta\Delta} & \mathbb{0} \\ C_\Delta A & C_\Delta B_\Delta & D_{\Delta\Delta} \\ \mathbb{0} & \mathbb{1} & \mathbb{0} \\ \mathbb{0} & \mathbb{0} & \mathbb{1} \end{bmatrix}$$

where $\mathbf{\Theta}$ is a separator for the structured uncertainty $\mathrm{diag}(\Delta, \Delta)$.

6.2.3 Quadratic stability and vertices

In the case of polytopic uncertainties specific methods are implemented. The first one is based on a unique Lyapunov function $V(x) = x^T \mathbf{P} x$. The stability is then proved by the LMI constraints:

$$\mathbf{P} > \mathbb{0} \qquad \begin{bmatrix} \mathbb{1} & A^{[i]T} \end{bmatrix} R_s \otimes \mathbf{P} \begin{bmatrix} \mathbb{1} \\ A^{[i]} \end{bmatrix} < \mathbb{0} \qquad \forall i \in \{1 \dots N\} \ .$$

6.2.4 Polytopic Lyapunov function and slack variables

A less conservative result, [PABB00], based on a polytopic Lyapunov function is also implemented. The result is such that the Lyapunov function is:

$$V(x,\zeta) = x^T \left(\sum_{i=1}^{N} \zeta_i \mathbf{P}^{[i]} \right) x$$

and the LMI constraints are:

$$\mathbf{P}^{[i]} > \mathbb{0} \qquad R_s \otimes \mathbf{P}^{[i]} + \left[\begin{array}{c} {A^{[i]}}^T \\ -\mathbb{1} \end{array} \right] \mathbf{G}^T + \mathbf{G} \left[\begin{array}{cc} A^{[i]} & -\mathbb{1} \end{array} \right] < \mathbb{0} \qquad \forall i \in \{1 \dots N\} \ .$$

6.2.5 Other methods for the next versions

Currently many other methods are developed such as [RP01, Bli04]. In cooperation with their authors we hope to integrate these results to the present tool as soon as possible.

6.3 Analysis and design problems

At this time, only robust analysis methods are implemented in RoMulOC. The next version with state-feedback and full-order dynamic output-feedback is expected for the autumn 2006. For these design problems, LMI conditions will be derived from the analysis inequalities with the help of the classical linearising changes of variables described in [BGP89] for the state-feedback case and [SGC97] for the full-order output-feedback case.

The toolbox is developed in order to give the possibility to define multi-objective design problems in a simple manner. Example 6.7 illustrates a mixed H_2/H_∞ declaration.

Example 6.7 Define a mixed H_2/H_∞ design problem

```
>> quiz = h2( quiz, sys1 );
>> quiz = hinfty( quiz, sys2, gamma );
```

6.4 Define and append a `ctrpb` object

All the control problems of the type considered in the toolbox depend on four elements:

- uncertainty modeling (LFT or Polytopic system),
- type of controller (state-feedback or full-order output-feedback),
- type of method (quadratic stability or parameter-dependent Lyapunov functions),
- stability or performance (pole location, H_∞, H_2 or impulse-to-peak).

For each case a specific Matlab function is developed. These functions are not accessible to the user but are called inside the `stability`, `dsatbility`, `hinfty`, `h2` and `i2p` functions. The system is always specified by the second input argument of the functions while the type of controller and the type of method are specified by a `ctrpb` object called here `quiz`, see Example 6.8.

Example 6.8 Define a control problem (`ctrpb`)

```
>> quiz = ctrpb;
 CHOOSE A CONTROL PROBLEM
(a) Analysis
(b) State-Feedback design
(c) Full-Order Dynamic Output-Feedback
choice > a
 CHOOSE A TYPE OF LYAPUNOV FUNCTION
(a) Unique over all uncertainties (quadratic stability)
(b) Quadratic w.r.t. del*(I-Ddd*del)^-1*Cd (for LFT SSMODEL objects)
(c) Polytopic (for polytopic USSMODEL objects)
choice > a

>> quiz
control problem: ANALYSIS
Lyapunov function: UNIQUE (quadratic stability)
No specified performance
```

Once the `quiz` variable is declared, the functions `stability`, `dsatbility`, `hinfty`, `h2` and `i2p` allow to append the object LMI constraints correspondingly to required stability or performances. Example 6.9 where `usys` is a `ussmodel` of LFT type named `Demo example 1` illustrates such a declaration.

Example 6.9 Append a control problem (`ctrpb`)

```
>> quiz = hinfty( quiz, us )
control problem: ANALYSIS
Lyapunov function: UNIQUE (quadratic stability)
Specified performances / systems:
# Hinfinity / Demo Example 1
```

6.5 Solve a control problem

The user of the RoMulOC toolbox may not know the structure of the `ctrpb` object and simply choose to solve the problem and get the solution. This is the case in Example 6.10.

For more details about solving control problems (`ctrpb` objects) in RoMulOC, the reader should get information about the YALMIP parser for optimization problems. Indeed, the `ctrpb` object contains the LMI contraints (`lmi`) involving matrix variables (`vars`) and a linear objective that can be visualized as shown in Example 6.11.

