LPVTools

A Toolbox for Modeling, Analysis, and Synthesis of Parameter Varying Control Systems.

Authors:

Gary J. Balas

Andrew Packard

Peter J. Seiler

Arnar Hjartarson.

MUSYN Inc.

27 Summit Court

St Paul MN 55102

UNITED STATES

Tel: 651-239-7144

musyn@musyn.com

www.musyn.com

Getting Started with LPVTools

LPVTools is a toolbox aimed at helping users design parameter dependent control systems using the Linear Parameter-Varying framework (LPV). LPVTools contains data structures to represent LPV systems in MATLAB® and Simulink®, and a collection of functions and tools for model reduction, analysis, synthesis and simulation in the LPV framework.

Product Description

Tutorials

Product Description

What is LPVTools?

The Linear Parameter-Varying Toolbox (LPVTools) is a toolbox for modeling, analysis and synthesis in the Linear Parameter-Varying framework (LPV). LPV framework provides a mathematically rigorous approach to the design of gain-scheduled controllers, with powerful guarantees on their performance and robustness in the presence of time-varying dynamics and uncertainty.

LPVTools provides LPV data structures and a set of tools for modeling, simulation, analysis and synthesis in the LPV framework. Its capabilities include tools for synthesis of parameter-varying output feedback controllers, state-feedback controllers, and estimators, which yield optimized performance for a given set of allowable parameter trajectories. Tools are provided for analysis of the stability and input-to-output gain of LPV systems (with and without uncertainty). Tools are provided for performing model reduction on LPV models. And finally, tools are provided for simulating the time-domain response of LPV systems along user-supplied parameter trajectories.

Key Features

- Modeling of parameter dependent systems and gain-scheduled control laws.
- LPV analysis and control synthesis.
- Simulation of LPV systems.
- Model reduction for parameter-dependent systems.

System Requirments

LPVTools requires MATLAB®, Simulink®, the Control System Toolbox®, and the Robust Control Toolbox®. LPVTools makes use of the Control System and Robust Control Toolbox's data structures, control synthesis and analysis algorithms.

About the Authors

LPVTools is developed by Drs. Gary J. Balas, Andrew Packard, Peter J. Seiler, and Arnar Hjartarson of MUSYN Inc.

Linear Parameter-Varying Systems

Contents

Linear Parameter-Varying Systems

LPV systems are time-varying, state-space models of the form:

$$\begin{bmatrix} \dot{x}(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} A(\rho(t)) & B(\rho(t)) \\ C(\rho(t)) & D(\rho(t)) \end{bmatrix} \begin{bmatrix} x(t) \\ u(t) \end{bmatrix} \qquad (1)$$

where $\rho \in \mathcal{R}^{n_\rho}$ is a vector of measurable parameters, $y \in \mathcal{R}^{n_y}$ is a vector of outputs, $x \in \mathcal{R}^{n_x}$ is the state vector, $u \in \mathcal{R}^{n_u}$ is a vector of inputs, and $A \in \mathcal{R}^{n_x \times n_x}$, $B \in \mathcal{R}^{n_x \times n_u}$, $C \in \mathcal{R}^{n_y \times n_x}$ and $D \in \mathcal{R}^{n_y \times n_u}$ are parameter dependent matrices.

The LPV system in Equation 1 depends on a set of time-varying parameters ρ. The trajectories of the parameters are assumed to take on values in a known compact set $\mathcal{P} \subseteq \mathcal{R}^{n_\rho}$, and to have known bounds on their derivatives with respect to time: $\overline{\nu} \leq \dot{\rho} \leq \underline{\nu}$, where $\overline{\nu}$ and $\underline{\nu} \in \mathcal{R}^{n_\rho}$. A trajectory is said to be "rate unbounded" if $\overline{\nu} = \infty$ and $\underline{\nu} = -\infty$.

For control design in the LPV framework, it is further assumed that time variations of $\rho(t)$ are not known in advance, and that the parameter values are measured and available in real-time with sensors. The controller produced is itself a LPV system which is optimized for the parameter trajectories in $\rho \in P$ subject to $\overline{\nu} \leq \dot{\rho} \leq \underline{\nu}$, and dependent on real-time measurements of the parameter.

LPVTools implements data structures for two types of LPV modeling approaches: i) Linearizations on a gridded domain, and ii) Linear Fractional Transformations (LFT).

Grid-Based LPV Models

Linearizations on a gridded domain are referred to as *grid-based LPV models*, because they require the user to divide the parameter domain into a grid of parameter values, and then specify the linear dynamics at each grid point. Linearizations on the gridded domain are obtained through Jacobian linearization at each grid point (e.g. batch linearization of Simulink models). All the linearized systems on the grid have identical inputs u, outputs y and state vectors x. Each linearization approximates the system's dynamics in the vicinity of a particular grid point, and the grid of linearizations captures the system's parameter dependence implicitly.

Figure 1 illustrates the concept. A nonlinear model is linearized along a grid of Mach and altitude values, resulting in an array of linear systems. Together the linearizations form a LPV system approximation of the original system. Linearization based LPV models do not require any special dependence on the parameter vector. This approach is motivated by the traditional gain-scheduling framework in aircraft flight control, for which models are typically constructed as linearizations around various flight operating points.

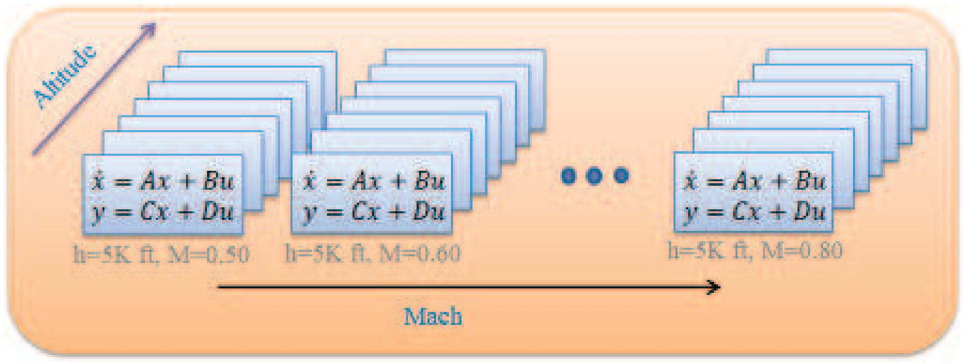


Figure 1: LPV model defined on a rectangular grid.

Further Reading

1. Marcos, A. and Balas G., "Development of Linear-Parameter-Varying Models for Aircraft," *Journal of Guidance, Control, and Dynamics*, Vol. 27, no. 2, 2004, pp 218-228, doi: 10.2514/1.9165.
2. B. Takarics and P. Seiler, "Gain Scheduling for Nonlinear Systems via Integral Quadratic Constraints," *accepted to the American Control Conference*, 2015.

LFT-Based LPV Models

An LPV model in Linear Fractional Transformation (LFT) form is an interconnection of a block that represents the plant's nominal dynamics (linear, time invariant), and a block that contains the time-varying parameters which the system depends on.

In the LFT-based approach the LPV system in Equation 1 is expressed as the interconnection of the blocks M and Δ_ρ, as seen in Figure 2.

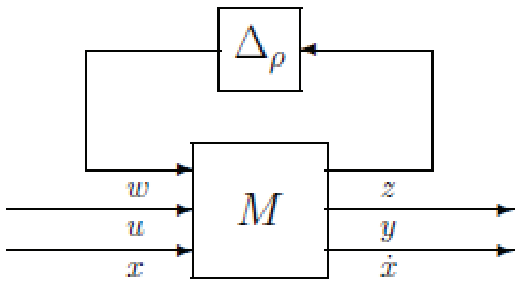


Figure 2: An LPV system in LFT form.

where M is a constant matrix such that

$$\begin{bmatrix} z(t) \\ y(t) \\ \dot{x}(t) \end{bmatrix} = M \begin{bmatrix} w(t) \\ u(t) \\ x(t) \end{bmatrix} \qquad (2)$$

and Δ_ρ is a diagonal matrix

$$\Delta_\rho = \left[\begin{array}{cccc} \rho_1(t)I_{r_1} & 0 & \dots & 0 \\ 0 & \rho_2(t)I_{r_2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \rho_{n_\rho}(t)I_{r_{n_\rho}} \end{array} \right] \qquad (3)$$

such that $w = \Delta_\rho z$. Where I_{r_i} indicates a $r_i \times r_i$ identity matrix, for positive integers $r_1, \dots, r_{n_\rho}$, and $\rho_1, \dots, \rho_{n_\rho}$ represent the elements of the parameter vector ρ. Note that the parameter dependence of a LFT model is modeled explicitly, and the LFT form can only be used to model LPV systems whose state matrices are rational functions of the parameters.

Further Reading

1. Cockburn, J. C. and Morton, B. G. "Linear Fractional Representations of Uncertain Systems," *Automatica*, Vol. 33, no. 7, 1997, pp 1263-1271, doi: 10.1016/S0005-1098(97)00049-6.
2. J. Doyle, A. Packard, and K. Zhou, "Review of LFTs, LMIs, and μ," *Proceedings of the 30th IEEE Conference on Decision and Control*, 1991, doi: 10.1109/CDC.1991.261572.
3. J.F. Magni, S. Bennani, J. P. Dijkgraaf, "An Overview of System Modelling in LFT Form," in *Advanced Techniques for Clearance of Flight Control Laws*, Springer-Verlag, Germany, pp. 169-195, 2002, doi: 10.1007/3-540-45864-6_11.

Published with MATLAB® R2014b

LPVTools Data Structures

LPVTools is implemented using object-oriented programming. The toolbox introduces several class-based data structures for modeling LPV systems. These data structures extend the functionality associated with standard MATLAB data structures from the Control Systems Toolbox and the Robust Control Toolbox into the LPV framework. This is pictorially represented in Table 1.

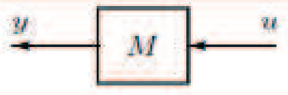
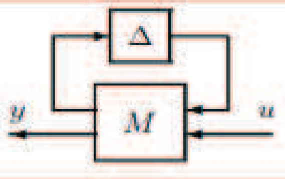
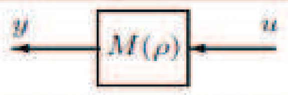
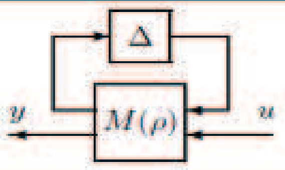
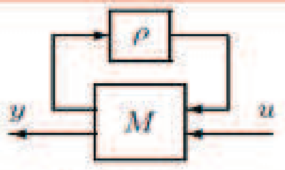
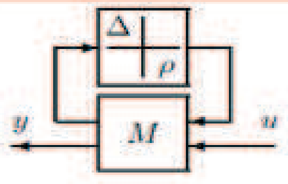
Object Type	Block	Matrix	System	Frequency Response
Nominal		double	ss	frd
Uncertain		umat	uss	ufrd
Nominal Gridded LPV		pmat	pss	pfrd
Uncertain Gridded LPV		upmat	upss	upfrd
Nominal LFT LPV		pmatlft	psslft	
Uncertain LFT LPV		pmatlft	psslft	

Table 1: Relation between LPVTools and MATLAB objects.

Table 1 shows the relation between the core LPVTools data objects and existing MATLAB objects. The first row of the table (``Nominal") shows the basic MATLAB objects: Matrices are `double` objects, state-space systems are `ss` objects, and frequency responses are `frd` objects. The third row of the table (``Nominal Gridded LPV") shows the corresponding core grid-based LPV objects. The core data structure for grid-based LPV models is the `pss` (denoting parameter-varying state space model), which stores the LPV system as a state space array (`ss`) defined on a finite, gridded domain. The notions of parameter-varying matrices and parameter-varying frequency responses arise naturally to complement the `pss` objects. LPV systems are time-varying and hence frequency responses can not be used to represent the system behavior as parameters vary. However frequency responses are useful to gain intuition about the system performance at fixed locations in the operating domain. LPVTools represents parameter varying matrices and frequency responses by `pmat` and `pfrd` data objects, respectively. These two data objects are both stored as a data array defined on a gridded domain. A `pmat` stores a `double` array, while a `pfrd` stores an array of frequency responses (`frd` object in the Control System Toolbox). The (`pmat`, `pss`, `pfrd`) objects should be viewed as parameter-varying extensions of the standard MATLAB and Control Systems Toolbox objects (`double`, `ss`, `frd`).

The second row of the table (``Uncertain") shows the equivalent objects used to represent uncertainty: Uncertain matrices, state space systems, and frequency responses are represented by `umat`, `uss`, and `ufrd` objects, respectively (from the Robust Control Toolbox). The fourth row of Table 1 (``Uncertain Gridded LPV") shows the corresponding parameter-varying objects with uncertainty: Uncertain parameter-varying matrices, state space systems, and frequency responses are represented by `upmat`, `upss`, and `upfrd` objects, respectively. These objects enable the integration of uncertainty into LPV models. The (`upmat`, `upss`, `upfrd`) objects should be viewed as parameter-varying extensions of the uncertain Robust Control Toolbox objects (`umat`, `uss`, `ufrd`).

LPVTools represents LFT-based parameter varying matrices and state-space systems by `plftmat` and `plftss` data objects, respectively. Uncertainty can be integrated into the `plftmat`, and `plftss` objects, allowing these data objects to model systems with, and without uncertainty. The `plftmat` and `plftss` objects should be viewed as LFT-based parameter-varying extensions of the standard MATLAB, Control System Toolbox, and Robust Control Toolbox objects `double`, `ss`, `umat`, and `uss`, as seen in rows five ("Nominal LFT LPV") and six ("Uncertain LFT LPV") in Table 1.

Published with MATLAB® R2014b

Tutorials

The following tutorials demonstrate some of the key features of LPVTools.

Tutorials

Modeling Gridded LPV Systems

Contents

Introduction

Let $G(\rho)$ represent a state-space system, which depends on Mach (M) and altitude (h) values, that has the standard form:

$$\begin{bmatrix} \dot{x}(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} A(\rho(t)) & B(\rho(t)) \\ C(\rho(t)) & D(\rho(t)) \end{bmatrix} \begin{bmatrix} x(t) \\ u(t) \end{bmatrix} \qquad (1)$$

where $\rho = [M, h]^T$.

A grid-based LPV model of this system is a collection of linearizations on a gridded domain of parameter values, as seen in Figure 1. Each linearization approximates the system's dynamics in the vicinity of a particular grid point, and the grid of linearizations captures the system's parameter dependence implicitly.

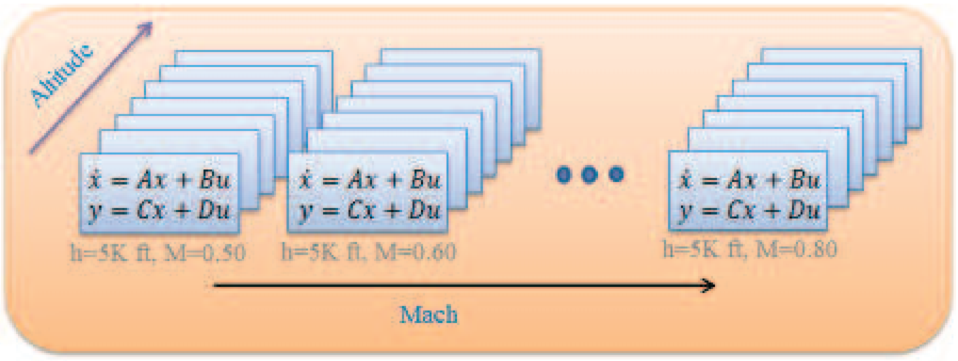


Figure 1: Approximate $G(\rho)$ as a grid-based LPV model on a (M, h) grid.

In LPVTools there are two ways to construct gridded LPV systems. When an analytical model of the LPV system is available it can be constructed using `pgrid` objects. More commonly, it is constructed directly from numerical data representing the linearized model at various grid points (e.g. data from batch linearization of Simulink models).

Construction from Data

Jacobian Linearization is the predominant method of constructing grid-based LPV models. Lets assume G has been linearized at a grid of Mach and altitude values: $(M, h) = [0.50.60.70.8] \times [5000, 6000, 7000]\ ft$ yielding a 4x3 `ss` array of linearizations. The process of creating a gridded LPV model from this data is as follows:

Load `Gss`, a 4x3 `ss` array of linearizations of $G(\rho)$. Every model in the `ss` array shares the same state vector, input vector, and output vector. This is required for the the construction of `pss` systems.

```
load GData
size(Gss)
```

```
4x3 array of state-space models.
Each model has 1 outputs, 1 inputs, and 1 states.
```

Define an `rgrid` object to represent the grid of Mach and altitude values:

```
Mach = [0.5 0.6 0.7 0.8];
altitude = [5000,6000,7000];
Domain = rgrid({'M','h'},{Mach,altitude})
```

```
RGRID with the following parameters:
  M: Gridded real, 4 points in [0.5,0.8], rate bounds [-Inf,Inf].
  h: Gridded real, 3 points in [5e+03,7e+03], rate bounds [-Inf,Inf].
```

Combine the state-space array in `Gss` with the `rgrid` object to form a `pss`:

```
Glpv = pss(Gss,Domain)
```

```
PSS with 1 States, 1 Outputs, 1 Inputs, Continuous System.
The PSS consists of the following blocks:
  M: Gridded real, 4 points in [0.5,0.8], rate bounds [-Inf,Inf].
  h: Gridded real, 3 points in [5e+03,7e+03], rate bounds [-Inf,Inf].
```

Note that an explicit model of the parameter dependence in $G(\rho)$ is not required to construct `Glpv`. Instead the array of linearizations captures the parameter dependence of $G(\rho)$ implicitly. This is an advantage when dealing with complex nonlinear models, for which an analytical linearization may not be available.

Construction from analytical model

Lets assume an analytical model of $G(\rho)$ is available:

$$\begin{aligned} \dot{x} &= -Mx + Mhu \\ y &= x \end{aligned} \qquad (2)$$

In this case the `pss` can be constructed using a `pgrid` object. The `pgrid` represents a time-varying real parameter and its values.

Define the Mach number as a time-varying parameter with 4 points in the range [0.5 0.8]

```
M = pgrid('M',0.5:0.1:0.8)
```

```
Gridded real parameter "M" with 4 points in [0.5,0.8] and rate bounds [-Inf,Inf].
```

Define thealtitude as a time-varying parameter with 3 points in the range [5000 7000]

```
h = pgrid('h',[5000 6000 7000])
```

```
Gridded real parameter "h" with 3 points in [5e+03,7e+03] and rate bounds [-Inf,Inf].
```

Define the `pss` representation of $G(\rho)$:

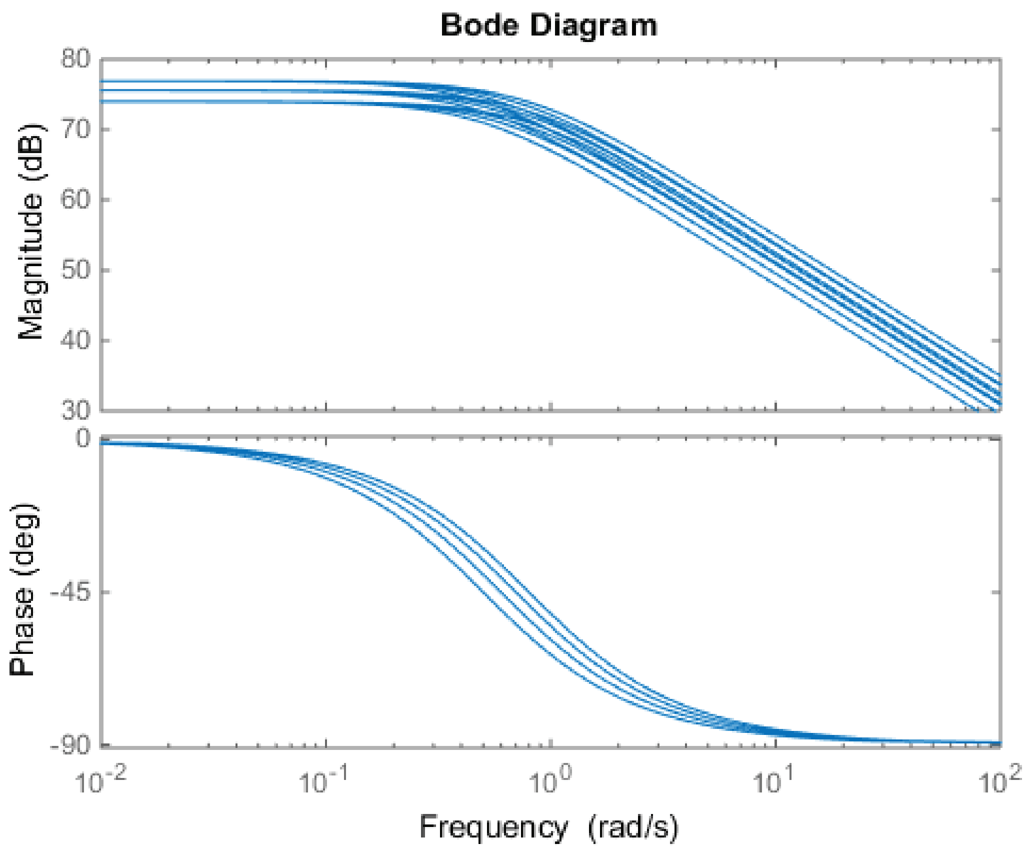
```
Glpv2 = ss(-M,M*h,1,0)
```

```
PSS with 1 States, 1 Outputs, 1 Inputs, Continuous System.
The PSS consists of the following blocks:
  M: Gridded real, 4 points in [0.5,0.8], rate bounds [-Inf,Inf].
  h: Gridded real, 3 points in [5e+03,7e+03], rate bounds [-Inf,Inf].
```

