We are interested in solving the following problem (either $\mathcal{H}_2$ or $\mathcal{H}_\infty$) in a data driven framework

$$\min_{K} \max_{i=1\ \dots\ n} \left\| \begin{bmatrix} W_{1o}\mathcal{S}_i \\ W_{2o}\mathcal{T}_i \\ W_{3o}\mathcal{U}_i \end{bmatrix} \right\| \tag{1}$$

$$\|W_{1c}\mathcal{S}_i\|_\infty \le 1 \quad \|W_{2c}\mathcal{T}_i\|_\infty \le 1 \quad \|W_{3c}\mathcal{U}_i\|_\infty \le 1 \quad \|W_{4c}K\|_\infty \le 1, \quad i = 1\ \dots\ n$$

where

$$\mathcal{S}_i = (1 + G_i K)^{-1},\ \mathcal{T}_i = 1 - \mathcal{S}_i,\ \mathcal{U}_i = K\mathcal{S}_i.$$

The controller is denoted by K and the different (non-parametric) models by

$$G_i \in \{G_1, \dots, G_n\},$$

`datadrivenACS` class

The `datadrivenACS` class in MATLAB (R2018a and later) handles the implementation using MOSEK 9.1 **and** MOSEK Fusion. To install MOSEK and MOSEK Fusion, refer to
`https://docs.mosek.com/9.1/toolbox/install-interface.html`.

Add the MOSEK Fusion java .jar to the environment:
`javaaddpath PATH/mosek/9.1/tools/platform/ARCH/bin/mosek.jar`
(replace `PATH` and `ARCH` with the correct path and architecture)

To check if the instalation has been successfull, the following tests can be made:

- Using `mosekdiag` in MATLAB, check if MOSEK 9.1 has been installed properly
- Using

  ```
  import mosek.fusion.*;
  M = Model()
  ```

  in MATLAB, check if no error are returned.

`datadrivenACS` class

The `datadrivenACS` can be invoked in MATLAB as follows:

```
P = datadrivenACS;
```

- `P`, the `datadrivenACS` object (similar to a struct, but with functions attached). The different accessible properties are
  - `Model`. Information about the model.
  - `Feedback`. Information about the feedback controller design
  - `Feedforward`. Information about the feedforward controller design
  - `Logs`. Logs of the different iterations

Model. Information about the model.

- Plant Model of the system. Accepts any type of SISO model than can be called by the freqresp function (ss, tf, frd, etc). For the multi-model case, the different models should be stacked:
  (G = stack(1,G1,G2,G3))
- Frequency Frequency vector at which the optimization problem is solved. If not specified, takes frequency values based on the model dynamics (through the nyquist command) for parametric models, or the frequency vector from the Plant if available.

`Feedback`. Information about the feedback controller design

- `objective` Objective of the optimization problem. All fields of `Objective` must either be a double, or callable by the `freqresp` function.

  `o2W1, o2W2, o2W3` correspond to the filters W_{1o}, W_{2o} and W_{2o} in

$$\min \max_{i=1 \ \dots \ n} \left\| \begin{bmatrix} W_{1o}\mathcal{S}_i \\ W_{2o}\mathcal{T}_i \\ W_{3o}\mathcal{U}_i \end{bmatrix} \right\|_2$$

  `oinfW1, oinfW2, oinfW3` correspond to the filters W_{1o}, W_{2o} and W_{3o} in

$$\min \max_{i=1 \ \dots \ n} \left\| \begin{bmatrix} W_{1o}\mathcal{S}_i \\ W_{2o}\mathcal{T}_i \\ W_{3o}\mathcal{U}_i \end{bmatrix} \right\|_\infty$$

`Feedback`. Information about the feedback controller design

- `constraints` Constraints of the optimization problem. All fields of `constraints` must either be a double, or callable by the `freqresp` function.