One of the main features of YALMIP is the possibility to call various SDP solvers. The usage here is the same as for YALMIP (`sdpsettings`) and is illustrated in Example 6.12.

Example 6.10 Solve a control problem (`ctrpb`)

```
>> solvesdp( quiz )
SeDuMi 1.05R5 by Jos F. Sturm, 1998, 2001-2003.
Alg = 2: xz-corrector, Step-Differentiation, theta = 0.250, beta = 0.500
eqs m = 60, order n = 33, dim = 337, blocks = 5
nnz(A) = 563 + 0, nnz(ADA) = 3132, nnz(L) = 1596
 it :     b*y       gap    delta  rate   t/tP*  t/tD*   feas cg cg
  0 :            5.67E+02 0.000
  1 :  -1.79E+00 1.16E+02 0.000 0.2040 0.9028 0.9000   0.97  1  1
  2 :  -7.25E+00 3.07E+01 0.000 0.2650 0.9000 0.9121   0.07  1  1
...
 33 :  -8.17E+00 7.19E-11 0.000 0.5127 0.9000 0.9000   0.44 19 24
 34 :  -8.17E+00 3.01E-11 0.000 0.4180 0.9000 0.9000   0.58 19 25
Run into numerical problems.
iter seconds digits       c*x               b*y
 34      3.6   Inf -8.1713266118e+00 -8.1713258776e+00
|Ax-b| =   3.6e-08, [Ay-c]_+ =   0.0E+00, |x|=  1.3e+01, |y|=  6.3e+04
Max-norms: ||b||=1, ||c|| = 2,
Cholesky |add|=2, |skip| = 10, ||L.L|| = 500000.
Warning: Numerical problems (SeDuMi)

Hinfinity norm = 2.85855 is guaranteed for all uncertainties

ans =

    2.8586
```

Example 6.11 LMIs and variables of a control problem (`ctrpb`)

```
>> quiz.lmi
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
|   ID|      Constraint|                       Type|                    Tag|
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
|   #1|   Numeric value|            Element-wise 1x1|   multi-conv : Zu3>=0|
|   #2|   Numeric value|     Matrix inequality 2x2|   multi-conv : Zu4>=0|
|   #3|   Numeric value|     Matrix inequality 3x3|     separate vertex 1|
|   #4|   Numeric value|            Element-wise 1x1|      Dscaling : Du1>0|
|   #5|   Numeric value|            Element-wise 1x1|      Dscaling : Du2>0|
|   #6|   Numeric value|   Matrix inequality 16x16|           Var Lyap <0|
|   #7|   Numeric value|     Matrix inequality 8x8|               Lyap >0|
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
>> quiz.vars

ans =

    [ 8x8  sdpvar]    'Lyapunov matrix'
    [ 1x1  sdpvar]    'g > (Hinf cost)^2'
    [13x13 sdpvar]    'Quadratic separator'

>> quiz.vars{1}
Linear matrix variable 8x8 (symmetric, real, eigenvalues between [1.1228,153.6718])
```

Example 6.12 Solve control problem with SDPT3 solver and silent display

```
>> opt = sdpsettings( 'solver', 'sdpt3', 'verbose', 0 );
>> Hinf = solvesdp( quiz, opt )

Hinf =

    2.8585
```

Contents

List of examples

References

[BGP89] J. Bernussou, J.C. Geromel, and P.L.D. Peres. A linear programing oriented procedure for quadratic stabilization of uncertain systems. *Systems & Control Letters*, 13:65–72, July 1989.

[Bli04] P.-A. Bliman. A convex approach to robust stability for linear systems with uncertain scalar parameters. *SIAM J. Control and Optimization*, 42:2016–2042, 2004.

[GdOB02] J.C. Geromel, M.C. de Oliveira, and J. Bernussou. Robust filtering of discrete-time linear systems with parameter dependent Lyapunov functions. *SIAM J. Control and Optimization*, 41:700–711, 2002.

[IH98] T. Iwasaki and S. Hara. Well-posedness of feedback systems: Insights into exact robustness analysis and approximate computations. *IEEE Trans. on Automat. Control*, 43(5):619–630, 1998.

[Löf04a] J. Löfberg. *YALMIP : A Toolbox for Modeling and Optimization in MATLAB*, 2004.

[Löf04b] J. Löfberg. YALMIP : A toolbox for modeling and optimization in MATLAB. In *Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004.

[PABB00] D. Peaucelle, D. Arzelier, O. Bachelier, and J. Bernussou. A new robust D-stability condition for real convex polytopic uncertainty. *Systems & Control Letters*, 40(1):21–30, May 2000.

[RP01] D.C.W. Ramos and P.L.D. Peres. A less conservative LMI condition for the robust stability of discrete-time uncertain systems. *Systems & Control Letters*, 43:371–378, 2001.

[SGC97] C. Scherer, P. Gahinet, and M. Chilali. Multiobjective output-feedback control via LMI optimization. *IEEE Trans. on Automat. Control*, 42(7):896–911, July 1997.

[XLZZ04] L. Xie, L. Lilei, D. Zhang, and H. Zhang. Improved H_2 and H_∞ filtering for uncertain discrete-time systems. *Automatica*, 40:873–880, 2004.