A study of the pointwise LTI dynamics

A Bode plot of Glpv demonstrates how the dynamics of of $G(\rho)$ change as a function of Mach and altitude. Each frequency response in the Bode plot corresponds to the LTI dynamics at a single grid point, when the Mach and altitude is held fixed at that grid point.

```
bode(Glpv)
```



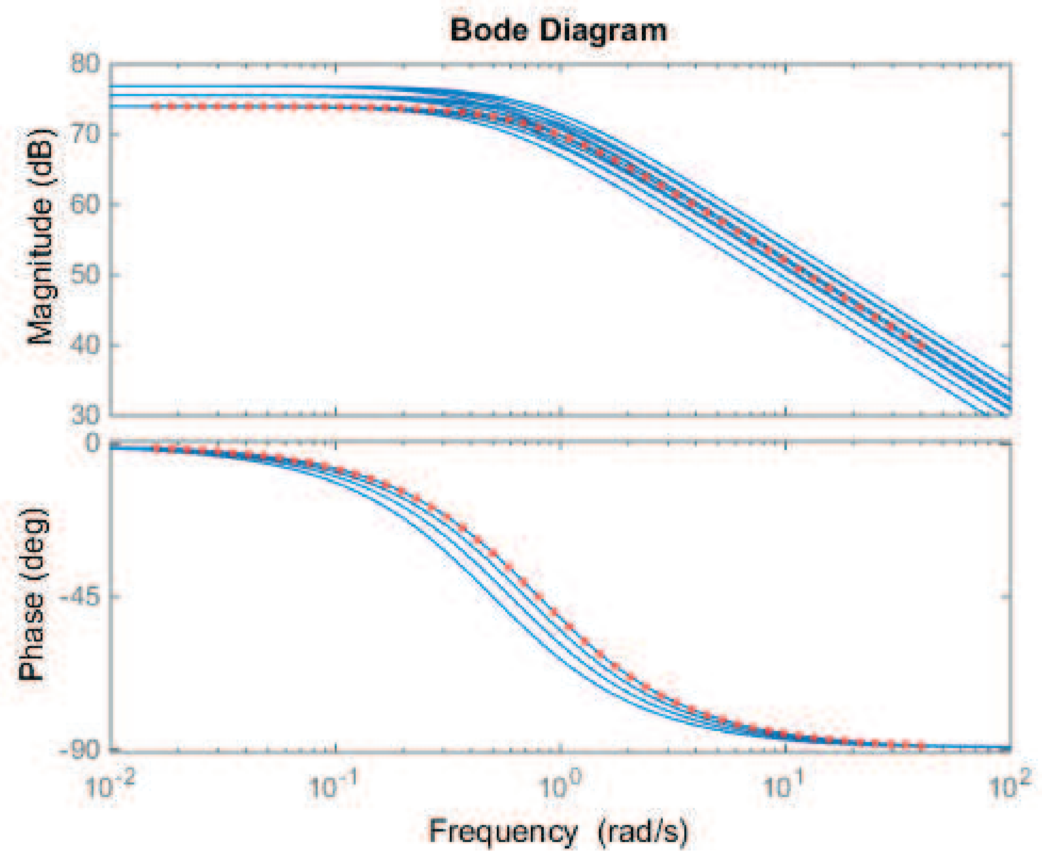
The dynamics at a particular (M, h) grid point at easily retrieved from the `pss` using the `.value` method:

Grab the LTI system associated with $M = 0.8$ and $h = 5000$:

```
Gpoint = Glpv.value('M',0.8,'h',5000);
```

Compare the dynamics associated with $M = 0.8$ and $h = 5000$ against the dynamics at the other points:

```
bode(Glpv)
hold on
bode(Gpoint,'r.')
```



Its also possible to retrieve the data associated with several grid points. To illustrate this we will look at the Bode plot of the dynamics associated with $M = 0.8$ and $h = [500060007000]$.

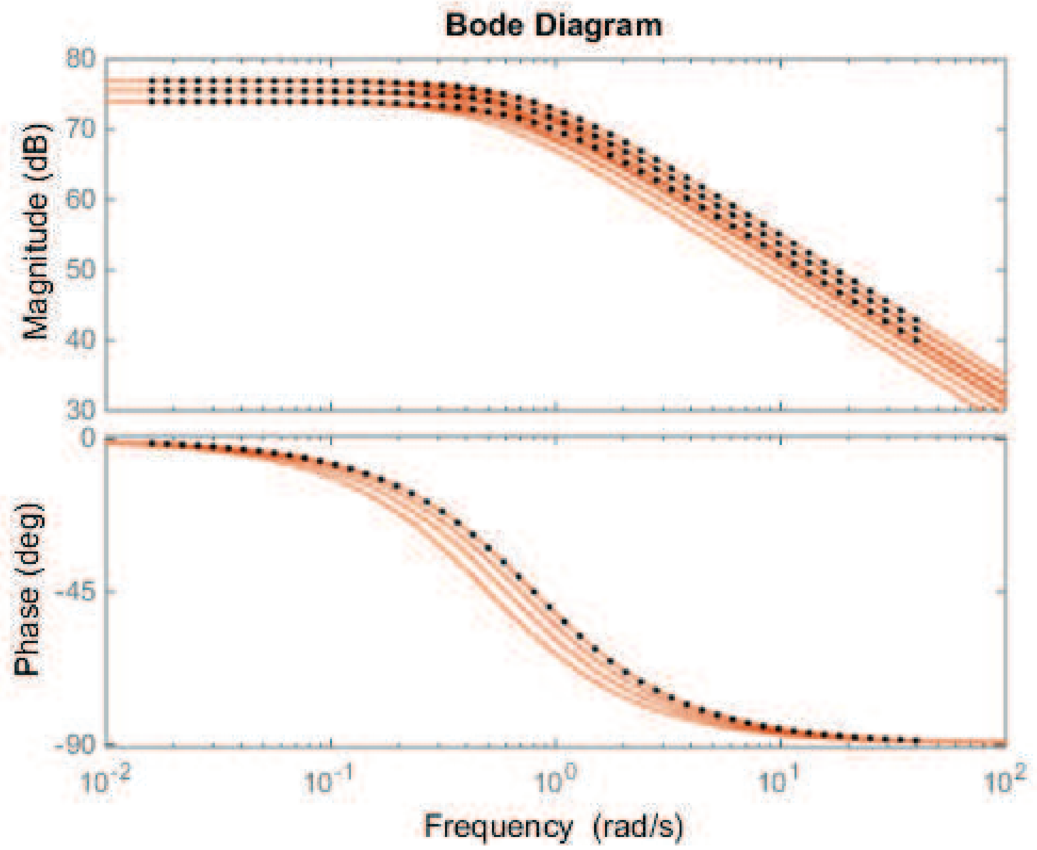
Start by grabbing the data associated with $M = 0.8$ and $h = [500060007000]$

```
Gmach = lpvsplit(Glpv,'M',0.8)
```

```
PSS with 1 States, 1 Outputs, 1 Inputs, Continuous System.
The PSS consists of the following blocks:
  M: Gridded real, 1 points in [0.8,0.8], rate bounds [-Inf,Inf].
  h: Gridded real, 3 points in [5e+03,7e+03], rate bounds [-Inf,Inf].
```

Compare the dynamics associated with $M = 0.8$ and $h = [500060007000]$ against the dynamics at the other points:

```
bode(Glpv)
hold on
bode(Gmach,'k.')
```



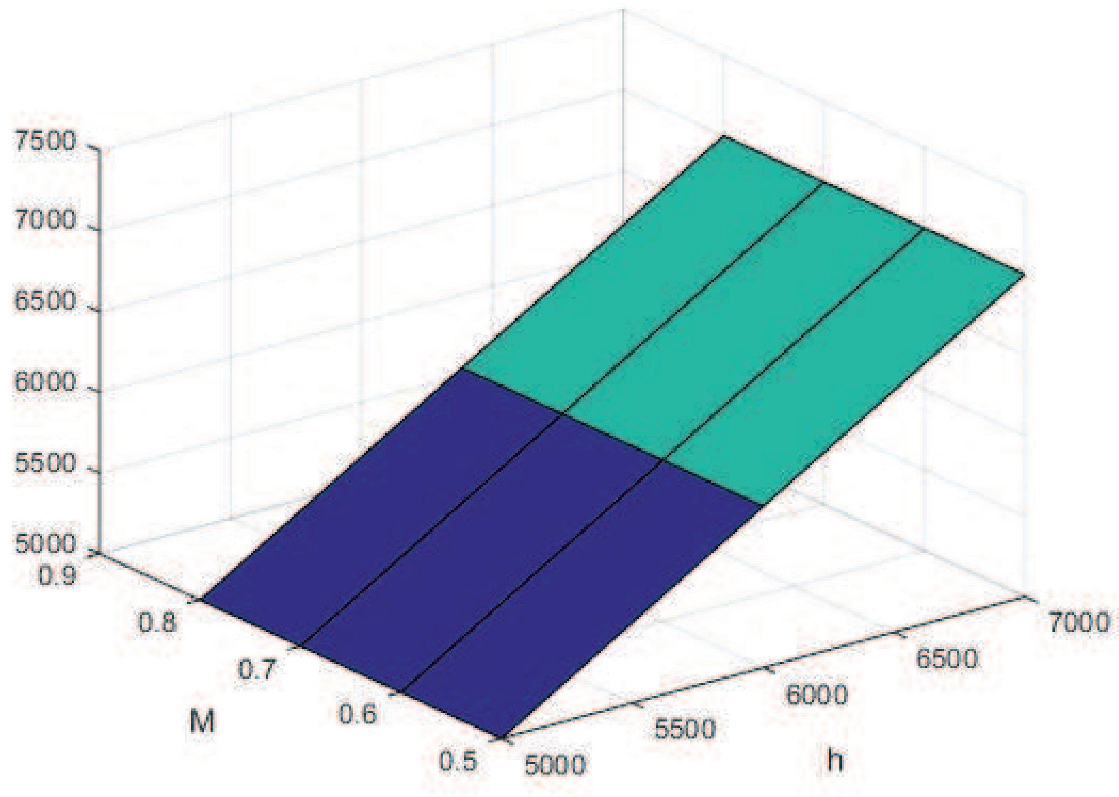
The pointwise gain of `Glpv` is computed using the `norm` function. The results are returned as a `pmat` object, representing a parameter varying matrix:

```
ng = norm(Glpv,inf)
```

```
PMAT with 1 rows and 1 columns.
The PMAT consists of the following blocks:
  M: Gridded real, 4 points in [0.5,0.8], rate bounds [-Inf,Inf].
  h: Gridded real, 3 points in [5e+03,7e+03], rate bounds [-Inf,Inf].
```

`ng` is an array of `double` values arranged on a grid of Mach and altitude values. `ng` contains the infinity norm of `Glpv` computd pointwise at each of the grid points in `Domain`. Lets plot how the value of the infinity norm changes as a function of Mach and altitude:

```
rcplot(ng)
```



Properties of a gridded LPV model

The time-varying parameters that underlie the gridded LPV objects can be accessed through the "Parameter" field.

```
Glpv.Parameter
```

```
ans =
    M: [1x1 pgrid]
    h: [1x1 pgrid]
```

It is possible to change the properties of the time-varying parameters by accessing their properties trough the "Parameter" field. Lets change the rate-bounds of the parameter `M` to be ± 0.3

```
Glpv.Parameter.M.RateBounds = [-0.3 0.3]
```

```
PSS with 1 States, 1 Outputs, 1 Inputs, Continuous System.
The PSS consists of the following blocks:
  M: Gridded real, 4 points in [0.5,0.8], rate bounds [-0.3,0.3].
  h: Gridded real, 3 points in [5e+03,7e+03], rate bounds [-Inf,Inf].
```

Published with MATLAB® R2014b

LPV Analysis

Contents

Problem Statement

The following example illustrates how the LPV approach can help to analyze a subtle difference between two parameter dependent systems.

Consider a first order Linear Time-Invariant (LTI) system G:

$$\begin{aligned}\dot{x} &= -x + u \\ y &= x\end{aligned} \qquad (1)$$

and a time-varying parameter $\delta(t)$, subject to $-1 \leq \delta \leq 1$ and $-\beta \leq \dot{\delta} \leq \beta$, in a parallel interconnection as shown in Figure 1.

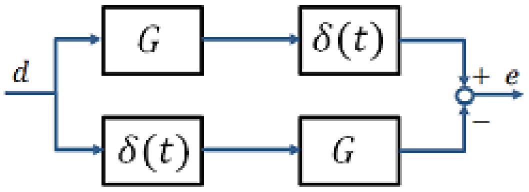


Figure 1: A parallel interconnection of two first order systems.

The parallel signal paths in Figure 1 describe two systems: $G\delta$ and δG. The systems differ only in the position of the δ parameter. In one system δ is applied to the input of G, while it is applied to the output of G in the other. The output of the interconnection, e, is the difference between the outputs of the two systems.

Question: Is there any difference between placing the scalar δ before or after G in the signal path?

Modeling

The following code buils up the system G, the time-varying parameter δ, and the interconnection shown in Figure 1:

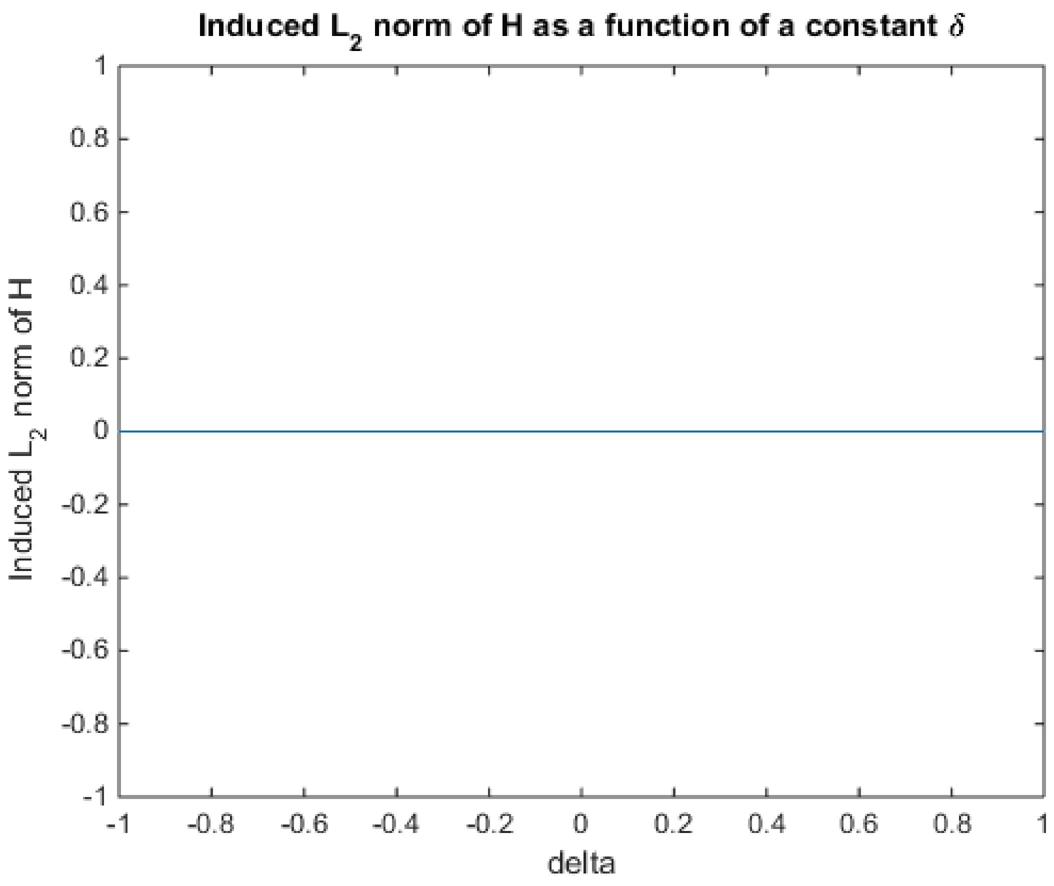
```
% Define the LTI System G
```

```
G = ss(-1,1,1,0);

% Define a scalar parameter 'delta' with values at 20 grid points between -1 and 1
Vals = linspace(-1,1,10);
delta = pgrid('delta',Vals);

% Define a parameter dependent system that describes the interconnection in Figure 1
H = delta*G-G*delta;
```

LTI Analysis

If $\delta(t)$ is constant, then the gain from d to e can be easily computed. The follwing code computes the induced L_2 norm from d to e (i.e. the infinity norm of H), and plots how it changes as a function of the parameter δ.

```
% Compute the induced L2 norm of H
n = norm(H,inf);

% Plot how this norm varies with the value of the parameter delta
lpvplot(n)
title('Induced L_2 norm of H as a function of a constant \delta')
ylabel('Induced L_2 norm of H')
```



LTI Analysis Result

When δ is held constant, the induced L_2 norm of H is zero for all values of δ. Judging from the LTI analysis, the position of the parameter in the signal path has no effect. The LTI analysis is not capable of discriminating between the two systems: $G\delta$ and δG.

LPV Analysis

Now compute the induced L_2 norm of H while taking into account the time-varying nature of δ. The following code computes the induced L_2 norm of H for any trajectory of δ which satisfies: $-1 \leq \delta \leq 1$ and $-\infty \leq \dot{\delta} \leq \infty$.

```
syslpvnorm = lpvnorm(H)
```

```
syslpvnorm =
    1.0024
```

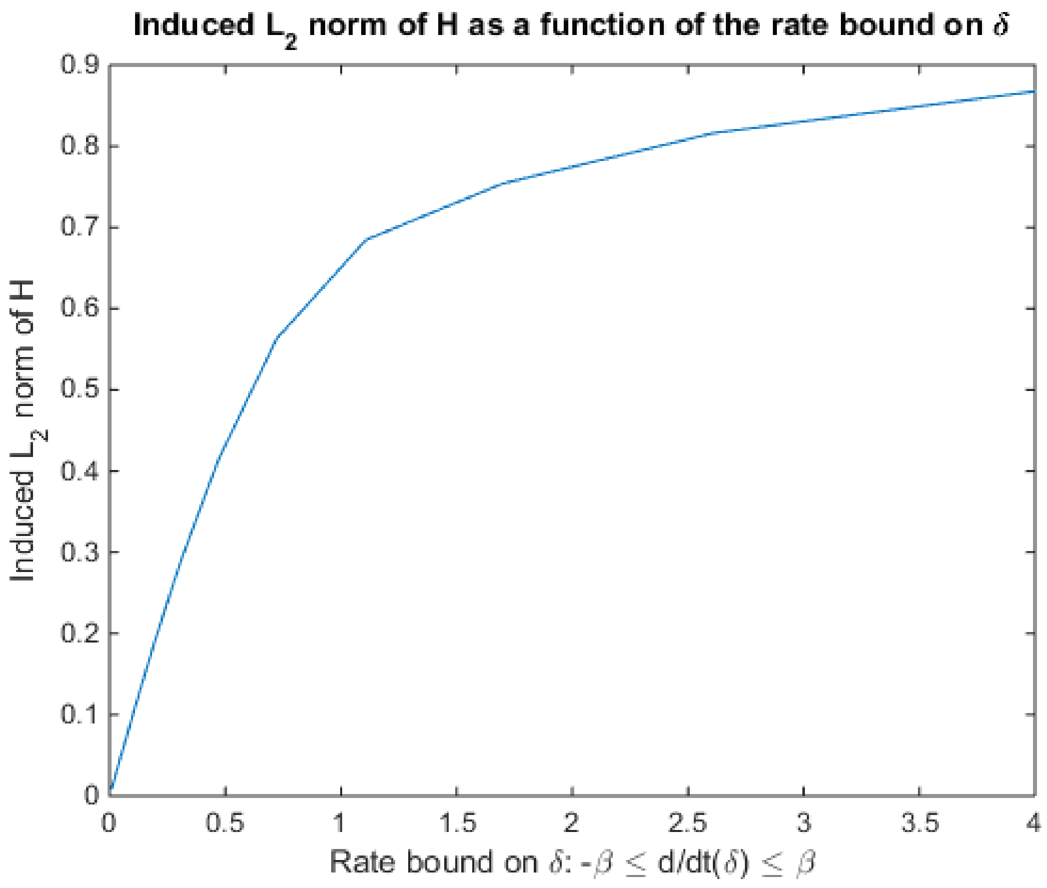
LPV Analysis Results

The LPV analysis yields a non-zero induced L_2 norm for H when δ is allowed to vary with time. This means that there exists some trejectory of δ, subject to $-1 \leq \delta \leq 1$ and $-\infty \leq \dot{\delta} \leq \infty$. such that the two different signal paths through the interconnection in Figure 1 do not yield the same result.

The previous analysis assumed that δ could change arbitrarily fast with time, i.e. $-\infty \leq \dot{\delta} \leq \infty$. Lets repeat the previous analysis with different bounds on the rate of variation of δ.