  `cinfW1`, `cinfW2`, `cinfW3` correspond to the filters W_{1c}, W_{2c} and W_{3c} in

$$\|W_{1c}\mathcal{S}_i\|_\infty \leq 1 \quad \|W_{2c}\mathcal{T}_i\|_\infty \leq 1 \quad \|W_{3c}\mathcal{U}_i\|_\infty \leq 1 \qquad i = 1 \ \ldots \ n$$

`Feedback`. Information about the feedback controller design

- `controller` Parameters related to the controller
  - `order`: Order of the resulting controller
  - `K0`: Initial controller without the fixed parts. Must be a discrete transfer function with sampling time T_s.
  - `Fy`: Fixed parts in the controller denominator. Must be a discrete transfer function with sampling time T_s
  -
  - `Ts`: Sampling time of the controller
- `P.setKinit(K)` is the lazy way of doing

```
P.Feedback.controller.K0 = K0
P.Feedback.controller.Ts = K.Ts
P.Feedback.controller.Fy = Fy
```

where $K(z) = K_0(z) \cdot \frac{1}{F_y(z)}$ and $F_y(z)$ the poles of the controller on the unit circle.

`Feedback`. Information about the feedback controller design

- `parameters` General parameters
  - `maxIter`: maximum number of iterations
  - `tol`: threshold relative decrease objective stopping criterion: $\frac{J[k]-J[k+1]}{J[k]} \leq tol$
  - `exit`: threshold objective stopping criterion $J[k] \leq tol$
  - `c0` Stability of the poles of the controller. Increasing `c0` will increase the likelihood of all poles of K remaining inside the unit circle. Usual values if not 0 : $10^{-3} - 10^{-1}$
  - `c1` Stability of the closed-loop. Increasing `c1` will increase the likelihood keeping the same number of encirclement in the Nyquist.
  - `c2` Minimum value of the (absolute value of the) controller numerator at $z = 1$.

`Feedforward`. Same parameters as `Feedback`.

Similar example example as MATLAB's *Simultaneous Stabilization Using Robust Control* example.

$$\min_K \left\| \begin{bmatrix} W_1 \mathcal{S} \\ W_2 \mathcal{T} \\ W_3 \mathcal{U} \end{bmatrix} \right\|_\infty$$

And the different models

$$G_{nom} = \frac{2}{s-2}$$

$$G_1 = G_{nom} \frac{1}{0.06s+1} \quad G_2 = e^{-0.02s} G_{nom} \quad G_3 = G_{nom} \frac{50^2}{s^2+10s+50^2}$$

$$G_4 = G_{nom} \frac{70^2}{s^2+28s+70^2} \quad G_5 = \frac{2.4}{s-2.2}. \quad G_6 = \frac{1.6}{s-1.6}$$

Same example for $\mathcal{H}_\infty$ controller synthesis using a data-driven approach:

```
P = datadrivenACS;

P.Model.Plant = Pnom; % see def of Pnom in MATLAB example

P.setKinit(3*(z-0.99)/(z-1))
P.Feedback.controller.order = 4;

P.Feedback.objective.oinfW1 = (s+10)/2/s;
P.Feedback.objective.oinfW2 = W2; % obtained from UCOVER
P.Feedback.objective.oinfW3 = 1/15;
P.Feedback.parameters.maxIter = 100;
[Kmat,obj] = solveFB(P);
KFB = computeKFB(P);
```

and $K_{init} = 3 \cdot (z - 0.99)/(z - 1)$ a stabilzing controller.

Example 1

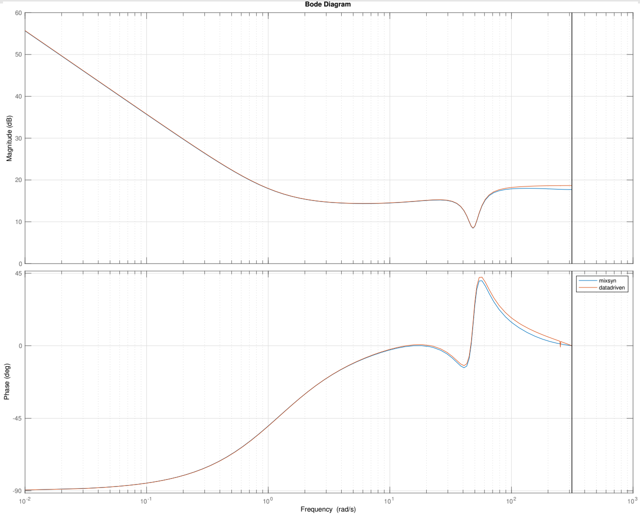


Figure 1: Controllers obtained using the different approaches

Remarks

- Default frequency grid might not be adequate, change `P.Model.Frequency` to an adequate vector of frequencies (linear or logarithmic spaced)
- The final datadriven controller is dependent on the initial controller, the frequency grid, etc.
- datadriven approach handles model uncertainty. No need for the multiplicative uncertainty filter.
- If robust performance is defined as $\|W_1 \mathcal{S}_i\|_\infty \leq 1 \; i = 1 \; \dots \; 6$, then $W_3 \mathcal{U}_i$ should be removed from the objective and changed to a constraint.