The following code computes the induced L_2 norm of H when the rate of variation of δ is constrained: $-\beta \leq \dot{\delta} \leq \beta$, with $\beta < \infty$.

```
% Define basis functions for the analysis algorithm.
bf = basis(delta,'delta',1);
Xb = [1;bf;bf^2;bf^3];
% Define a set of rate bounds (beta) to try: 15 values between 0.01 and 4.
rb = logspace(-2,log10(4),15);
for i=1:numel(rb)
  % Set the rate bounds of H to be +/- rb(i)
  H.Parameter.delta.RateBounds = [-rb(i) rb(i)];
  % Compute the induced L2 norm of H, subject to a time-varying delta
  % lying between -1 and 1, with d/dt(delta) between +/- rb(i)
  NormBounds(i) = lpvnorm(H,Xb);
end


plot(rb,NormBounds)
xlabel('Rate bound on \delta: -\beta \leq d/dt(\delta) \leq \beta')
ylabel('Induced L_2 norm of H')
title('Induced L_2 norm of H as a function of the rate bound on \delta')
```



LPV simulation

Lets compare the time-domain response of the two signal paths n H, i.e. the systems $G\delta$ and δG. We will use `lpvstep` to compare the parameter dependent step response:

Start by defining a time vector

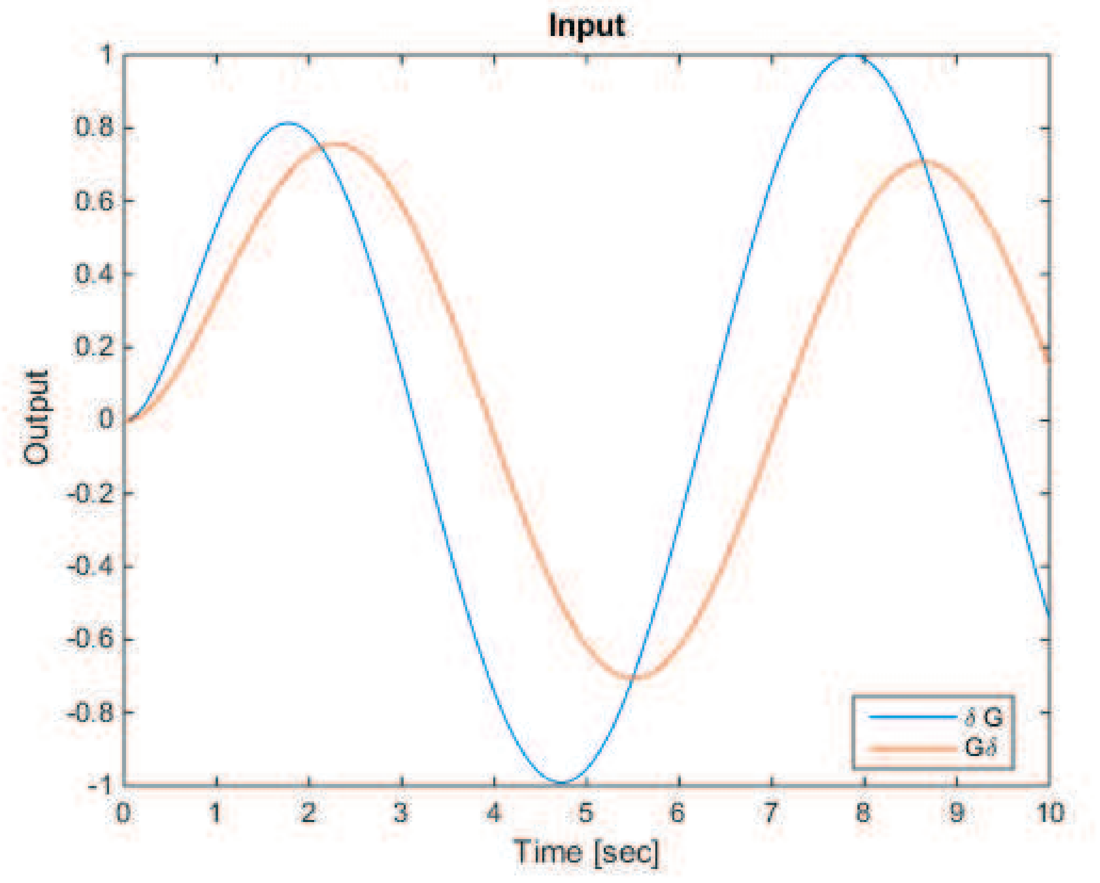
```
t = 0:0.01:10;
```

Define a structure whose fields describe the trajectory of the parameter δ:

```
ptraj.time = t;
ptraj.delta = sin(t);
```

Plot the step response of $G\delta$ and δG when $\delta(t) = sin(t)$. The time domain response highlights the difference between the two signal paths when δ is treated as a time-varying parameter.

```
lpvstep(delta*G,ptraj)
hold on
lpvstep(G*delta,ptraj)
legend('\delta G','G\delta','location','best')
```



Summary

The preceding example demonstrates the power of the LPV approach, and its ability to augment traditional LTI analysis methods with results that take into account the time-varying nature of system components.

If the time varying nature of δ is ignored, an analysis in the LTI framework indicates that the two signal paths in Figure 1 are equivalent. However, if the the time-varying nature of δ is taken into account, an analysis in the LPV framework demonstrates that the position of δ in this interconnection can have a drastic effect on the results.

If δ varies slowly with time the difference between the two signal paths is small, e.g. its on the order of 1% when $|\dot{\delta}| \leq 0.01$. However, when δ changes faster, the difference becomes significant, e.g. the difference between the two singal paths is on the order of 20% when $|\dot{\delta}| = 0.2$, .

Reference

This example was published by Tamas Peni and Peter Seiler in [1]

1. T. Peni, and P. Seiler, "Computation of a lower bound for the induced L2 norm of LPV systems," *accepted to the American Control Conference*, 2015.

Published with MATLAB® R2014b

Synthesis for gridded LPV systems

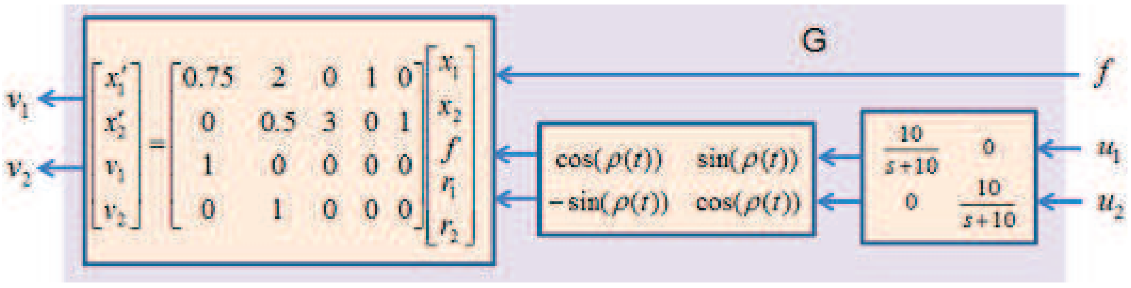
Lets consider a rate-dependent, output-feedback control problem involving stabilization, tracking, disturbance rejection and input penalty. The problem is taken from a

1. G. Meyer, and L. Cicolani, "Application of nonlinear systems inverses to automatic flight control design-system concepts and flight evaluations," *AGARDograph: Theory and Applications 01 Optimal Control in Aerospace Systems*, No. 251,1981.
2. F. Wu, X. H. Yang, A. Packard, and G. Becker, "Induced L2-norm control for LPV systems with bounded parameter variation rates," *Int. J Robust and Nonlinear Control*, Vol. 6, Issue 9-10, pp. 983-998, 1996, doi: 10.1002/(SICI)1099-1239(199611)6:9/10<983::AID-RNC263>3.0.CO;2-C.

Contents

The System

The generalized plant model, G, is created from 3 subsystems as seen in Figure 1, an unstable continuous-time plant, P, a parameter-dependent rotation matrix, R, and two 1st order actuator models.



G
v_1
v_2
f
u_1
u_2
$\begin{bmatrix} x_1' \\ x_2' \\ v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0.75 & 2 & 0 & 1 & 0 \\ 0 & 0.5 & 3 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ f \\ r_1 \\ r_2 \end{bmatrix}$
$\begin{bmatrix} \cos(\rho(t)) & \sin(\rho(t)) \\ -\sin(\rho(t)) & \cos(\rho(t)) \end{bmatrix}$
$\begin{bmatrix} \frac{10}{s+10} & 0 \\ 0 & \frac{10}{s+10} \end{bmatrix}$

Figure 1: The parameter-dependent system G.

G is an LPV system with three inputs (f, u_1, u_2), two outputs (v_1, v_2), and four states. G can be written in the standard form as a parameter varying state-space system:

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \dot{x}_3(t) \\ \dot{x}_4(t) \end{bmatrix} = \begin{bmatrix} 0.75 & 2 & cos(\rho) & sin(\rho) \\ 0 & 0.5 & -sin(\rho) & cos(\rho) \\ 0 & 0 & -10 & 0 \\ 0 & 0 & 0 & -10 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix} \begin{bmatrix} 0 & 0 & 0 \\ 3 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 10 \end{bmatrix} \begin{bmatrix} f \\ u_1 \\ u_2 \end{bmatrix} \qquad (1)$$

$$\begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \qquad (2)$$

The following commands create a grid-based LPV model of the parameter dependent system in Equation (1):

```
% Define the time-varying real parameter.
rho = pgrid('rho',linspace(-pi,pi,7));
rho.RateBounds = [-5 5];

% Define the A, B, C, and D matrices of the LPV system in Equation (1)
pcos = cos(rho);
psin = sin(rho);
A = [0.75 2 pcos psin;0 0.5 -psin pcos;0 0 -10 0; 0 0 0 -10];
B = [0 0 0;3 0 0;0 10 0;0 0 10];
C = [1 0 0 0;0 1 0 0];
D = zeros(2,3);

% Form the grid-based parameter-varying state-space system:
G = pss(A,B,C,D)
```

```
PSS with 4 States, 2 Outputs, 3 Inputs, Continuous System.
The PSS consists of the following blocks:
  rho: Gridded real, 7 points in [-3.14,3.14], rate bounds [-5,5].
```

Problem Formulation

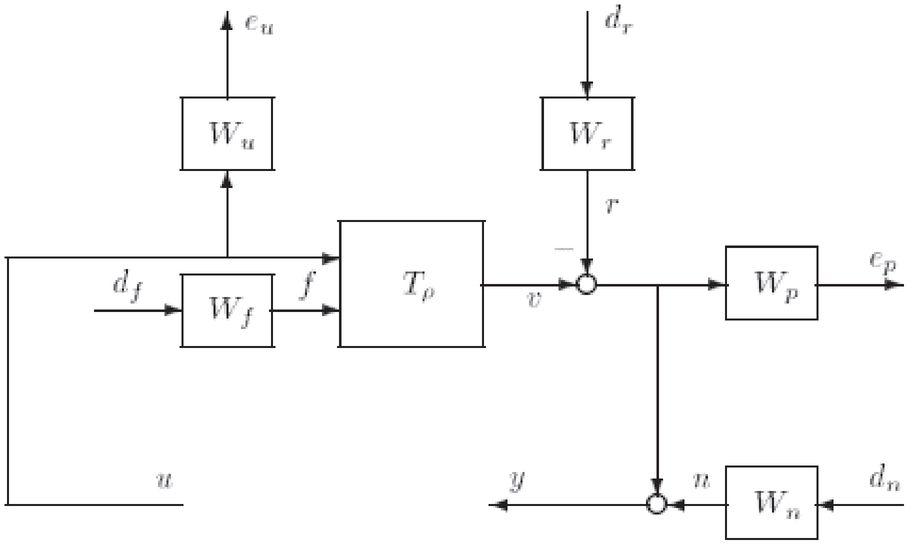


Figure 2: Weighted interconnection for synthesis (from [2])

The control interconnection structure is given in Figure 2, and the weights are defined as follows (from [2])

$$W_p = I_2 \quad W_n = \frac{10(s+10)}{s+1000} \quad W_f = 1 \quad W_u = \frac{1}{280} I_2 \quad W_r = \frac{20}{s+0.2} I_2$$

The weights are generated using the following commands:

```
% Weights
Wp = eye(2);
Wn = ss(10*tf([1 10],[1 1000]))*eye(2);
Wf = 1;
```

```
Wu = (1/280)*eye(2);
Wr = ss(tf(20,[1 0.2]))*eye(2);
```

The control problem interconnection with the weighting function is denoted as H, and is generated using the `sysic` command:

```
% Control Interconnection Structure
systemnames  = 'G Wp Wn Wf Wu Wr';
input_to_G   = '[ Wf; u ]';
input_to_Wp  = '[ G-Wr ]';
input_to_Wn  = '[ dn ]';
input_to_Wf  = '[ df ]';
input_to_Wu  = '[ u ]';
input_to_Wr  = '[ dr ]';
inputvar     = '[ df; dr(2); dn(2); u(2)]';
outputvar    = '[ Wu; Wp; G-Wr+Wn ]';
H = sysic
```

```
PSS with 8 States, 6 Outputs, 7 Inputs, Continuous System.
The PSS consists of the following blocks:
  rho: Gridded real, 7 points in [-3.14,3.14], rate bounds [-5,5].
```

Synthesis

The original system G depends on the time-varying parameter ρ, and the weighted interconnection H inherits this parameter dependence. Next we will synthesize a LPV controller for H. The LPV controller will be optimized for the prescribed parameter trajectories, i.e. $-\pi \leq \rho\pi$ and $-5 \leq \dot{rho} \leq 5$. The resulting LPV controller will itself be parameter dependent and will depend on the parameter ρ and its derivative $\dot{rho}$.

The following code finds a controller `Klpv` which minimizes the induced L_2 norm of lft(H,Klpv) when the rate of variation of ρ is constrained: $-5 \leq \dot{\rho} \leq 5$.

```
% Basis function,
b1 = basis(1,0);
bcos = basis(pcos,'rho',-psin);
bsin = basis(psin,'rho',pcos);
Xb = [b1;bcos;bsin];
Yb = Xb;

% LPV Rate-Bounded Control Design
opt = lpvsynOptions('BackOffFactor',1.02);
[Klpv,normlpv] = lpvsyn(H,2,2,Xb,Yb,opt);
```

The LPV controller is a `pss` object `klpv`

```
Klpv
```

```
PSS with 8 States, 2 Outputs, 2 Inputs, Continuous System.
```

```
The PSS consists of the following blocks:
  rho: Gridded real, 7 points in [-3.14,3.14], rate bounds [-5,5].
  rhoDot: Gridded real, 2 points in [-5,5], rate bounds [-Inf,Inf].
```

If we close the loop around the weighted interconnection, and form `lft(H,Klpv)`, the controller achieves an induced L_2 norm which is bounded from above `normlpv`:

```
normlpv
```

```
normlpv =
    0.9250
```

Pointwise analysis in the LTI framework

Lets apply `Klpv` to the original system `G`, and compare the open-loop vs closed loop response for an f input:

Start by forming the closed loop system:

```
CL = feedback(G,Klpv,[2 3],[1 2],+1)
```

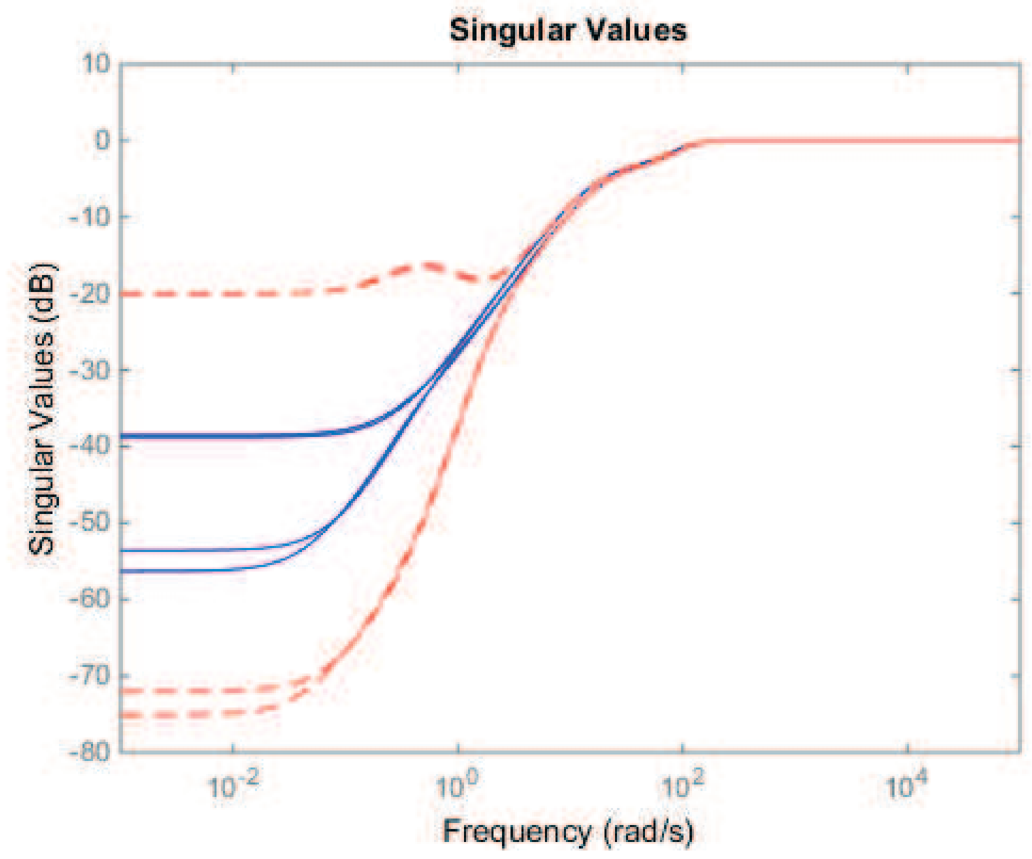
```
PSS with 12 States, 2 Outputs, 3 Inputs, Continuous System.
The PSS consists of the following blocks:
  rho: Gridded real, 7 points in [-3.14,3.14], rate bounds [-5,5].
  rhoDot: Gridded real, 2 points in [-5,5], rate bounds [-Inf,Inf].
```

Set the input and output names for CL

```
CL.InputName = {'f','u_1','u_2'};
CL.OutputName = {'v_1','v_2'};
```

Plot the output and input sensitivity functions at each point in the domain

```
SF = loopsens(G(:,2:3),Klpv);
sigma(SF.So,'b',SF.Si,'r--')
```



Compute the input disk margins of the closed-loop system at each point in the domain:

```
DMI = loopmargin(G(:,2:3),Klpv,'di');
```

The smallest input disk margin in the domain has a gain margin of:

```
lpvmin(DMI.GainMargin(2))
```

```
PMAT with 1 rows and 1 columns.

ans =
   23.4339
```

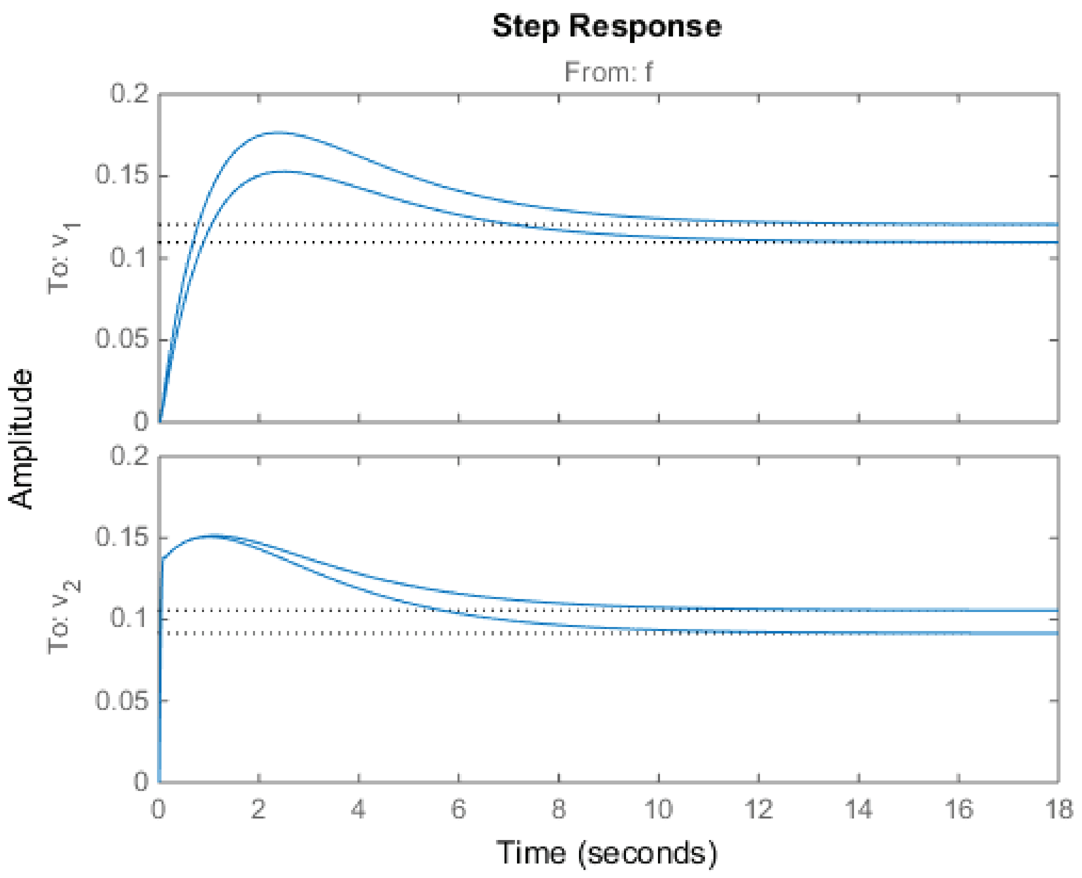
The smallest input disk margin in the domain has a phase margin of:

```
lpvmin(DMI.PhaseMargin(2))
```

```
PMAT with 1 rows and 1 columns.

ans =
   85.1130
```

Simulate the step response of the closed-loop system to a unit f input, at each point in the domain:

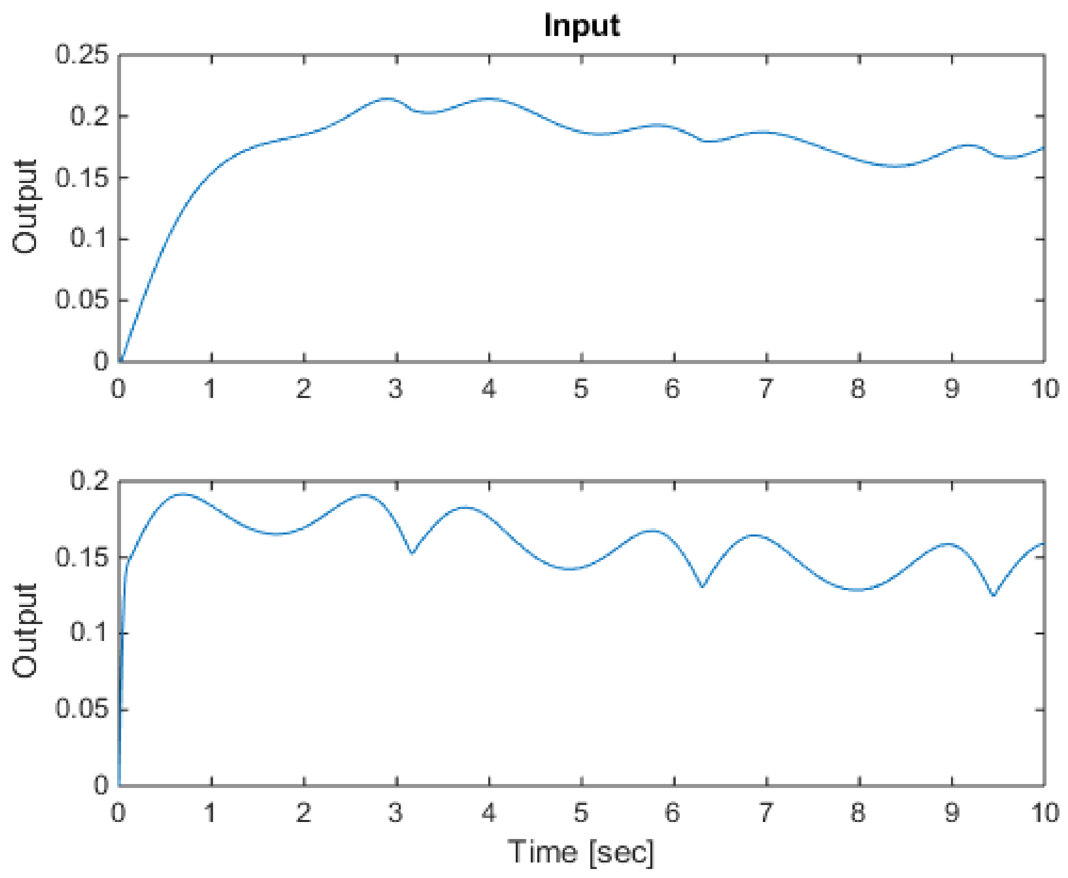
```
step(CL(:,1))
```



LPV Simulation

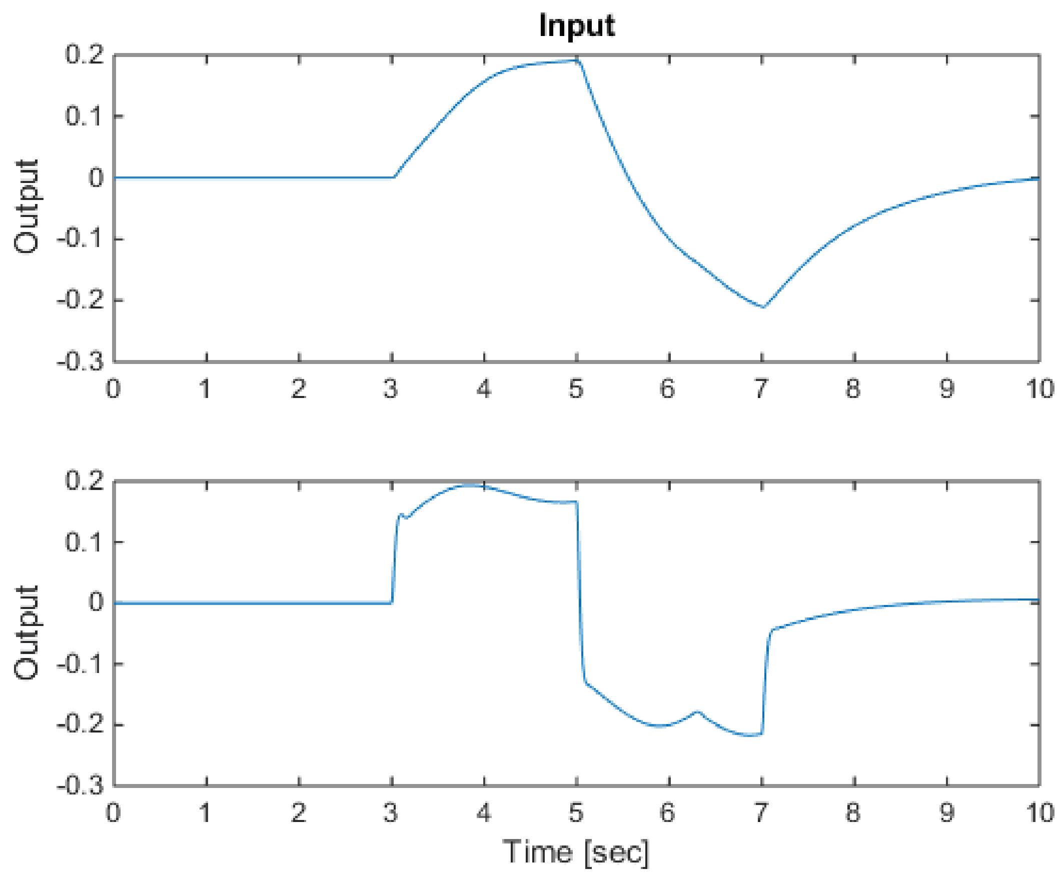
Next, we will compute the time-domain response of the parameter dependent closed loop system as the parameter ρ follows a particular trajectory: $\rho = sin(t)$. This simulation is different from the previous simulation, generated by the command `step(CL(:,1))`, which simlated the step LTI response of the closed-loop system at fixed ρ values. To perform a parameter dependent simulation of the step response we use the `lpvstep` command:

```
% Define a time vector for the simulation
t = 0:0.01:10;

% Define a structure whose fields describe the trajectory of the parameter
ptraj.time = t;
ptraj.rho = sin(t);
ptraj.rhoDot = cos(t);

% Plot the parameter dependent step response for $\rho(t) = sin(t)$.
lpvstep(CL(:,1),ptraj)
```



We can also look at the time-domain response for a custom input: Plot the parameter dependent response for a unit doublet command when $\rho(t) = sin(t)$.

```
u = [zeros(size(0:0.01:3)) ones(size(3.01:0.01:5)),...
     -ones(size(5.01:0.01:7)) zeros(size(7.01:0.01:10))]';
 lpvlsim(CL(:,1),ptraj,u,t);
```



Published with MATLAB® R2014b

Modeling LFT LPV Systems

Contents

Introduction

A key component of the LFT-based LPVTools infrastructure is the core LFT data structure object, referred to as a `tvreal` (denoting a time-varying parameter). The `tvreal` object is used to create a time-varying, real valued scalar object. The `tvreal` has a range, denoting the maximum and minimum value that the time-varying scalar can assume, and a rate-bound denoting the maximum and minimum rate of change of the time-varying scalar. The `tvreal` is used to model individual time-varying parameters, and construct parameter dependent LFT matrices and systems. LPVTools represents LFT-based parameter varying matrices and state-space systems by `plftmat` and `plftss` data objects, respectively. The `plftmat`, and `plftss` objects are constructed using `tvreal` elements, using a syntax that is a direct parallel to the `ureal` syntax that is used to define `umat` and `uss` objects in the Robust Control Toolbox.

Example of LFT construction

We will design a LFT-based LPV controller for the system $G(\rho)$:

$$\begin{aligned} \dot{x} &= -\rho x + \rho u \\ y &= x \end{aligned} \qquad (1)$$

LFT-based LPV models are restricted to systems with rational parameter dependence, which the system in Equation (1) satisfies. Hence we can construct $G(\rho)$ as a LFT-based LPV system using a `tvreal`. The first argument of `tvreal` is the name of the parameter, the second argument is the range of the parameter, and the third argment is the range for the parameter's rate of variation. Lets model $G(\rho)$ for $1 \leq \rho \leq 10$ and $-1 \leq \dot{\rho} \leq 1$.

```
% Define a time-varying real parameter.
rho = tvreal('rho',[1 10],[-1 1]);

% Construct a parameter varying LFT state-space systems:
P = ss(-rho,rho,1,0)
```

```
Continuous-time PLFTSS with 1 outputs, 1 inputs, 1 states.
The model consists of the following blocks:
  rho: Time-varying real, range = [1,10], rate bounds = [-1,1], 1 occurrences
```

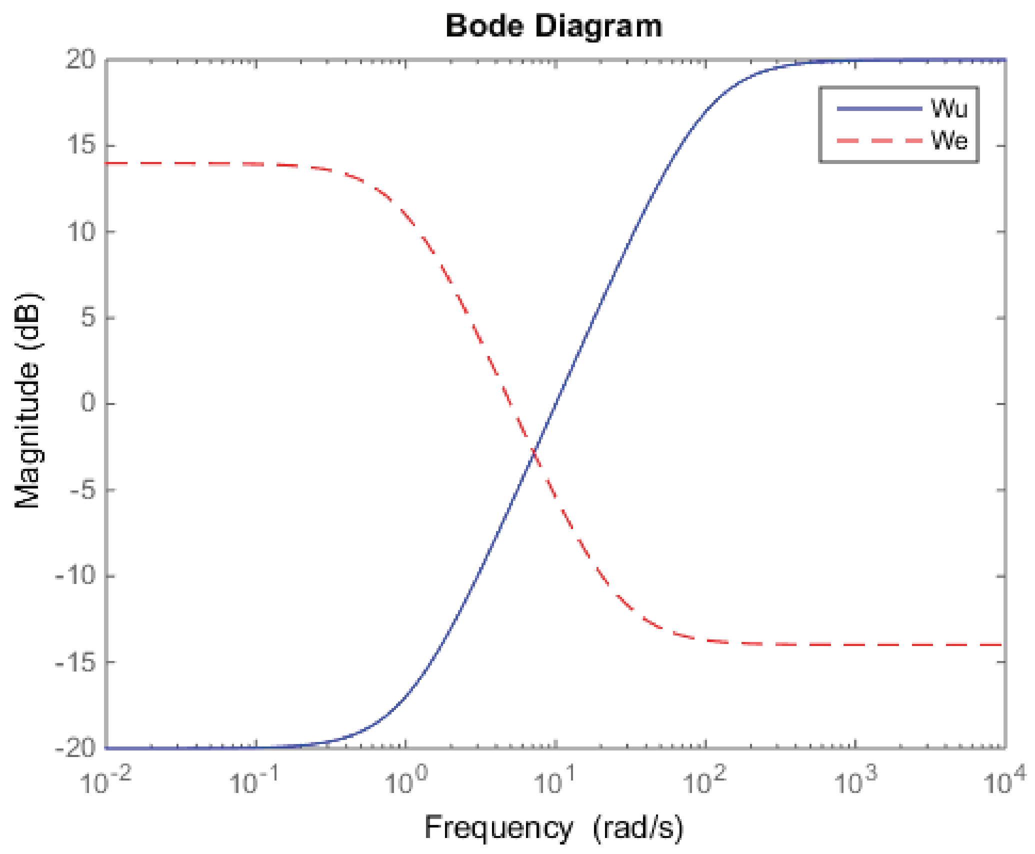
LFT-based LPV Synthesis

We will use the command `lpvsyn` to synthesize a LFT-based LPV controller for this system. `lpvsyn` requires that the closed-loop performance objectives be characterized in terms of a weighted interconnection (analogous to H_∞ performance problems) so we define one using a set of dynamic weights and the command `sysic`.

The exogenous signal passing through the weight `Wd` is added to the control signal from the controller. The weight `Wu` acts on the disturbed control signal going from the control to the plant input. The weight `We` acts on the error between the reference signal and the plant output.

The `Wu` weight expresses the requirement that the control signal should be less than 10 up to 1 rad/s, roll off and cross over at 10 rad/s, and be less than 0.1 above 100 rad/s. The `We` weight expresses the requirement that the tracking error be less than 0.2 at frequencies below 1 rad/s, it can then increase, and must be less than 1 at 5 rad/s, and less than 5 at frequencies above 25 rad/s. The `Wd` expresses the fact that the disturbance on the control signal will have magnitude no larger than 0.1.

```
% Define and plot weights for synthesis problem
Wu = tf([10 10],[1 100]);
We = tf([1 25],[5 5]);
Wd = ss(0.1);

bodemag(Wu,'b',We,'r--')
legend('Wu','We')
```



Define a weighted interconnection for the synthesis problem

```
systemnames = 'P Wu We Wd';
inputvar = '[r; d; u]';
outputvar = '[We; Wu; r-P]';
```

```
input_to_We = '[r-P]';
input_to_Wu = '[u+Wd]';
input_to_Wd = '[d]';
input_to_P = '[Wu+u]';
Pg = sysic
```

```
Continuous-time PLFTSS with 3 outputs, 3 inputs, 3 states.
The model consists of the following blocks:
  rho: Time-varying real, range = [1,10], rate bounds = [-1,1], 1 occurrences
```

Next we will synthesize a LFT-based LPV controller that minimizes the induced L_2 norm of the weighted interconnection Pg. The first argument of `lpvsyn` is the weighted interconnection. The second argument is the number of measurments available to the controller. The third argument is the number of control inputs available to the controller:

```
% Perform LPV design with LFT approach
nmeas = 1;
ncon = 1;
[KbLFT,GAMbLFT,INFObLFT] = lpvsyn(Pg,nmeas,ncon);
```

The LFT-based controller `KbLFT` is guarenteed to acheive a induced L_2 norm of `GAMbLFT`:

```
GAMbLFT
```

```
GAMbLFT =
    2.1480
```

There are two important points to note. First, the algorithm implemented in `lpvsyn` for LFT-based LPV systems (see [1,2,3,4] for details), does not take into account the bounds on the parameter rate-of-variation. Hence, `GAMbLFT` is a bound on the induced L_2 norm when there are no limits to how fast the parameter can change with time. Second, `GAMbLFT` is only an upper bound on the induced L_2 norm achived by `KbLFT` Hence, for input signals that have induced L_2 norms bounded by 1, the induced L_2 norm is guarenteed to be no larger than `GAMbLFT`. for any parameter trajectory such that: $1 \leq \rho(t) \leq 10$

LPV Analysis Incorporating Rate-Bounds

The system does have rate-bounds on the parameter $-1 \leq \dot{\rho}(t) \leq 1$. We will now compute a induced L_2 norm achived by `KbLFT` when these bounds are taken into account. To do this we use the function `lpvnorm`, which will detect the rate-bounds in the system and incorporate them into the analysis:

```
% Form weighted interconnection with controller in the loop:
IC = lft(Pg,KbLFT);

% Compute the induced $L_2$ norm achived by KbLFT
Gamma = lpvnorm(IC)
```

```
Gamma =
       1.7317
```

Gamma is the upper bound on the induced L_2 norm achived by KbLFT. Hence, for input signals that have induced L_2 norms bounded by 1, KbLFT is guarenteed to achieve an induced L_2 norm that is no greater than Gamma for all permissible parameter trajectories (in this case: $1 \leq \rho(t) \leq 10$ and $-1 \leq \dot{\rho}(t) \leq 1$). We note that the induced L_2 norm achived by KbLFT is significantly lower when the rate-bounds on the parameter are taken into account.

Pointwise LTI Analysis of LFT-Based LPV Systems

A LFT-based LPV system can be transformed into a Linear Time-Invariant (LTI) system by holding its parameters at a constant value. Hence, it is possible to apply standard LTI analysis techniques to evaluate the pointwise performance of the LFT-based LPV controller. We will evaluate its performance on a grid of 5 points: $\rho \in [1, 2, 3, 4, 5]$. The syntax to perform pointwise LTI analysis requires the user to pass in a rgrid object that specifies the grid of parameter values that the LFT-based LPV system should be evaluated at. Hence, we define the rgrid object Domain to specify the desired grid points:

```
% Define the grid of parameter values:
Domain = rgrid('rho',1:5,[-1 1])
```

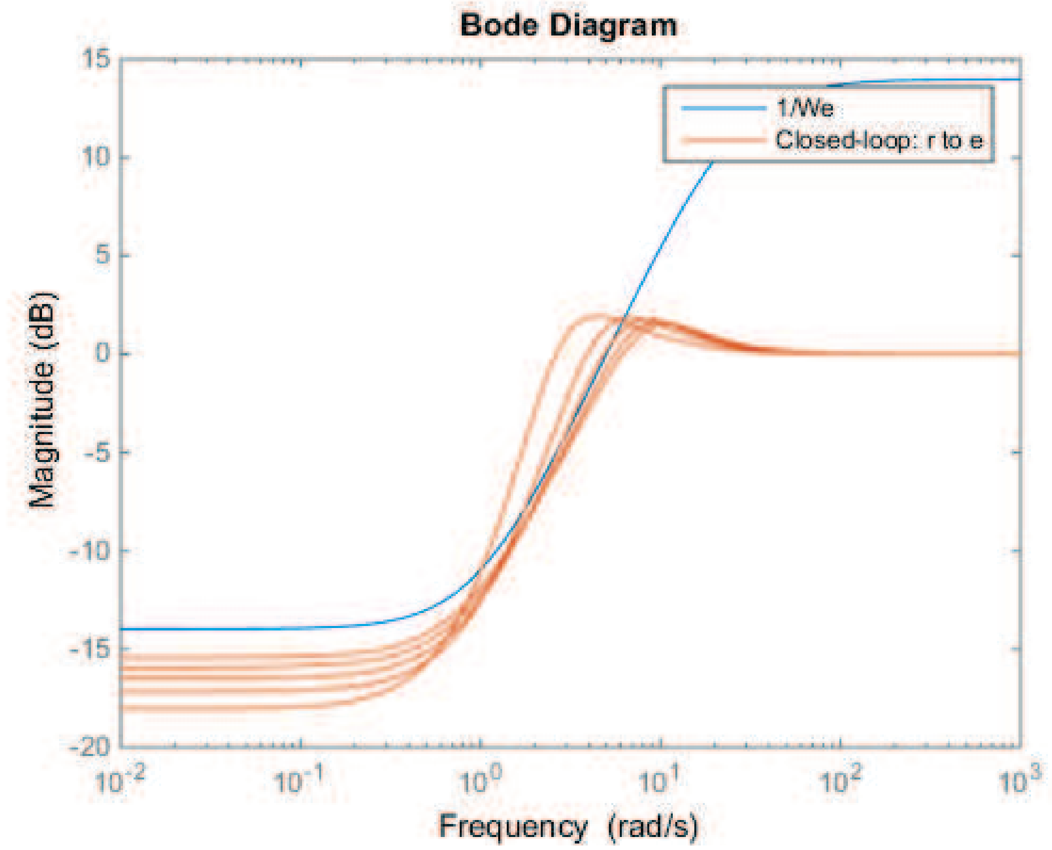
```
RGRID with the following parameters:
  rho: Gridded real, 5 points in [1,5], rate bounds [-1,1].
```

Multiple LTI analysis and simulation functions are overloaded for plftss objects. Lets use them to study the frequency response of the closed-loop system. Start by forming the closed-loop system consisting of the controller and plant, without any of the weights:

```
% Form closed-loop system without weights:
systemnames = 'P KbLFT';
inputvar = '[r; d; u]';
outputvar = '[r-P; KbLFT;P]';
input_to_KbLFT = '[r-P]';
input_to_P = '[KbLFT+d]';
CL = sysic;
CL.InputName ={'r','d','u'};
CL.OutputName = {'e','u','y'};
```

Now plot a Bode plot of the performance requirement, expressed by We and compare it against the closed-loop response from the referene to the error.