The same problem can be revisited, using multimodel uncetanity instead of multiplicative:

$$G_i \in \{G_1, \ \ldots, \ G_6\}.$$

and changing the weights on $\mathcal{U}_i$ in the objective to a constraint:

$$\min_{K} \ \max_{i=1 \ \ldots \ 6} \left\| \begin{bmatrix} W_1\mathcal{S}_i \\ 0 \cdot \mathcal{T}_i \\ 0 \cdot \mathcal{U}_i \end{bmatrix} \right\|_\infty = \|W_1\mathcal{S}_i\|\|_\infty$$

$$\left\| \frac{1}{15} \cdot \mathcal{U}_i \right\|_\infty \leq 1 \quad i = 1 \ \ldots \ 6$$

Revisited example for $\mathcal{H}_\infty$ controller synthesis using a non-parametric model

```
P = datadrivenACS;

P.Model.Plant = Parray; % see def, of Parray in MATLAB example
P.Model.Frequency = logspace(-2,log10(pi/Ts),400);

Kinit =  3*(z-0.99)/(z-1);
P.setKinit(Kinit)
P.Feedback.controller.order = 4;

P.Feedback.objective.oinfW1 = (s+10)/2/s;;
P.Feedback.constraints.cinfW3 = 1/15;

[Kmat,obj] = solveFB(P);
KFB = computeKFB(P);
```

Example 1 bis

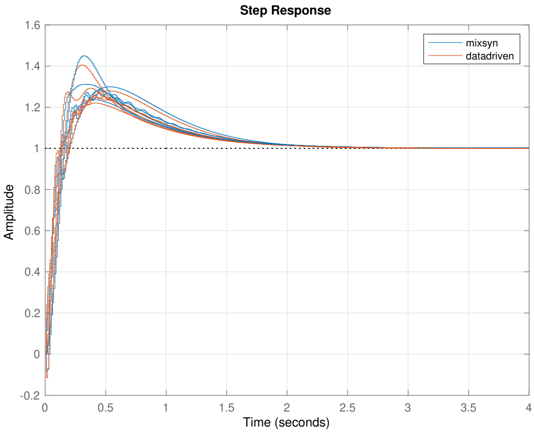


Figure 2: Step response using multimodel or multiplicative uncertainity

- By default, the optimization stops using $\texttt{tol} = 10^{-3}$ and $\texttt{maxIter} = 10$. In this case they have been increased to 10^{-6} and 200 respectively.
- The final datadriven controller is dependent on the initial controller, the frequency grid, etc.
- The mixsyn controller does not achieve robust performance, whereas the datadriven controller does (and with a lower order controller!).

Example for data-driven $\mathcal{H}_2$ controller synthesis

$$\min_{K} \max_{i=1 \,\dots\, 6} \left\| \begin{bmatrix} W_1 \mathcal{S}_i \\ 0 \cdot \mathcal{T}_i \\ W_3 \cdot \mathcal{U}_i \end{bmatrix} \right\|_2,$$

using the same models as in Example 1:

$$G_i \in \{G_1, \ \dots, \ G_6\}.$$

Example for $\mathcal{H}_2$ controller synthesis using data-driven method - MATLAB code.

```
P = datadrivenACS;

P.Model.Plant = Parray; % see def. of Parray in MATLAB example
P.Model.Frequency = logspace(-2,log10(pi/Ts),400);


P.setKinit(3*(z-0.99)/(z-1))
P.Feedback.controller.order = 4;

P.Feedback.objective.o2W1 = (s+10)/2/s;
P.Feedback.objective.o2W3 = 1/15;

[Kmat,obj] = solveFB(P);
KFB = computeKFB(P);
```

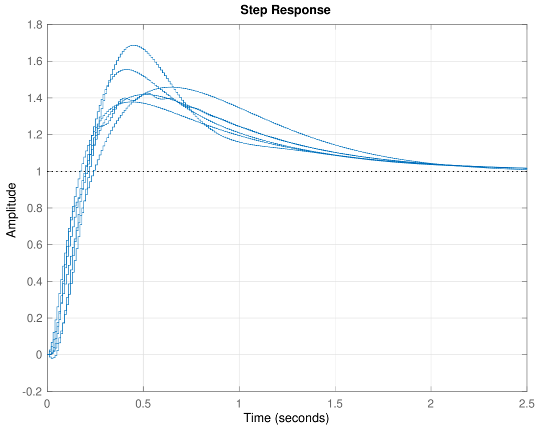
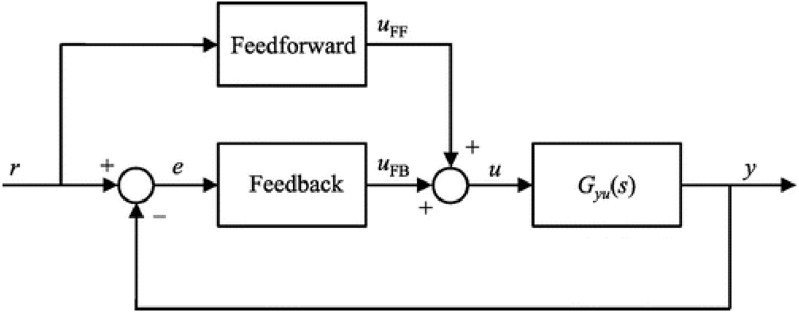


Figure 3: Close-loop step responses datadriven $\mathcal{H}_2$

Remarks

- In some cases, the poles on the controller become unstable. Increasing `P.Feedback.parameters.c0` to $10^{-3} - 10^{-1}$ will help prevent this.
- In some cases, the gain is not high enough at very low frequencies. Increasing the value `P.Feedback.parameters.c2`, corresponding the a lower bound of the numerator at $z = 1$, will increase the gain at low frequencies. Remember no free lunch: worse performance somewhere else.

Using the same framework, a feedfoward controller can also be designed using a non-parametric model



Bloc diagram with feedforward

The feedback controller should be computed before the feedfoward controller

The template feedforward problem (either $\mathcal{H}_2$ or $\mathcal{H}_\infty$) in a data driven framework:

$$\min_{K_{FF}} \max_{i=1\ \dots\ n} \left\| \begin{bmatrix} W_{1o}\mathcal{S}_{FF,i} \\ W_{2o}\mathcal{T}_{FF,i} \\ W_{3o}\mathcal{U}_{FF,i} \end{bmatrix} \right\| \tag{2}$$

$$\|W_{1c}\mathcal{S}_{FF,i}\|_\infty \leq 1 \quad \|W_{2c}\mathcal{T}_{FF,i}\|_\infty \leq 1 \quad \|W_{3c}\mathcal{U}_{FF,i}\|_\infty \leq 1 \quad \|W_{4c}K\|_\infty \leq i, \quad i = 1\ \dots\ n$$

$$G_i \in \{G_1,\ \dots,\ G_n\}$$

The new closed-loop transfer functions including feedforward are

$$S_{FF,i} = \frac{1 - G_i K_{FF}}{1 + G_i K_{FB}}, \quad \mathcal{T}_{FF,i} = 1 - S_{FF,i}, \quad U_{FF,i} = S_{FF,i} \cdot (K_{FF} + K_{FB}).$$

The problem can be convexified if K_{FB} is know.

Example for $\mathcal{H}_2$ feedforward controller synthesis using a non-parametric model

```
% solveFB(P); solve FB before

P.Feedforward.controller.order = 4;
P.Feedforward.controller.K0 =  1; % feedfoward: any stable controller

P.Feedforward.objective.o2W1     = 1e2/s; % objective
P.Feedforward.constraints.cinfW2 = 1/makeweight(1.5,0.25*pi/Ts,0);
% roll-of at high frequencies

solveFF(P);
KFF = computeKFF(P)
```

`P.Model` should have been set before, during the feedback controller design phase.