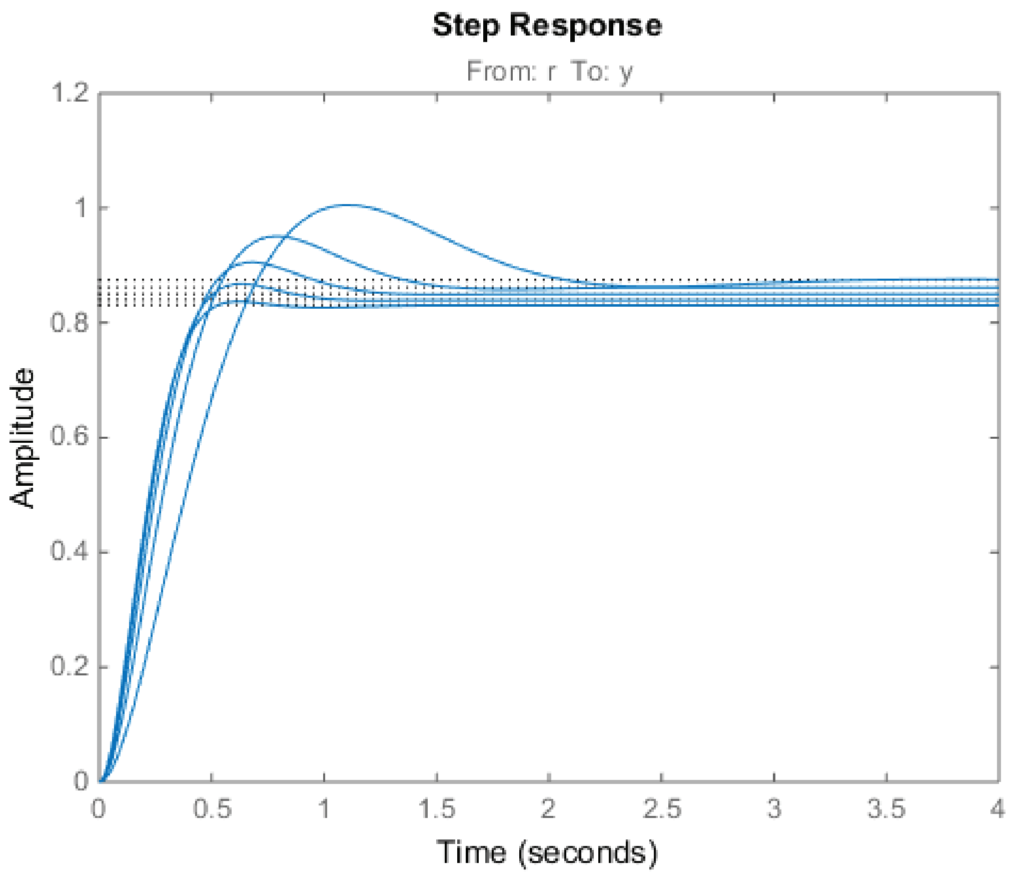
```
bodemag(1/We)
hold on
bodemag(CL('e','r'),Domain)
legend('1/We','Closed-loop: r to e')
hold off
```



The performance requirement is not satisfied at $\rho = 1$, $\rho = 2$, and $\rho = 3$ in the frequency band 1-6 rad/s.

Lets look at the LTI step response from reference to output:

```
step(CL('y','r'),Domain)
```



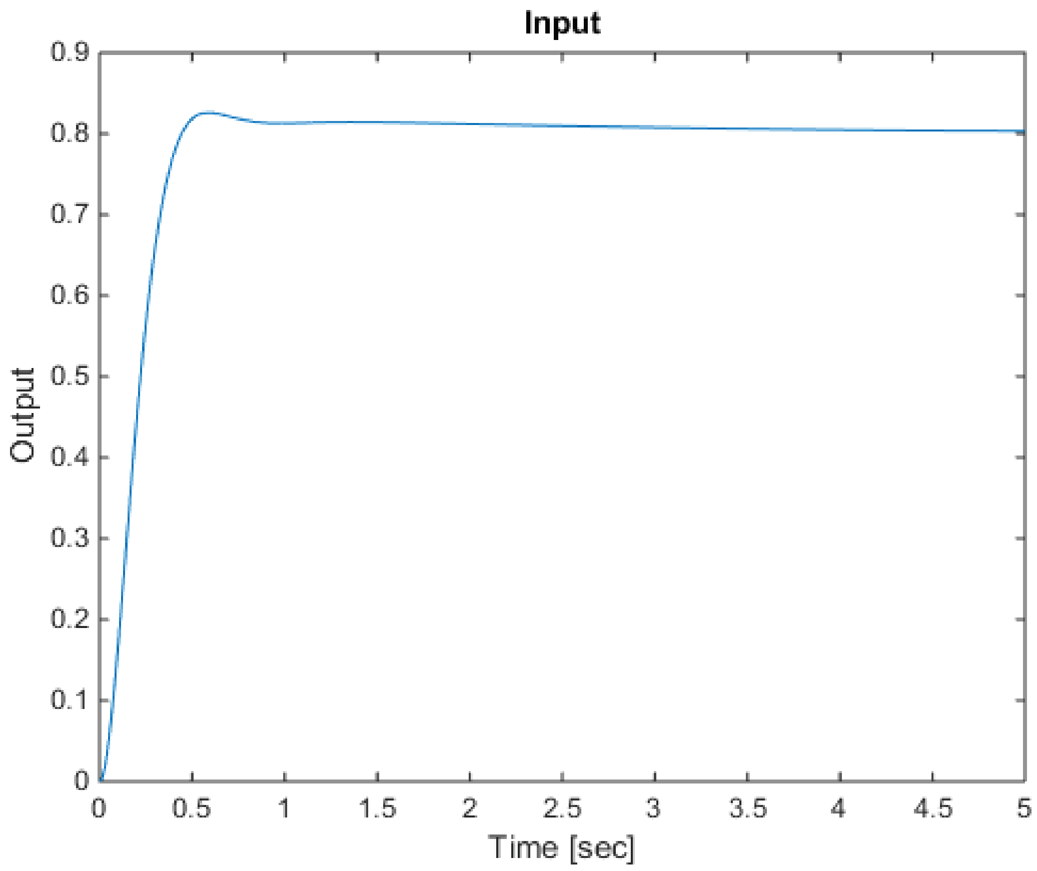
LPV Simulation

The step response is well behaved and has less than 20% steady state tracking error, which satisfies the design specification expressed by `We`.

The LPV system is time-varying, and LTI analysis does not capture the time-varying nature of the model. We can evaluate the performance of the LFT controller as the parameter varies with time by using time-domain simulation for a particular parameter trajectory. LPVTools provides a set of functions for LPV simulation: `lpvlsim`, `lpvstep`, `lpvinitial`, and `lpvimpulse`.

Lets look at the step response of the closed-loop system as the parameter traverses the trajectory: $\rho(t) = 4\sin(0.25t) + 5$

```
% Define the trajectories of the parameters:
t =0:0.01:5;
ptraj.time = t;
ptraj.rho = 4*sin(0.25*t)+5;

% Perform LPV simulation:
lpvstep(CL('y','r'),ptraj);
```



The tracking response is excellent for this particular parameter trajectory, and settles down to a steady state error of approximatly 20%.

References

1. A. Packard, "Gain Scheduling via Linear Fractional Transformations," *System and Control Letters*, 1994.
2. P. Apkarian and P. Gahinet, "A Convex Characterization of Gain-Scheduled H-Infinity Controllers," *IEEE Transactions on Automatic Control,* Vol. 40, No. 5 pp. 853-864, 1995.
3. P. Apkarian and P. Gahinet, "Erratum to: A Convex Characterization of Gain-Scheduled H-Infinity Controllers," *IEEE Transactions on Automatic Control,* 1995.
4. P. Gahinet, "Explicit Controller Formulas for LMI-based H-Infinity Synthesis," *Automatica*, Vol. 32, No. 7, pp. 1007-1014, 1996.

Published with MATLAB® R2014b

Converting between LFT- and Grid-based LPV Systems

LPVTools provides tools to convert a LFT-based LPV system into a grid-based LPV system, and vice versa. In this example we will showcase the functionality of these tools. We will utilize them to take a grid-based LPV system and transform it into LFT-based form to do control design. Finally, the LFT-based controller will be transformed into grid-based form for evaluation.

Contents

Define a Grid-Based LPV Model

We have a LPV model $G(\rho)$:

$$\begin{aligned} \dot{x}_G &= -\rho_1 x_G + \rho_2 u_G \\ y &= \rho_1 * \rho_2 x_G \end{aligned} \qquad (1)$$

which is modeled as a grid-based LPV model using `pss` and `pgrid` objects:

```
% Define a time-varying real parameter.
rho1 = pgrid('rho1',1:10,[-1 1]);
rho2 = pgrid('rho2',1:3,[-10 10]);

% Construct a grid-based LPV system:
P = ss(-rho1,rho2,rho1*rho2,0);
```

Converting a Grid-based LPV Model into a LFT-based LPV Model

Lets transform P from a grid-based LPV system into a LFT-based system. This is accomplished using the function `grid2lft`, which transforms a grid-based LPV model into a LFT-based LPV model by approximating the parameter dependence of the underlying data and expressing it as a rational function of the parameter, which can then be rewritten in LFT form.

A grid-based LPV system consists of an array of state-space models, arranged on a grid of parameter values. The current implementation of `grid2lft` takes each element of these parameter dependent state-space matrices and finds a polynomial function of the parameter, which captures how that element changes as a function of the parameter. Once the array of state-space models has been replaced by a single state-space model whose matrix elements are polynomial functions of the parameters, it can be rewritten as a LFT.

The user can specify the desired form the the polynomial function used to fit the matrix elements. In this example we will use a polynomial of the form: (1,x,y,x^2,x*y,y^2) to fit the grid-based data.

We will use `grid2lft` to transform P into a LFT-based model `Plft`. The first argument to `grid2lft` is the grid-based model that will be approximated as a LFT. The second argument is the desired order of the polynomial used for the fit, in this case (1,x,y,x^2,x*y,y^2) corresponds to a second order polynomial, so we put in the number 2:

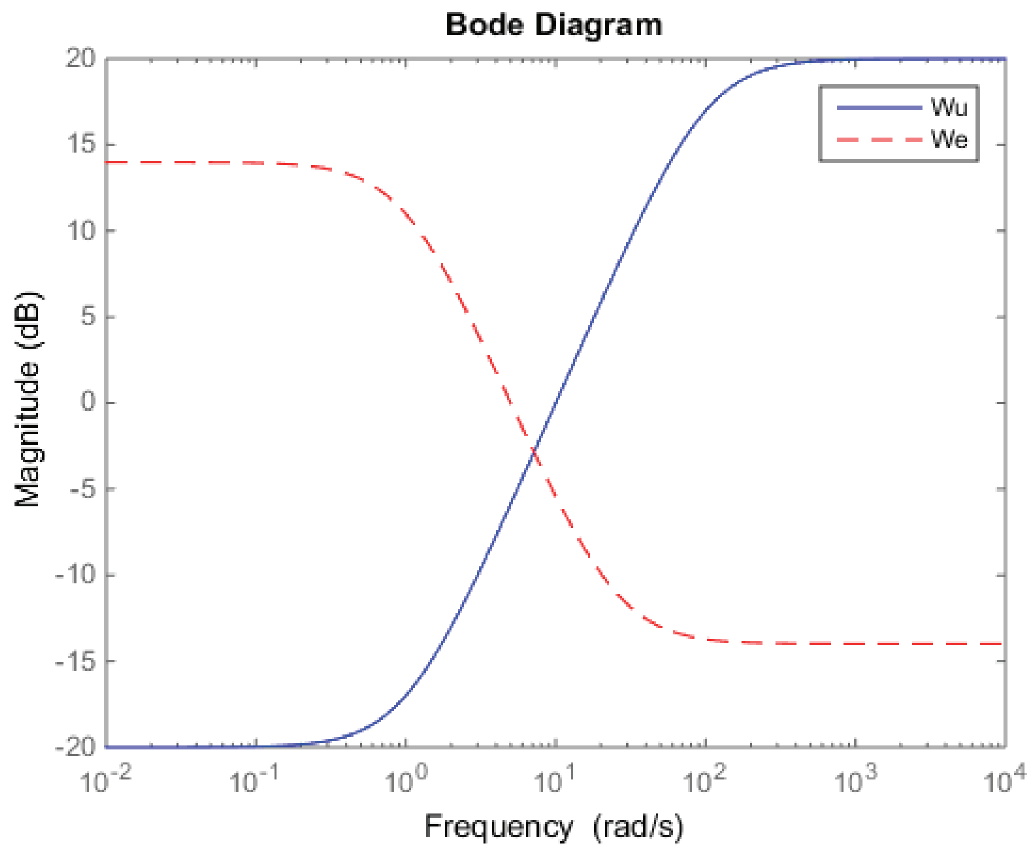
```
% Transfrom P into a LFT model:
Plft = grid2lft(P,2)
```

```
Continuous-time PLFTSS with 1 outputs, 1 inputs, 1 states.
The model consists of the following blocks:
  rho1: Time-varying real, range = [1,10], rate bounds = [-1,1], 1 occurrences
  rho2: Time-varying real, range = [1,3], rate bounds = [-10,10], 2 occurrences
```

LPV Design problem

We can define the control design problem as follows:

```
% Define and plot weights for synthesis problem
Wu = tf([10 10],[1 100]);
We = tf([1 25],[5 5]);
Wd = ss(0.1);

bodemag(Wu,'b',We,'r--')
legend('Wu','We')
```



Define a weighted interconnection for the synthesis problem

```
systemnames = 'Plft Wu We Wd';
inputvar = '[r; d; u]';
```

```
outputvar = '[We; Wu; r-Plft]';
input_to_We = '[r-Plft]';
input_to_Wu = '[u+Wd]';
input_to_Wd = '[d]';
input_to_Plft = '[Wu+u]';
Gweights = sysic
```

```
Continuous-time PLFTSS with 3 outputs, 3 inputs, 3 states.
The model consists of the following blocks:
  rho1: Time-varying real, range = [1,10], rate bounds = [-1,1], 1 occurrences
  rho2: Time-varying real, range = [1,3], rate bounds = [-10,10], 2 occurrences
```

LFT-based LPV Synthesis

We will use `lpvsyn` to synthesize the LFT controller:

```
% Perform LPV design with LFT approach
nmeas = 1;
ncont = 1;
[Klft,GammaLFT] = lpvsyn(Gweights,nmeas,ncont);
```

The LFT-based controller `Klft` is guarenteed to acheive a induced L_2 norm of `GammaLFT`:

```
GammaLFT
```

```
GammaLFT =
    4.8339
```

Convert LFT-Based System to Grid-Based System

We can transform the LFT controller into a grid-based LPV controller. This is accomplished using the `lft2grid` function. The process of transforming a LFT based system into a grid-based LPV system is simple: First, pick a desired grid of parameter values for the resulting grid-based system. Second, evaluate the LFT-based system at each grid point by replacing the time-varying parameter in the LFT, with a parameter values at each grid point. The resulting array of state-space models and assocaited grid of parameter values constitutes a grid-based LPV model approximation of the LFT-based model. Lets transform `Klft` into a grid-based LPV system.

```
% We will use the parameter grid from the original system P
Domain = P.Domain;
```

```
% Transform Klft into a grid-based LPV system:
Kg = lft2grid(Klft,Domain)
```

```
PSS with 3 States, 1 Outputs, 1 Inputs, Continuous System.
The PSS consists of the following blocks:
  rho1: Gridded real, 10 points in [1,10], rate bounds [-1,1].
```

```
    rho2: Gridded real, 3 points in [1,3], rate bounds [-10,10].
```

We transform the weighted interconnection `Gweights` into grid-based model, and compute the induced L_2 norm achieved by the grid-based version of the controller:

```
% Transform the  weighted interconnection into a grid-based system
GweightsGRID = lft2grid(Gweights,Domain)
```

```
PSS with 3 States, 3 Outputs, 3 Inputs, Continuous System.
The PSS consists of the following blocks:
  rho1: Gridded real, 10 points in [1,10], rate bounds [-1,1].
  rho2: Gridded real, 3 points in [1,3], rate bounds [-10,10].
```

```
% Closed the loop around the controller and the weighted interconnection:
WeightedCL = lft(GweightsGRID,Kg)
```

```
PSS with 6 States, 2 Outputs, 2 Inputs, Continuous System.
The PSS consists of the following blocks:
  rho1: Gridded real, 10 points in [1,10], rate bounds [-1,1].
  rho2: Gridded real, 3 points in [1,3], rate bounds [-10,10].
```

```
% Compute the induced $L_2$ norm:
GammaGrid = lpvnorm(WeightedCL)
```

```
GammaGrid =
    2.4102
```

The induced L_2 norm computed for the grid-based version of the controller is substantially lower than the induced L_2 norm computed for the LFT version. There are a few issues that can explain this. First, the computed induced L_2 norm is in both cases only an upper bound, hence the results are not inconsistent. Second, the LFT-based LPV model is a smooth function of the parameter, and includes every intermediary parameter value between the grid points in `Domain` (the grid of parameter values which underlies the grid-based LPV model). The dynamics at these intermediary points was approximated from the existing grid-based model. Hence, if this approximation is inaccurate, the LFT-based analysis will be taking into account dynamics that are not really there.

Published with MATLAB® R2014b

Modeling LPV systems

A primer on modeling LPV systems.

Contents

LPV Modeling Commands

PGRID	Gridded real parameter.
RGRID	Rectangular grid of parameter values.
PMAT	Parameter-varying matrix.
PSS	Parameter-varying state-space system.
PFRD	Parameter-varying frequency response data model.
UPMAT	Parameter-varying uncertain matrix.
UPSS	Parameter-varying uncertain state-space system.
UPFRD	Parameter-varying uncertain frequency response data model.
BASIS	Parameter-varying basis function for analysis and synthesis.
PSTRUCT	Parameter-varying structure.

DOMUNION	Map LPV objects onto a common domain.
LPVSPLIT	Extract LPV model data from a subset of its parameter domain.
LPVINTERP	Interpolate a grid-based LPV model.
LPVSUBS	Substitute values of parameters.
LPVELIMIV	Eliminate parameters which only have a single grid point.
LPVSAMPLE	Sample a grid-based LPV object.
LPVBALANCE	Diagonal scaling for LPV models.

Examples and How To

- Tutorial: Constructing grid-based LPV models
- Tutorial: Constructing LFT-based LPV models
- Tutorial: Conversion between LFT and LPV models
- Tutorial: Creating a grid-based LPV model from analytical linearization.
- Tutorial: Creating a grid-based LPV model from a nonlinear model.

Concepts

- Permissible Parameter Trajectories
- Grid-based LPV model.
- LFT-based LPV model.
- Quasi-LPV Models
- State Consistency

Published with MATLAB® R2014b

Deriving LPV models from Analytical Jacobian linearization

Contents

Introduction

Consider a nonlinear system:

$$\dot{x}(t) = f(x(t), u(t), \rho(t)) \qquad (1)$$

$$y(t) = h(x(t), u(t), \rho(t)) \qquad (2)$$

Where $x \in \mathcal{R}^{n_x}$, $y \in \mathcal{R}^{n_y}$, $u \in \mathcal{R}^{n_u}$, and $\rho \in \mathcal{R}^{n_{rho}}$.

Assume that $\rho(t) = \rho_0$ and $u(t) = \bar{u}(\rho_0)$ are constant $\forall t \geq 0$. Then the solution of the nonlinear system is any $x(t) = \bar{x}(\rho_0)$ and $y(t) = \bar{y}(\rho_0)$, such that if $x(0) = \bar{x}(\rho_0)$, then $\forall t \geq 0$:

$$\dot{x}(t) = 0 = f(\bar{x}(\rho_0), \bar{u}(\rho_0), \rho_0) \qquad (3)$$

$$y(t) = \bar{y}(\rho_0) = h(\bar{x}(\rho_0), \bar{u}(\rho_0), \rho_0) \qquad (4)$$

When $\rho(t)$ is a function of time, then the equilibrium $(\bar{x}(\rho(t)), \bar{u}(\rho(t)), \bar{y}(\rho(t)))$ is, in general, not a solution of the nonlinear system:

$$\frac{d}{dt}\bar{x}(\rho(t)) \neq 0 = f(\bar{x}(\rho(t)), \bar{u}(\rho(t)), \rho(t)) \qquad (5)$$

We can linearize around $(\bar{x}(\rho(t)), \bar{u}(\rho(t)), \bar{y}(\rho(t)))$ even though it is not, in general, a solution of the nonlinear system. Lets define perturbed quantities:

$$\delta_x(t) = x(t) - \bar{x}(\rho(t)) \qquad (6)$$

$$\delta_u(t) = u(t) - \bar{u}(\rho(t)) \qquad (7)$$

$$\delta_y(t) = y(t) - \bar{y}(\rho(t)) \qquad (8)$$

Using Taylor series expansion about $(\bar{x}(\rho(t)), \bar{u}(\rho(t)), \bar{y}(\rho(t)))$, the system dynamics can be expressed as (dropping the notational dependence on time):

$$f(x, u, \rho) = f(\bar{x}(\rho), \bar{u}(\rho), \rho) + A(\rho)\delta_x + B(\rho)\delta_u + \Delta_f(\delta_x, \delta_u, \rho) \qquad (9)$$

$$h(x, u, \rho) = h(\bar{x}(\rho), \bar{u}(\rho), \rho) + C(\rho)\delta_x + D(\rho)\delta_u + \Delta_h(\delta_x, \delta_u, \rho) \qquad (10)$$

where $f(\bar{x}(\rho), \bar{u}(\rho), \rho) = 0$, $h(\bar{x}(\rho), \bar{u}(\rho), \rho) = \bar{y}(\rho)$, $\Delta_f(\delta_x, \delta_u, \rho)$ and $\Delta_h(\delta_x, \delta_u, \rho)$ terms represent higher-order terms of the Taylor series approximations, and

$$A(\rho) = \frac{\partial}{\partial x} f(x,u,\rho)\Big|_{(x,u)=(\bar{x}(\rho),\bar{u}(\rho))} \qquad (11)$$

$$B(\rho) = \frac{\partial}{\partial u} f(x,u,\rho)\Big|_{(x,u)=(\bar{x}(\rho),\bar{u}(\rho))} \qquad (12)$$

$$C(\rho) = \frac{\partial}{\partial x} h(x,u,\rho)\Big|_{(x,u)=(\bar{x}(\rho),\bar{u}(\rho))} \qquad (13)$$

$$D(\rho) = \frac{\partial}{\partial u} h(x,u,\rho)\Big|_{(x,u)=(\bar{x}(\rho),\bar{u}(\rho))} \qquad (14)$$

Using this Taylor series approximation to linearize the dynamics of the nonlinear system, yields:

$$\begin{aligned}\tfrac{d}{dt}\delta_x &= \tfrac{d}{dt}\left(x - \bar{x}(\rho)\right) \\ &= \dot{x} - \tfrac{d}{dt}\bar{x}(\rho) \\ &= f(x,u,\rho) - \tfrac{d}{dt}\bar{x}(\rho) \\ &= A(\rho)\delta_x + B(\rho)\delta_x + \Delta_f(\delta_x,\delta_u,\rho) - \tfrac{d}{dt}\bar{x}(\rho)\end{aligned} \qquad (15)$$

Similarly, the Taylor series approximation of $h(x,y,\rho)$ can be used to linearize the output equation:

$$\begin{aligned}\delta_y &= y - \bar{y}(\rho) \\ &= h(x,y,\rho) - \bar{y}(\rho) \\ &= [\bar{y}(\rho) + C(\rho)\delta_x + D(\rho)\delta_x + \Delta_h(\delta_x,\delta_u,\rho)] - \bar{y}(\rho) \\ &= C(\rho)\delta_x + D(\rho)\delta_x + \Delta_h(\delta_x,\delta_u,\rho)\end{aligned} \qquad (16)$$

The final LPV model is thus:

$$\begin{aligned}\tfrac{d}{dt}\delta_x &= A(\rho)\delta_x + B(\rho)\delta_x + \Delta_f(\delta_x,\delta_u,\rho) - \tfrac{d}{dt}\bar{x}(\rho) \\ \delta_y &= C(\rho)\delta_x + D(\rho)\delta_x + \Delta_h(\delta_x,\delta_u,\rho)\end{aligned} \qquad (17)$$

Approximations

Standard LPV approach is to neglect higher order terms Δ_f and Δ_h, and the $-\dot{\bar{x}}$ term. However, the $-\dot{\bar{x}}$ term can be retained and treated as a measurable disturbance. This can be expressed as $-\dot{\bar{x}} = G(\rho)\dot{\rho}$, where $G(\rho) = -\frac{d\bar{x}(\rho)}{d\rho}$ The higher order terms Δ_f and Δ_h are nonlinear functions. They can be handled (locally) as uncertainties using integral quadratic constraints.

An interested reader, can refer to the work by Takarics and Seiler [1] for additional details on this approach. If an analytical linearization is not possible, an LPV model can be constructed using numerical linearization directly from a nonlinear model (e.g. a Simulink model). Refer to section XXX for details.

Example

Consider the nonlinear system (from [2]):

$$\begin{bmatrix}\dot{x}_1(t)\\ \dot{x}_2(t)\end{bmatrix} = \begin{bmatrix}-1 & 0\\ 1 & 0\end{bmatrix}\begin{bmatrix}x_1(t)\\ x_2(t)\end{bmatrix} + \begin{bmatrix}1\\ 0\end{bmatrix}u + \begin{bmatrix}0\\ -x_2|x_2| - 10\end{bmatrix} \qquad (18)$$

$$y(t) = x_2 \qquad (19)$$

Lets assume that we are given the control objective to make the output $y(t)$ track a reference command $r(t)$. We will frame this as a LPV control problem, and derive a LPV model of this nonlinear model for this problem.

Let the desired operating point be scheduled y $\rho = r$. In this formulation neither the dynamics (f in Equation (1)), nor the output equation (h in Equation (2)) directly depend on ρ.

The equilibrium point, parameterized by ρ is given by:

$$\bar{x}_1(\rho) = \rho|\rho| + 10 \qquad (20)$$

$$\bar{x}_2(\rho) = \rho \qquad (21)$$

$$\bar{u}(\rho) = \rho|\rho| + 10 = \bar{x}_1(\rho) \qquad (22)$$

$$\bar{y}(\rho) = \rho \qquad (23)$$

Applying the approach described above, the nonlinear system in Equations (18)-(19) is linearized about the parameterized equilibrium point to obtain a LPV system:

$$\dot{\delta_x} = \left[\begin{array}{cc} -1 & 0 \\ 1 & -2|\rho|) \end{array}\right] \delta_x + \left[\begin{array}{c} 1 \\ 0 \end{array}\right] u + \left[\begin{array}{c} -2|\rho| \\ 1) \end{array}\right] \qquad (24)$$

$$\delta_y = \left[\begin{array}{cc} 0 & 1 \end{array}\right] \delta_x \qquad (25)$$

By formulating the control problem in the form of a LPV system which described the behaviour of the nonlinear system about a desired reference command, we have recast the problem into a regulation problem: $\delta_y = y - \bar{y}(\rho) = y - \rho$ and the control objective is to regulate $\delta_y(t) \rightarrow 0$ in the LPV model.

If we neglect the $-\dot{\bar{x}}$ term in the LPV system of Equations (24)-(25), a grid-based LPV model of the system for $\rho \in [-5010]$ can be constructed using the following commands:

```
% Define the parameter
p = pgrid('p',[-5 0 10]);

% Define the system matrices
A = [-1 0;1 -2*abs(p)];
B = [1;0];
C = [0 1];

% Define the grid-based LPV model
sys = ss(A,B,C,0)
```

```
PSS with 2 States, 1 Outputs, 1 Inputs, Continuous System.
The PSS consists of the following blocks:
  p: Gridded real, 3 points in [-5,10], rate bounds [-Inf,Inf].
```

If we treat the $-\dot{\bar{x}}$ term as a exogenous disturbance to the model then the grid-based LPV system can be modeled as:

```
Bd = [-2*abs(p);-1];
sys_dis = ss(A,[B Bd],C,0)
```

```
PSS with 2 States, 1 Outputs, 2 Inputs, Continuous System.
The PSS consists of the following blocks:
  p: Gridded real, 3 points in [-5,10], rate bounds [-Inf,Inf].
```

The $\dot{\rho}$ term in $-\dot{\bar{x}}$ is now an input to the model. It is being treated as an exogenous disturbance, that is independent of ρ. This assumption is, in general, conservative.

References

1. B. Takarics and P. Seiler, "Gain Scheduling for Nonlinear Systems via Integral Quadratic Constraints," *accepted to the American Control Conference*, 2015.
2. D. J. Leith and W. E. Leithead, "Counter-Example to a Common LPV Gain-Scheduling Design Approach," *UKACC International Control Conference*, 2000.

Deriving LPV models from Nonlinear Simulation Models

Contents

Introduction

Numerical linearization can be used to derive grid-based LPV models from nonlinear simulation models. The goal is then to approximate the nonlinear model as a grid-based LPV system by generating a array of state-space models with consistent state vectors, inputs and outputs. This section will outline one approach to generating an array of state-space models from a Simulink model. The process is as follows:

Overview of Modeling Process

1. Desired LPV model depends on n_ρ parameters, $\rho \in \mathcal{R}^{n_\rho}$.
2. Create a grid of parameter values P.
3. Trim the nonlinear model at each grid point.
4. Linearize the nonlienar model at each trim point.

Gridding the Parameter Space

The choice of scheduling parameter ρ is at the users's discretion. A common choice for aircraft applications is Mach and altitude, due to the change in aircraft dynamics as a function of these two parameters.

Once the set of scheduling parameters has been chosen the set is gridded to form P. The grid should be made dense enough to capture significant dynamics in the model. The trade-off is that too dense a grid will be cumbersome from a computational perspective.

Example

Given an aircraft model with Mach and altitude chosen as the scheduling parameters, we are interested in developing a model for the aircraft at Mach values between 0.5 and 0.8, and at altitudes between 5,000 ft and 15,000 ft.

Lets assume that the dynamic of the aircraft vary smoothly inside this flight envelope. A first attempt at deriving an LPV model for this system might grid the parameters as follows:

$$Mach \in [0.5, 0.6, 0.7, 0.8]$$

$$Altitude \in [5,000\ ft, 10,000\ ft, 15,000\ ft]$$

The set of gridded parameter values P consists of 12 points arranged in a 4x3 grid defined as follows:

$$P = [Mach, altitute] \in [0.5, 0.6, 0.7, 0.8] \times [5,000\ ft, 10,000\ ft, 15,000\ ft]$$

This grid can be defined as an `rgrid` object in LPVTools using the following commands:

```
Mach = [0.5:0.1:0.8];
Altitude = [5000:5000:15000];
P = rgrid({'Mach','Altitude'},{Mach,Altitude})
```

```
RGRID with the following parameters:
  Mach: Gridded real, 4 points in [0.5,0.8], rate bounds [-Inf,Inf].
  Altitude: Gridded real, 3 points in [5e+03,1.5e+04], rate bounds [-Inf,Inf].
```

Trimming in MATLAB/Simulink

Once a desired grid of parameter values, P, has been chosen. The nonlinear simulation model will need to be trimmed and linearized at each grid point.

The process of trimming a Simulink model relies on the following proceedures and MATLAB commands:

- Start by manually adding the desired model input/output points to the relevant signal lines in the Simulink model (see the help for `getlinio` for details). Then use `getlinio` to create a object that describes the input/output points of the desired linearized model.
- Use `operspec` to grab the operating point specification of the Simulink model. The resulting object contains fields for each input/output point and state in the Simulink model.
- Loop through the parameter grid, P, and at each point: (1) Configure the `operspec` object to specify the desired trim point (i.e. set bounds on inputs, outputs, states, and their derivatives). (2) Use `findop` to trim the Simulink model at the desired grid point. The process is shown graphically in Figure 1.

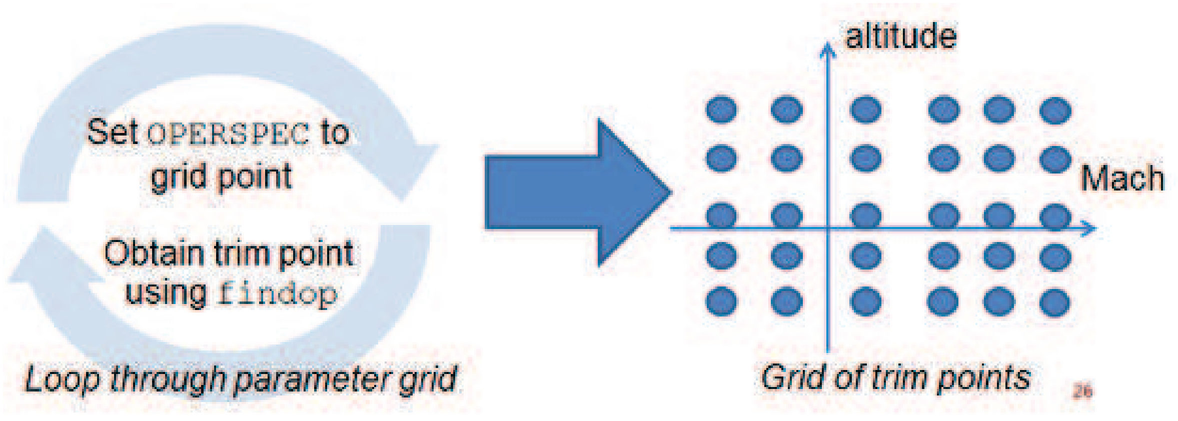

Set OPERSPEC to grid point
Obtain trim point using findop
Loop through parameter grid
altitude
Mach
Grid of trim points
26

Figure 1: Trimming a Simulink model.

Linearization in MATLAB/Simulink

Once a valid trim point has been created for each point in the parameter grid P. The function `linearize` is used to derive a linearized model at each point. The process is shown in Figure 2. Care must be taken that the linearized models that are being generated share a consistent input, output, and state vector. The resulting array of state-space models can be combined with an `rgrid` object that described the parameter grid P to form a grid-based LPV model. Refer to the grid-based LPV modeling tutorial for details on how a grid-based LPV model is assembled out of state-space model data.

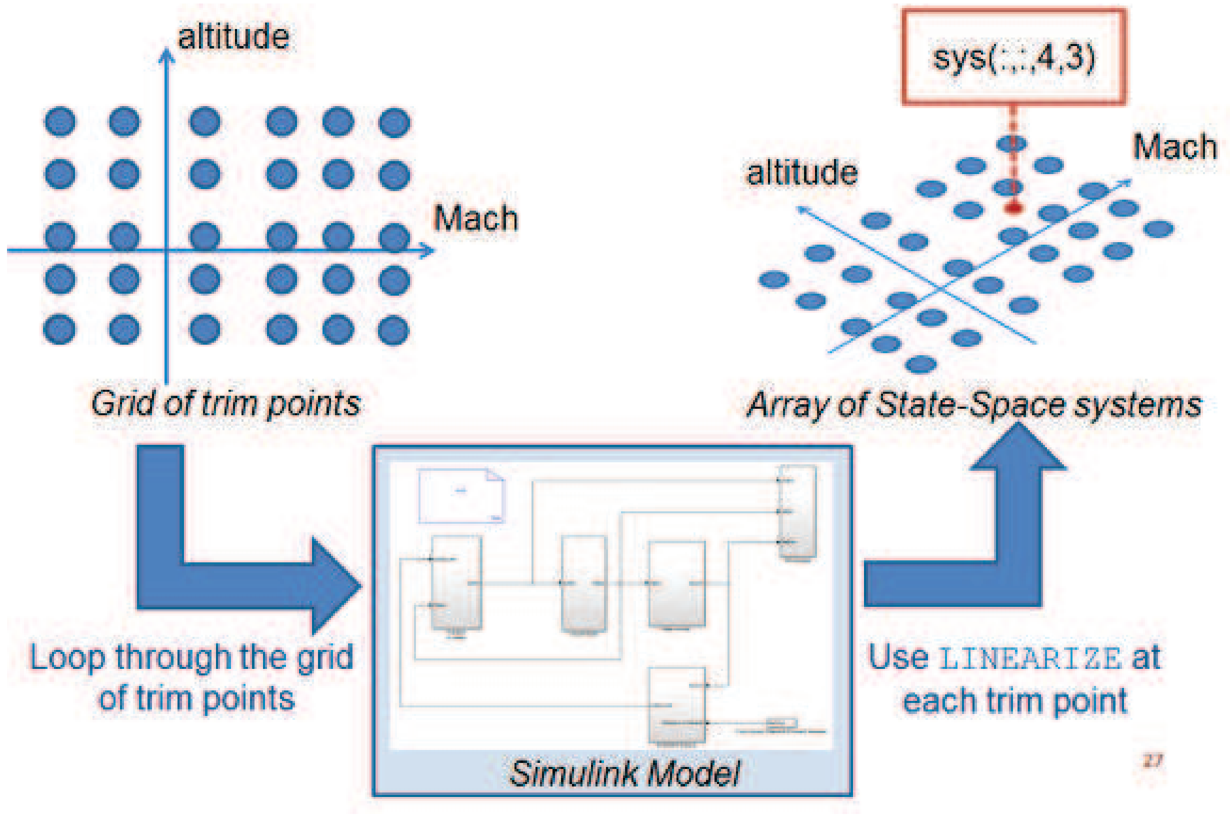

altitude
Mach
Grid of trim points
sys(:,:,4,3)
Mach
altitude
Array of State-Space systems
Loop through the grid of trim points
Simulink Model
Use LINEARIZE at each trim point
27

Figure 2: Linearizing a Simulink model on a grid of parameter values.

Published with MATLAB® R2014b

Permissible Parameter Trajectories

Contents

Introduction

An LPV system is a time-varying, state-space model of the form:

$$\begin{bmatrix} \dot{x}(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} A(\rho(t)) & B(\rho(t)) \\ C(\rho(t)) & D(\rho(t)) \end{bmatrix} \begin{bmatrix} x(t) \\ u(t) \end{bmatrix} \qquad (1)$$

The LPV model in Equation (1) describes how the LPV system depends on a set of time-varying parameters. Its important to understand that for practical applications (e.g. analysis in the LPV framework) each time-varying parameter in (1) has associated with it a set of *permissible parameter trajectories*, which describe how the parameter can change with time in the model. The permissible parameter trajectories contstrain the parameter values to those for which the model is valid.

The set of allowable trajectories for a particular parameter satisfies two properties: First, the parameter's value remains inside some interval of allowable values $[\rho_{min}, \rho_{max}]$ (an interval on the real line). Second, the parameter's rate of change lies inside some interval $[\overline{\nu}, \underline{\nu}]$ (also an interval on the real line). Hence, for an LPV system that only depends on a single parameter $\rho \in \mathcal{R}$, a permissible trajectory is any trajectory such that: $\rho_{min} \leq \rho(t) \leq \rho_{max}$ and $\underline{\nu} \leq \dot{\rho}(t) \leq \overline{\nu}$ for all t. A trajectory is said to be "rate unbounded" if $\overline{\nu} = \infty$ and $\underline{\nu} = -\infty$.

Example

Lets assume the LPV model in Equation (1) represents an aircraft, and that the model is scheduled on altitude h. If this particular model is only valid for a limited range of altitudes h: $5000\ ft \leq h \leq 10000\ ft$, and for slow variations in altitude $-10\ ft/sec \leq \dot{h} \leq 10\ ft/sec$, then the set of permissible parameter trajectories contains any altitude trajectory such that

$$h(t) \in [5000, 10000]\ ft,\ \forall t$$

and

$$\dot{h}(t) \in [-10, 10]\ ft/sec,\ \forall t$$.

An example of a permissible parameter trajectory for this system is shown in Figure 1.

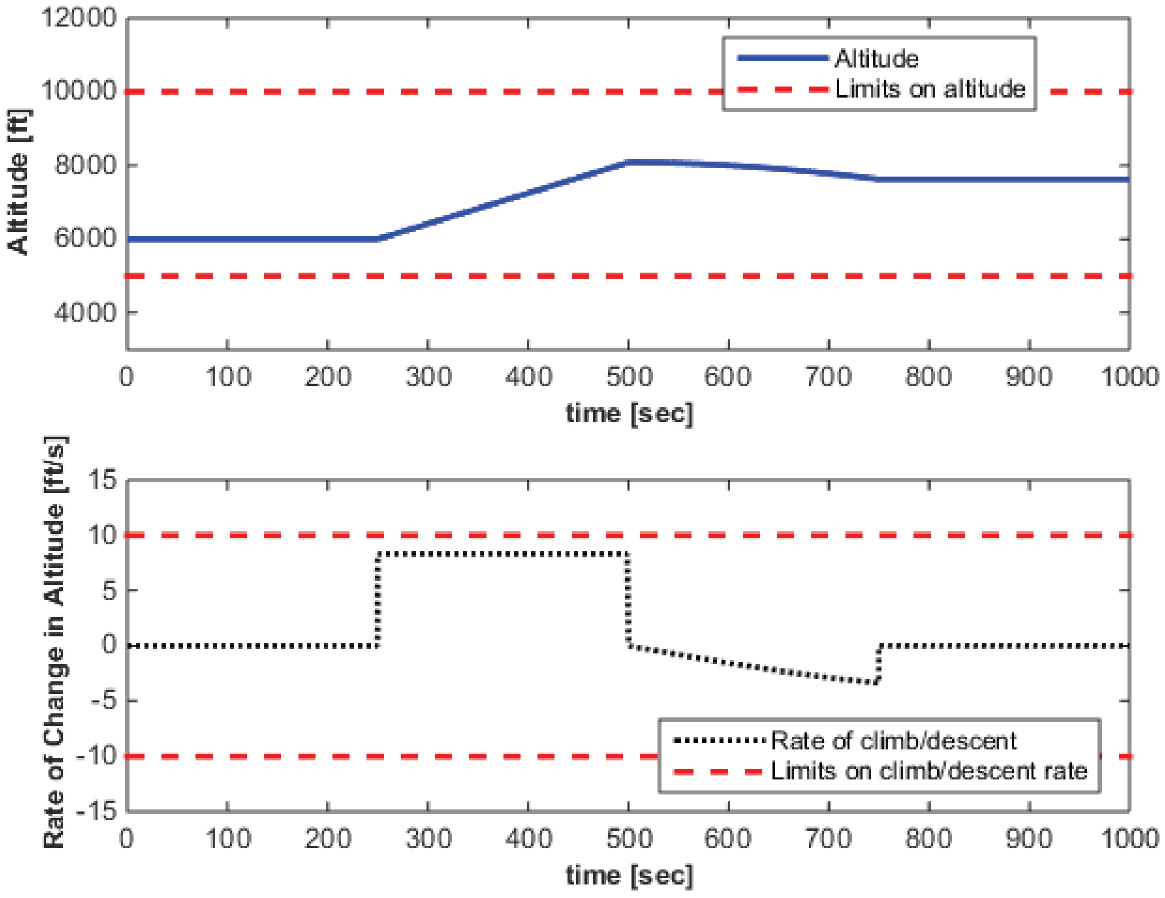


Figure 1: A permissible altitude trajectory.

Formal Definition

Given an LPV system that depend on a set of time-varying parameters $\rho \in \mathcal{R}^{n_\rho}$. A permissible parameter trajectory is any trajectory such that ρ lies inside the compact set $\mathcal{P} \subseteq \mathcal{R}^{n_\rho}$ and $\dot{\rho}(t)$ lies inside the set $\mathcal{D} \subseteq \mathcal{R}^{n_\rho}$. The set $\mathcal{P}$ is the n_ρ dimensional hyper rectangle formed by $[\rho_{1,min}, \rho_{1,max}] \times [\rho_{2,min}, \rho_{2,max}] \times \ldots \times [\rho_{n_\rho,min}, \rho_{n_\rho,max}]$, and the set $\mathcal{D}$ is the n_ρ dimensional hyper rectangle formed by $[\underline{\nu}_1, \overline{\nu}_1] \times [\underline{\nu}_2, \overline{\nu}_2] \times \ldots \times [\underline{\nu}_{n_\rho}, \overline{\nu}_{n_\rho}]$.

Published with MATLAB® R2014b

Quasi-LPV Systems

A quasi-LPV system refers to case where the parameter trajectory is a function of the system state and inputs, i.e. $\rho(x,u,t)$ where x is the system state, u is the system input, and t is time. The term "quasi" is used to indicate that the parameter dependence actually introduces a nonlinearity in the system dynamics.

The current implementation of LPVtools treats quasi-LPV systems just like any other LPV system for the purposes of analysis and synthesis, i.e. the parameter is assumed to vary independently of the system's state and input. This assumption will result in conservative results when working with quasi-LPV systems in LPVTools. However, quasi-LPV systems can be simulated correctly using the `lpvlsim`, `lpvstep`, `lpvimpulse`, and `lpvinitial` functions in LPVTools. Specifically, the the parameter trajectory that is used in the simulation can be specified as a function of the state, input, and time.

Published with MATLAB® R2014b

State Consistency

LPV systems are time-varying, state-space models of the form:

$$\begin{bmatrix} \dot{x}(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} A(\rho(t)) & B(\rho(t)) \\ C(\rho(t)) & D(\rho(t)) \end{bmatrix} \begin{bmatrix} x(t) \\ u(t) \end{bmatrix} \qquad (1)$$

where $\rho \in \mathcal{R}^{n_\rho}$ is a vector of measurable parameters, $y \in \mathcal{R}^{n_y}$ is a vector of outputs, $x \in \mathcal{R}^{n_x}$ is the state vector, $u \in \mathcal{R}^{n_u}$ is a vector of inputs, and $A \in \mathcal{R}^{n_x \times n_x}$, $B \in \mathcal{R}^{n_x \times n_u}$, $C \in \mathcal{R}^{n_y \times n_x}$ and $D \in \mathcal{R}^{n_y \times n_u}$ are parameter dependent matrices.

Note that the state-vector of the system in Equation (1) remains the same for all values of the parameter, i.e. the states in x are ordered the same way, and their interpretation remains the same, irrespective of the value of ρ. This property is referred to as *state consistency*, and it must be kept in mind when working with LPV models.

State Consistency in LPV Model Construction

A common approach to constructing LPV models is to use Jacobian linearization along a grid of parameter values (e.g. batch linearization of Simulink models) to construct a grid-based LPV system. In this case, the user must ensure that the models generated by the linearization all share the same state-vector.

Figure 1 illustrates the concept. A nonlinear model is linearized along a grid of Mach and altitude values, resulting in an array of linearized systems. State consistency requires the state vectors (x) of all the individual linearizations to be identical if these models are to be used to contruct a grid-based LPV system.

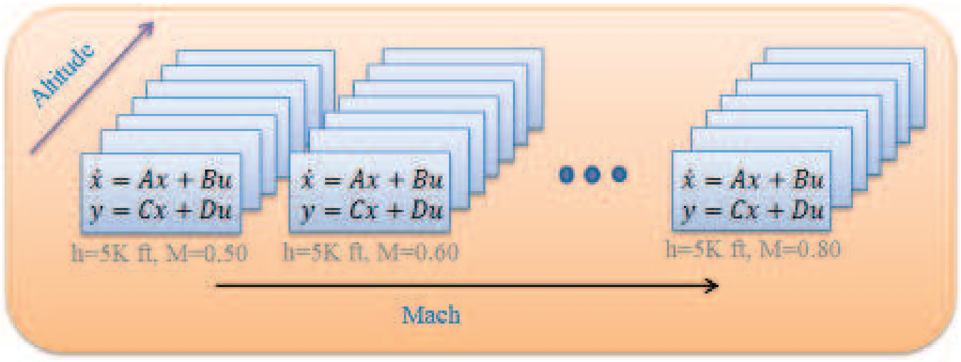


Figure 1: A grid-based LPV system.

Maintaining State Consistency

There are some operations that are commonly applied to Linear Time-Invariant (LTI) systems, that can result in loss of state-consistency of a LPV model. A good example is `balreal`, which performs a Gramian-based balancing of a LTI state-space realization. If `balreal` is applied to a grid-based LPV system it will balance each of the LTI models, which the grid-based LPV system is comprised of, and the resulting systems will no longer have state consistency. An alternative function that will maintain state consistency is `lpvbalreal` which computes a balancing realization for the LPV system as a whole, yielding a balanced LPV system with state consistency.

Published with MATLAB® R2014b

Analysis of LPV systems

A primer on analysis in the LPV framework.

Contents

LPV Analysis Commands

LPVNORM	Compute the gain of a LPV system.
LPVWCGAIN	Compute the worst-case gain of an uncertain LPV system.

Examples and How To

- Tutorial: Creating basis functions
- Tutorial: Analysis and simulation of gridded LPV systems
- Tutorial: Modeling and Control of LFT LPV Systems
- Tutorial: Worst-case LPV analysis using `lpvwcgain`
- Example: Stochastic LPV control of spinning mass
- Example: LPV control of spinning mass using LFT framework

Concepts

- Permissible Parameter Trajectories
- Stability and Induced Gain
- Uncertain Models
- Integral Quadratic Constraints

Published with MATLAB® R2014b

Stability and Gain of an LPV system

LPVTools provides a suite of functions to analyze the stability and gain of LPV systems. Meanwhile, LPVTools synthesis functions generate controllers that are provide closed-loop stability for an LPV system, while optimizing the gain. This section will discuss what stability and gain mean for an LPV system. Furthermore, this section highlights some of the computational issues that arise when LPV analysis conditions are implemented.

Contents

Stability and Gain of an LPV system

LPV systems are time-varying, state-space models of the form:

$$\begin{bmatrix} \dot{x}(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} A(\rho(t)) & B(\rho(t)) \\ C(\rho(t)) & D(\rho(t)) \end{bmatrix} \begin{bmatrix} x(t) \\ u(t) \end{bmatrix} \qquad (1)$$

where $\rho \in \mathcal{R}^{n_\rho}$ is a vector of measurable parameters, $y \in \mathcal{R}^{n_y}$ is a vector of outputs, $x \in \mathcal{R}^{n_x}$ is the state vector, $u \in \mathcal{R}^{n_u}$ is a vector of inputs, and $A \in \mathcal{R}^{n_x \times n_x}$, $B \in \mathcal{R}^{n_x \times n_u}$, $C \in \mathcal{R}^{n_y \times n_x}$ and $D \in \mathcal{R}^{n_y \times n_u}$ are parameter dependent matrices.

The LPV system in Equation (1) depends on a set of time-varying parameters ρ. The trajectories of the parameters are assumed to take on values in a known compact set $\mathcal{P} \subseteq \mathcal{R}^{n_\rho}$, and to have known bounds on their derivatives with respect to time: $\overline{\nu} \leq \dot{\rho} \leq \underline{\nu}$, where $\overline{\nu}$ and $\underline{\nu} \in \mathcal{R}^{n_\rho}$. A trajectory is said to be "rate unbounded" if $\overline{\nu} = \infty$ and $\underline{\nu} = -\infty$.

The LPV system processes the inputs u linearly, but can depend nonlinearly on the time-varying parameter ρ. The analysis problem is is to determine if the system is stable, and to quantify the input-to-output gain of the system. Denote the LPV system in Equation (1) by $G(\rho)$. Analysis in the LPV framework determines if $G(\rho)$ is internally exponentially stable, and whether the input/output map $G(\rho)$ from $u(t)$ to $y(t)$ has certain properties.

Definitions of Gain

LPVTools implements two methodologies for synthesis and analysis in the LPV framework. The two methodologies differ in their formulation of the input/output map $G(\rho)$. The first methodology formulates this input/output map in terms of the induced L_2 norm (gain) of the system:

$$\|G(\rho)\|_{2\to 2} = \max_{\rho\in\mathcal{P},\ \overline{\nu}\leq\dot{\rho}\leq\underline{\nu}} \quad \max_{u\in L_2,\ \|u\|_2\neq 0} \frac{\|G(\rho)u\|_2}{\|u\|_2} \qquad (2)$$

In calculating this induced norm it is assumed that $x(0) = 0$. The second methodology formulates the input/output map in terms of the stochastic LPV bound on $G(\rho)$:

$$stoch\left(G(\rho)\right) = \lim_{T\to\infty} \max_{\rho\in\mathcal{P},\ \overline{\nu}\leq\dot{\rho}\leq\underline{\nu}} E\left\{\frac{1}{T}\int_0^T y^T(t)y(t)dt\right\} \qquad (3)$$

which describes the variance of y when the input u is a zero mean, white-noise processes with unit intensity.

Computing the nominal L_2 norm of a grid-based LPV system:

`lpvnorm` implements algorithms to compute the gain of LPV systems. This section will review the analysis conditions that `lpvnorm` implements to compute the induced L_2 norm of a grid-based nominal (not uncertain) LPV system. These analysis conditions will serve to illuminate many of the key issues in LPV analysis techniques. Refer to the references at the end of this chapter for conditions used in other analysis scenarios.

The Objective

The theory underpinning the LPV analysis results which are implemented in `lpvnorm` frames the analysis problem in terms of a dissipation inequality. For the LPV system in Equation (1), the problem boils down to a set Linear Matrix Inequalities (LMIs) which need to be solved to prove that:

$$\int_0^T y(t)^T y(t)\, dt \leq \gamma^2 \int_0^T u(t)^T u(t)\, dt \qquad (4)$$

for all $\rho \in \mathcal{P}$ and $\overline{\nu} \leq \dot{\rho} \leq \underline{\nu}$, with some $\gamma \in \mathcal{R}^+$ and initial condition $x(0) = 0$.

Solving the LMIs to show that the dissipation inequality in Equation (4) holds, is sufficient to prove that the system is internally exponentially stable, and that the gain of the system has a finite upper bound (γ). The nominal induced L_2 norm analysis conditions used by `lpvnorm` are based on result by F. Wu. [1,2]

Analysis Conditions

The following theorem, taken from [1,2], gives a condition for an upper bound on the induced L_2 norm of the nominal LPV system $G(\rho)$ in Equation (1). For simplicity we will assume that the rate bounds on the parameter are symmetric: $\nu = \overline{\nu} = -\underline{\nu}$.

Theorem 1: If there exists a piecewise continuous symmetric function $X : \mathcal{R}^{n_\rho} \rightarrow \mathcal{R}^{n_x \times n_x}$ and a $\gamma \in \mathcal{R}^+$, such that $X(\rho) > 0$ and

$$\begin{bmatrix} A^T(\rho)X(\rho) + X(\rho)A(\rho) + \sum_{i=1}^{n_\rho} \beta_i \frac{\partial X}{\partial \rho_i} & X(\rho)B(\rho) & \gamma^{-1}C^T(\rho) \\ B^T(\rho)X(\rho) & -I_{n_u} & \gamma^{-1}D^T(\rho) \\ \gamma^{-1}C(\rho) & \gamma^{-1}D(\rho) & -I_{n_y} \end{bmatrix} < 0 \qquad (5)$$

$\forall \rho \in \mathcal{P}$, and $-\nu \leq \dot{\rho} \leq \nu$, with $|\beta_i| \leq \nu_i (i = 1, \ldots, n_\rho)$, then:

- The system G is parametrically-dependent stable over $\mathcal{P}$.
- $\exists k$ with $0 \leq k < \gamma$ such that $\|G\|_{2\rightarrow 2} \leq k$.

The theorem above assume that the rate bounds of the time-varying parameter are symmetric, but it can be extended to the unsymmetric case, and the software handles the unsymmetric case. The conditions in Theorem 1 are a parameterized set of linear matrix inequalities (LMIs) that must be verified for all $\rho \in \mathcal{P}$ and all $|\beta_i| \leq \nu_i$. The conditions are infinite dimensional, since $A(\rho)$, $B(\rho)$, $C(\rho)$, $D(\rho)$ and $X(\rho)$ are all continuous functions of the parameter ρ.

Implementation in LPVTools

Its possible to obtain an approximate solution to the infinite dimensional feasibility conditions in Theorem 1 by converting them into a finite-dimensional set of Linear Matrix Inequalities (LMIs). This is accomplished by the following proceedure:

1. Grid the set $\mathcal{P}$ into a set of n_r parameter values: $\{\hat{\rho}_1, \hat{\rho}_2, \dots \hat{\rho}_{n_r}\}$. Require that the LMIs in Equation (5) hold at each grid point.
2. Pick a basis for $X(\rho)$ so that $X(\rho) = \sum_{k=1}^{n_b} f_k(\rho) X_k$, where n_b is the number of basis functions used to construct $X(\rho)$, the scalar functions $f_1, \dots, f_{n_b} : \mathcal{R}^{n_\rho} \rightarrow \mathcal{R}$ are the chosen basis functions, and $X_1, \dots, X_{n_b} \in \mathcal{R}^{n_x \times n_x}$ are constant matrices to be determined (see the tutorial on picking basis functions for an example of how $f_1, \dots, f_{n_b}$ are defined in LPVTools). If the parameter's in the LPV system are rate unbounded (i.e. $\nu = \infty$) then use a constant (parameter independent) Lyapunov matrix $X(\rho) = X \in \mathcal{R}^{n_x \times n_x}$.
3. Exploit the fact that the β_i enter affinely in Equation (4) to reduce the problem to 2^{n_ρ} LMIs at each grid point. Specifically, if the LMIs hold for all combinations of $\beta_i = \pm\nu_i$ (a total of 2^{n_ρ} combinations formed by the n_ρ-dimensional polytope: $[-\nu_1, \nu_1] \times [-\nu_2, \nu_2] \times \ldots \times [-\nu_{n_\rho}, \nu_{n_\rho}]$) then they hold for all $|\beta_i| \leq \nu_i$. This reduces the problem to $n_r 2^{n_\rho}$ LMIs total (n_r grid points, with 2^{n_ρ} LMIs at each point.)
4. Solve for γ and $X_1, \dots, X_{n_b}$, subject to the $(n_r 2^{n_\rho})$ LMIs formed at the grid points by the condition in Equation (5).

The function `lpvnorm` implements this proceedure to approximately solve the conditions in Theorem 1 by enforcing the LMIs on the set of gridded points in the domain of the grid-based LPV system (for a grid-based LPV system the set of possible ρ values, $\mathcal{P}$, is gridded as a matter of course during the modeling process).

The computational growth of these conditions is an issue. Let n_r denote the total number of grid points used to approximate $\mathcal{P}$. A rate bounded analysis must enforce the LMI conditions at all n_r grid points and for all 2^{n_ρ} combinations of $\beta_i = \pm\nu_i$. Thus there are a total of $n_r 2^{n_\rho}$ constraints, each of dimension $(n_x + n_u + n_y)$. If there are n_b basis functions, then the Lyapunov matrix has n_b symmetric matrix decision variables $\{X_j\}_{j=1}^{n_b}$ each of dimension $n_x \times n_x$. This gives a total of $n_r \frac{n_x(n_x+1)}{2}$ individual decision variables in the rate bounded analysis. LMI optimization solvers have an asymptotic complexity that depends on both the number of decision variables and the number/dimension of LMI constraints. For example, LMILab has a floating point operation growth of O($n_{row} n_v^3$) where n_{row} is the total row dimension of the LMI conditions and n_v is the total number of decision variables [3]. This complexity assumes the default Cholesky factorization of the Hessian matrix is used to solve the least squares problem that arises in each iteration. Thus the complexity of solving the LPV analysis condition is roughly $O\left(n_r 2^{n_\rho}(n_x + n_u + n_y)\left(n_b n_x^2\right)^3\right)$. This growth limits the analysis to a modest number of parameters, grid points, and basis functions.

Alternative Approaches

The LPV analysis problem is formulated differently when the system is represented in the LFT-based LPV framework. In this case, the rate-bounds can still be taken into account in the analysis, but they do not require the user to define basis functions. The resulting feasability conditions are different from the ones listed in the grid-based LPV analysis above. However, the implementations of the two approaches have many features in common: Solution involves convex constraints (LMIs), and the complexity grows with $O(2^{n_\rho})$. Further information on the analysis conditions for the LFT-based LPV approach can be found in P. Apkarian and P.Gahinet [4], A. Packard [5], A. Helmersson [6], and C. Scherer [7].

The analysis conditions that apply for the stochastic LPV bound can be found in the work by F. Wu [1], and the results for worst-case LPV analysis can be found in C. Scherer [7,8,9] and H. Pfifer and P. Seiler [10].

References

1. F. Wu, "Control of Linear Parameter Varying Systems," PhD thesis, University of California, Berkeley, 1993.
2. F. Wu, X. Yang, A. Packard, and G. Becker, "Induced L2 norm control for LPV systems with bounded parameter variation rates," *International Journal of Nonlinear and Robust Control*, vol. 6, pp. 983-998, 1996.
3. P. Gahinet, A. Nemirovski, A. Laub, and M. Chilali, "LMI control toolbox user's guide," tech. rep., The Mathworks,

1995.

4. P. Apkarian and P.Gahinet, "A convex characterization of gain-scheduled Hinfinity controllers," *IEEE Transactions on Automatic Control*, vol. 40, no. 5, pp. 853-864, 1995.
5. A. Packard, "Gain scheduling via linear fractional transformations," *Systems and Control Letters*, vol. 22, no. 2, pp. 79-92, 1994.
6. A. Helmersson, "An IQC-based stability criterion for systems with slowly varying parameters," Technical Report LiTH-ISYR-1979, Linkoping University 1997.
7. C. Scherer and S. Wieland, "Linear matrix inequalities in control," Lecture notes for a course of the dutch institute of systems and control, Delft University of Technology, 2004.
8. C. Scherer and I. Kose, "Robustness with dynamic IQCs: An exact state-space characterization of nominal stability with applications to robust estimation," *Automatica*, Vol. 44, No. 7, pp. 1666-1675, 2008.
9. C. Scherer, "LPV control and full-block multipliers," *Automatica*, Vol. 37, No. 3, pp. 361-375, 2001.
10. H. Pfifer, and P. Seiler. "Robustness analysis of linear parameter varying systems using integral quadratic constraints," *International Journal of Robust and Nonlinear Control*, 2014, doi: 10.1002/rnc.3240.

Published with MATLAB® R2014b

Integral Quadratic Constraints

Introduction

Integral Quadratic Constraints (IQCs) are used in some LPV analysis algorithms in LPVTools. Their function is to bound the input-to-output map of a system component for the purposes of analysis. IQCs were introduced by A. Megretski and A. Rantzer [1] to provide a general framework for robustness analysis.

An IQC is defined by a symmetric matrix $M = M^T \in \mathcal{R}^{n_z \times n_z}$ and a stable linear system $\Psi \in \mathcal{RH}^{n_z \times (m_1+m_2)}$. Ψ is denoted as

$$\Psi(j\omega) := C_\psi (j\omega I - A_\psi)^{-1} [B_{\psi 1} \; B_{\psi 2}] + [D_{\psi 1} \; D_{\psi 2}] \qquad (1)$$

A bounded, causal operator $\Delta : \mathcal{L}_{2e}^{m_1} \to \mathcal{L}_{2e}^{m_2}$ satisfies an IQC defined by (Ψ, M) if the following inequality holds for all $v \in \mathcal{L}_2^{m_1}[0,\infty)$, $w = \Delta(v)$ and $T \geq 0$:

$$\int_0^T z(t)^T M z(t)\, dt \geq 0 \qquad (2)$$

where z is the output of the linear system Ψ:

$$\dot{x}_\psi(t) = A_\psi x_\psi(t) + B_{\psi 1} v(t) + B_{\psi 2} w(t), \; x_\psi(0) = 0 \qquad (3)$$

$$z(t) = C_\psi x_\psi(t) + D_{\psi 1} v(t) + D_{\psi 2} w(t) \qquad (4)$$

The notation $\Delta \in IQC(\Psi, M)$ is used if Δ satisfies the IQC defined by (Ψ, M). Figure 1 provides a graphic interpretation of the IQC. The input and output signals of Δ are filtered through Ψ. If $\Delta \in IQC(\Psi, M)$ then the output signal z satisfies the (time-domain) constraint in Equation (2) for any finite-horizon $T \geq 0$.

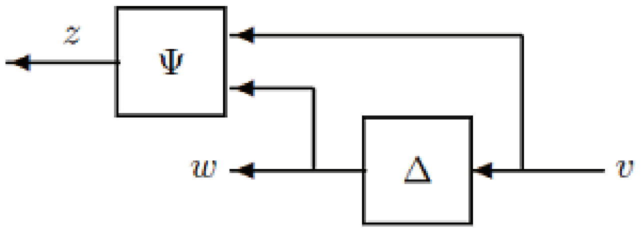


Figure 1: Graphic interpretation of the IQC.

Integral Quadratic Constraints in LPVTools

IQCs are used for worst-case analysis (`lpvwcgain`) of uncertain LPV systems (grid-based [3] and LFT-based [4,5,6]), and for analysis (`lpvnorm`) of nominal (not uncertain) rate-bounded LFT-based LPV systems [7]. In each case the IQCs are used to bound the input-to-output map of some element in the system (the uncertainty block for worst-case analysis, and the parameter block for rate-bounded analysis of LFT-based LPV systems). The implementation of these algorithms requires the user to specify basis functions for the stable linear system Ψ. The basis functions are currently constrained to be either constant or first order systems. The analysis functions require the user to supply a 1xN `double` row vector of positive numbers, which specify the real, stable poles of N first order basis functions to be used. If no vector is supplied, the software autmatically selects a constant term and three first order systems as the basis functions for Ψ in the analysis.

Additional Information

Reference [1] provides a library of IQC multipliers that are satisfied by many important system components, e.g. saturation, time delay, and norm bounded uncertainty. The IQCs in [1] are expressed in the frequency domain as an integral constraint defined using a multiplier Π. The multiplier Π can be factorized as $\Pi = \Psi^* M \Psi$ and this connects the frequency domain formulation to the time-domain formulation used here. One technical point is that, in general, the time domain IQC constraint only holds over infinite horizons ($T = \infty$). The work in [1,2] draws a distinction between hard/complete IQCs for which the integral constraint is valid over all finite time intervals and soft/conditional IQCs for which the integral constraint need not hold over finite time intervals. The formulation of an IQC here, as a finite-horizon (time-domain) inequality, is thus valid for any frequency-domain IQC that admits a hard/complete factorization (Ψ, M). While this is somewhat restrictive, it has recently been shown that a wide class of IQCs have a hard factorization [2].

References

1. A. Megretski, and A. Rantzer, "System Analysis via Integral Quadratic Constraints," *IEEE Transactions on Automatic Control*, Vol. 42, No. 6, pp. 819–830, 1997, doi: 10.1109/CDC.1994.411315.
2. A. Megretski, "KYP lemma for non-strict inequalities and the associated minimax theorem,”, *Arxiv*, 2010, (arXiv:1008.2552).
3. H. Pfifer and P. Seiler, "Robustness Analysis of Linear Parameter Varying Systems Using Integral Quadratic Constraints," *American Control Conference*, pp. 4476-4481, 2014, doi: 10.1109/ACC.2014.6858751.
4. C. Scherer and S. Wieland, "Linear matrix inequalities in control," Lecture notes for a course of the dutch institute of systems and control, Delft University of Technology, 2004.
5. C. Scherer and I. Kose, "Robustness with dynamic IQCs: An exact state-space characterization of nominal stability with applications to robust estimation," *Automatica*, Vol. 44, No. 7, pp. 1666-1675, 2008.
6. C. Scherer, "LPV control and full-block multipliers," *Automatica*, Vol. 37, No. 3, pp. 361-375, 2001.
7. A. Helmersson, "An IQC-based stability criterion for systems with slowly varying parameters," Technical Report LiTH-ISYR-1979, Linkoping University 1997.

Defining Basis Functions

Contents

Introduction

Basis functions are needed for rate-bounded LPV analysis and synthesis in the grid-based LPV framework. These are functions of the time-varying parameter present in the system being analyzed. Basis functions are specified using the `basis` object. To construct a `basis` function object the user provides a `pmat` or `pgrid` that defines the value of the basis function at each grid-point in the domain. Furthermore, the user provides the value of the partial derivative of the basis function with regard to each parameter in the system, at each point in the domain.

Constructing Basis Functions

Lets construct a `basis` object that describes the basis function $f = \rho(t) + \rho(t)^2$.

```
% Define the time-varying parameter rho
rho = pgrid('rho',1:5)
```

```
Gridded real parameter "rho" with 5 points in [1,5] and rate bounds [-Inf,Inf].
```

Define the basis function as a `pmat`:

```
f = rho + rho^2
```

```
PMAT with 1 rows and 1 columns.
The PMAT consists of the following blocks:
  rho: Gridded real, 5 points in [1,5], rate bounds [-Inf,Inf].
```

Define the value of the partial derivative of f with respect to rho:

```
pf = 1+2*rho
```

```
PMAT with 1 rows and 1 columns.
The PMAT consists of the following blocks:
  rho: Gridded real, 5 points in [1,5], rate bounds [-Inf,Inf].
```

Now we can define the `basis` object for this basis function. The first argument to `basis` is the value of the basis function. The second argument is the value of the partial derivative:

```
bf = basis(f,pf)
```

```
BASIS: 1 basis functions and 1 partial derivatives with respect to 1 PGRID
The BASIS object consists of the following blocks:
  rho: Gridded real, 5 points in [1,5]
```

Basic Arithmetic for Basis Functions

The `basis` object includes methods for basic arithmetic. Hence, an initial `basis` object can be used to construct others. Lets define another `basis` object that describes the basis function: $g = f^2 = \rho^2 + 2\rho^3 + \rho^4$.

```
bg = bf^2
```

```
BASIS: 1 basis functions and 1 partial derivatives with respect to 1 PGRID
The BASIS object consists of the following blocks:
  rho: Gridded real, 5 points in [1,5]
```

Note that there is no need to specify the partial derivatives for the new system `g`. These are automatically computed for `g` using the data in `f` and the chain rule of differentiation.

Aggregating Basis Functions

For analysis and synthesis, a set of basis functions can be groupped together using horzcar or vertcat:

```
BF = [bf,bg]
```

```
BASIS: 2 basis functions and 1 partial derivatives with respect to 1 PGRID
The BASIS object consists of the following blocks:
  rho: Gridded real, 5 points in [1,5]
```

In this case the `basis` object `BF` describes the two basis functions $f = \rho(t) + \rho(t)^2$ and $g = f^2 = \rho^2 + 2\rho^3 + \rho^4$.

Published with MATLAB® R2014b

Worst-Case Gain Analysis of LPV System

The following example is originally from [1], and was posed as an LPV analysis problem in [2].

Contents

An uncertain parameter dependent system

Consider a first order parameter dependent system $G(\rho)$:

$$\begin{aligned} \dot{x}_G &= -\frac{1}{\tau(\rho)}x_G + \frac{1}{\tau(\rho)}u_G \\ y &= K(\rho)x_G \end{aligned} \qquad (1)$$

where the elements $\tau(rho)$ and $K(\rho)$ are dependent on the parameter ρ as follows:

$$\tau(\rho) = \sqrt{133.6 - 16.8\rho} \qquad (3)$$

$$K(\rho) = \sqrt{4.8\rho - 8.6} \qquad (4)$$

The following analysis will study the system when the parameter is restricted to the interval $[2, 7]$. $G(\rho)$ is placed into an interconnection with a time delay $T_d = 0.5$ sec, and a multiplicative uncertainty Δ, as shown in Figure 1.

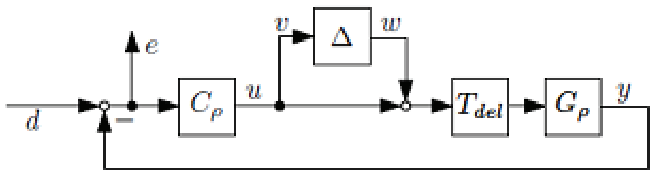


Figure 1: Interconnection for analysis.

The induced L_2 norm of the uncertainty is bounded by $\|\Delta\|_2 \le 0.1$, and the time delay is modeled by a second order Padé approximation $T_{del}(s)$:

$$T_{del}(s) = \frac{\frac{(T_d s)^2}{12} - \frac{T_d s}{2} + 1}{\frac{(T_d s)^2}{12} + \frac{T_d s}{2} + 1} \qquad (5)$$

A gain-scheduled Proportional-Integral controller $C_\rho(\rho)$ has been designed for this system. It is designed to achieve a a closed loop damping $\zeta_{cl} = 0.7$ and a closed loop frequency of $\omega_{cl} = 0.25$ at each point in the domain. The controller C_ρ has the followig form:

$$\begin{aligned} \dot{x}_c &= K_i(\rho) x_c e \\ u &= x_c + K_p(\rho) e \end{aligned} \qquad (6)$$

where the gains $K_i(\rho)$ and $K_p(\rho)$ are chosen as:

$$K_p(\rho) = \frac{2\zeta_{cl}\omega_{cl}\tau(\rho) - 1}{K(\rho)} \qquad (7)$$

$$K_i(\rho) = \frac{\omega_{cl}^2 \tau(\rho)}{K(\rho)} \qquad (8)$$

The analysis problem is to compute the worst-case induced L_2 norm from d to e in the interconnection shown in Figure 1.

Construct LPV system

The first step in the analysis is to construct an LPV model that represents the interconnection of systems in Figure 1.

```
% Define the parameter as a gridded real with 6 evenly space grid points:
p = pgrid('p',2:7);

% Define the plant
tau = sqrt(133.6-16.8*p);
K = sqrt(4.8*p-8.6);
G = ss(-1/tau,1/tau,K,0);

% Define the time delay:
Td = 0.5;
Tdel = tf([Td^2/12 -Td/2 1],[Td^2/12 Td/2 1]);

% Define the controller:
sigma = 0.7;
wcl = 0.25;
Kp = (2*sigma*wcl*tau-1)/K;
Ki = wcl^2*tau/K;
C = ss(0,Ki,1,Kp);

% Define the uncertainty:
Delta= ureal('Delta',0,'Range',[-.1 .1]);

% Apply a multiplicative uncertainty and time delay to input of plant:
Plant = G*Tdel*(1+Delta);

% Form closed-loop interconnection
systemnames = 'C Plant';
inputvar = '[d]';
outputvar = '[d-Plant]';
input_to_Plant = '[C]';
input_to_C = '[d-Plant]';
CL = sysic
```

```
UPSS with 4 States, 1 Outputs, 1 Inputs, Continuous System.
The UPSS consists of the following blocks:
  p: Gridded real, 6 points in [2,7], rate bounds [-Inf,Inf].
  Delta: Uncertain real, nominal = 0, range = [-0.1,0.1], 1 occurrences
```

Worst case analysis without rate-bounds

```
% Lets compute the worst-case induced $L_2$ norm of the closed-loop
% interconnection |CL|. We will use the function |lpvwcgain| to achieve this.
% First, we will compute the norm when we assume that there are no bounds
% on the parameter rate ($\dot{\rho}$):

GAM = lpvwcgain(CL)
```

```
GAM =
   63.8659
```

The worst-case induced L_2 norm of the closed-loop interconnection is 63.87. This means that for all norm bounded uncertainties $\|\Delta\| \leq 0.1$, and all norm bounded inputs $\|d\|_2 \leq 1$, and all parameter trajectories such that $2 \leq \rho(t) \leq 7$, the induced L_2 norm of the output e is guaranteed to be no larger than 63.87. This is an upper bound.

To arrive at a lower bound, we can compute the largest worst-case norm of the Linear Time-Invariant (LTI) systems at each frozen parameter value, insert the value of the correspondig worst-case uncertainty into the LPV interconnection in `CL`, and compute the induced L_2 norm of the resulting nominal (no uncertainty) LPV system using `lpvnorm`:

```
% Compute the worst-case induced $L_2$ norm of the LTI systems
% corresponding to each grid point of p:
[WCG,WCU,INFO] = wcgain(CL);

% Identify and extract the worst-case uncertainty:
[V,I]=lpvmax(WCG.LowerBound,'p');
wc_delta = WCU.index('p',6);
wc_delta = wc_delta.Delta
```

```
wc_delta =
    0.1000
```

Reform the interconnection in Figure 1, using the worst-case multiplicative uncertainty:

```
% Apply a multiplicative uncertainty and time delay to input of plant:
wc_Plant = G*Tdel*(1+wc_delta);

% Form closed-loop interconnection
systemnames = 'C wc_Plant';
inputvar = '[d]';
outputvar = '[d-wc_Plant]';
input_to_wc_Plant = '[C]';
input_to_C = '[d-wc_Plant]';
wc_CL = sysic;

% Compute the induced L2 norm of the nominal (no uncertainty) LPV system
% wc_CL:
[Gamma,X] = lpvnorm(wc_CL);
Gamma
```

```
Gamma =
   20.1476
```

The results indicate that the worst-case induced L_2 norm of `CL` lies between 20.1476 and 63.8659.

Worst-case analysis with rate-bounds

Lets assume that the parameter can not change arbitrarily fast with time, that it is rate-bounded: $\|r\dot{h}o\| \leq 0.1$. In this case the previous result is conservative, because it assumes that there is no limit to how fast the parameter can change with time. A more accurate results can be achived if we take into account the rate-bound on the parameter. To do this we again use `lpvwcgain`, but now the rate-bounded analysis requires parameter dependent basis functions to compute the induced L_2 norm (the reason that these basis functions are needed can be found in a description of the LPV analysis conditions, elsewhere in this manual

`basis` objects are used to represent basis functions in LPVTools. There are no firm rules about the choice of basis functions, but a good rule of thumb is to keep them as simple as possible, due to the added computational burden associated with each independent basis function that is added. For this example we will compare results for a set of four basis functions $f_1(\rho)$, $f_2(\rho)$, $f_3(\rho$, and $f_4(\rho)$: * Set 1: $f_1(\rho) = [1, \rho]$ * Set 2: $f_2(\rho) = [1, \rho, \rho^2]$ * Set 3: $f_3(\rho) = [1, \rho, \rho^2, \frac{1}{\rho}]$ * Set 4: $f_4(\rho) = [1, \rho, \rho^2, \frac{1}{\rho}, \sqrt{\rho}]$

We will not repeat the previous analysis and compute the worst-case induced L_2 norm of the closed-loop interconnection `CL`, while taking into accound the parameter's rate bound $\|r\dot{h}o\| \leq 0.1$:

```
% Change the rate-bounds of p in the closed-loop interconnection:
rb = 0.1;
CLrb = CL;
CLrb.Parameter.p.RateBounds = [-rb rb];

% Define three basis objects: b0 = 1, b1 = p, and b2 = sqrt(p).
% The first argument to |basis| is the value of the basis function at each
% grid point. The second argument is the value of the partial derivative of
% the basis function with respect to the parameter:
b0 = basis(1,0);
b1 = basis(p,1);
b2 = basis(sqrt(p),1/(2*sqrt(p)));
```

Start by computing the worst-case induced L_2 norm of the closed-loop interconnection `CL` using the set of basis functions: $f_1(\rho) = [1, \rho]$

```
% Define set of basis functions:
basis1 = [b0,b1];
% Perform rate-bounded worst-case LPV analysis:
GAM = lpvwcgain(CLrb,basis1)
GAM =
    3.0906
```

Repeat the analysis with basis functions: $f_2(\rho) = [1, \rho, \rho^2]$, $f_3(\rho) = [1, \rho, \rho^2, \frac{1}{\rho}]$, and $f_4(\rho) = [1, \rho, \rho^2, \frac{1}{\rho}, \sqrt{\rho}]$

```
basis2 = [b0,b1,b1^2];
GAM = lpvwcgain(CLrb,basis2)
```

```
GAM =
    2.0080
```

```
basis3 = [b0,b1,b1^2,1/b1];
GAM = lpvwcgain(CLrb,basis3)
```

```
GAM =
    1.9363
```

```
basis4 = [b0,b1,b1^2,1/b1,b2];
GAM = lpvwcgain(CLrb,basis4)
```

```
GAM =
    1.8925
```

The rate-bounded analysis results in a far lower worst-case norm. Clearly it is important to take into account the permissible parameter rate of variation. The upper bound on the worst-case norm appears to converge close to 1.89.

Adding terms to the basis function improves the bound on the worst-case norm. In this example, the effect is drastic when going from a simple linear basis function to a quadratic basis function, but modest when more complicated terms are added.

The result computed by `lpvwcgain` is only an upper bound. We can repeat the process we used before to arrive at a lower bound:

```
% Compute the worst-case induced $L_2$ norm of the LTI systems
% corresponding to each grid point of p:
[WCG,WCU,INFO] = wcgain(CL);

% Identify and extract the worst-case uncertainty:
[V,I]=lpvmax(WCG.LowerBound,'p');
wc_delta = WCU.index('p',6);
wc_delta = wc_delta.Delta
wc_delta =
    0.1000
```

Reform the interconnection in Figure 1, using the worst-case multiplicative uncertainty:

```
% Apply a multiplicative uncertainty and time delay to input of plant:
wc_Plant = G*Tdel*(1+wc_delta);

% Form closed-loop interconnection
systemnames = 'C wc_Plant';
inputvar = '[d]';
outputvar = '[d-wc_Plant]';
input_to_wc_Plant = '[C]';
input_to_C = '[d-wc_Plant]';
wc_CLrb = sysic;

% Set rate-bound:
wc_CLrb.Parameter.p.RateBounds = [-rb rb];

% Compute the induced L2 norm of the nominal (no uncertainty) LPV system
% wc_CL:
[Gamma,X] = lpvnorm(wc_CLrb,basis4);
Gamma
```

```
Gamma =
    1.2968
```

The worst-case induced L_2 norm lies somewhere between 1.2968 and 1.8925.

References:

1. S. Tan, C. C. Hang, and J. S. Chai, “Gain scheduling from conventional to neuro-fuzzy,” *Automatica*, Vol. 33, pp. 411–419, 1997.
2. H. Pfifer and P. Seiler, “Robustness analysis of linear parameter varying systems using integral quadratic constraints,” in *American Control Conference*, 2014.

Published with MATLAB® R2014b

Synthesis for LPV systems

A primer on synthesis in the LPV framework.

Contents

LPV Synthesis

Problem Statement

Consider a parameter dependent linear plant P_ρ of the form:

$$\begin{bmatrix} \dot{x}(t) \\ e(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} A(\rho(t)) & B_1(\rho(t)) & B_2(\rho(t)) \\ C_1(\rho(t)) & D_{11}(\rho(t)) & D_{12}(\rho(t)) \\ C_2(\rho(t)) & D_{21}(\rho(t)) & D_{22}(\rho(t)) \end{bmatrix} \begin{bmatrix} x(t) \\ d(t) \\ u(t) \end{bmatrix} \qquad (1)$$

where ρ is a time varying parameter, that takes on values in a known compact set $\mathcal{P}$ and has known bound on $\dot{\rho}$, $\overline{\nu} \leq \dot{\rho} \leq \underline{\nu}$. The time variations of $\rho(t)$ are not known in advance, but the parameter values are measured in real-time and available for control design.

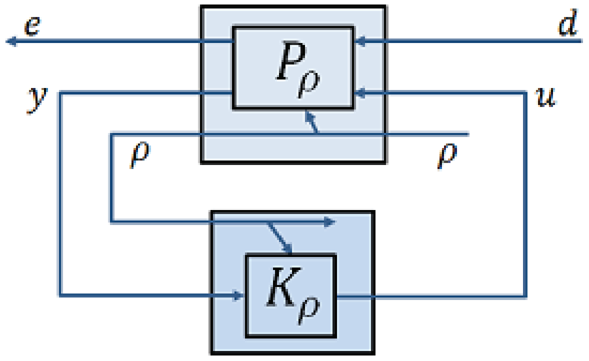


Figure 1: Closed-loop interconnection for LPV synthesis problem.

The control problem is to synthesize a controller K_ρ such that the closed-loop system shown in Figure 1, is stable and the gain from d to e is minimized. This requires that the controller be designed such that the closed-loop performance is optimized in the presence of rate-bounded, time-varying parameter trajectories $\rho \in \mathcal{P} \subset \mathcal{R}^n_\rho$. Denote the closed-loop system by $lft(P_\rho, K_\rho)$, and the gain of this closed-loop system by $\|lft(P_\rho, K_\rho)\|$ Then the design objective can be stated as:

$$\min_{K_\rho} \max_{\rho \in \mathcal{P}, \overline{\nu} \leq \dot{\rho} \leq \underline{\nu}} \|lft(P_\rho, K_\rho)\| \qquad (2)$$

The resulting controller is itself parameter dependent - using the available real-time information of the parameter variation. In the grid-based LPV framework

LPVTools Implementation

LPVTools implements LPV controller synthesis for both the LFT-based LPV framework and the grid-based LPV framework. The synthesis functions generate controllers which optimize the performance of the closed-loop system while taking into account the permissible parameter trajectories: $\rho \in \mathcal{P}$, subject to $\overline{\nu} \leq \dot{\rho} \leq \underline{\nu}$.

In the grid-based LPV framework `lpvsyn`, `lpvncfyn`, `lpvmixsyn`, `lpvloopshape`, and `lpvstochsyn` are used to synthesize LPV output-feedback controllers. `lpvsfsyn` is used to synthesize LPV state-feedback controllers, and `lpvestsyn` is used to generate LPV estimators. These functions can be used to generate controllers and estimators to minimize either the induced L_2 norm (based on results by Becker [1] and Wu [2,3], with pole-constrained synthesis based on the derivation by Lee [4]) or the stochastic LPV bound (based on results by Wu [2]). In the LFT-based LPV framework only `lpvsyn` is provided to synthesize LPV output-feedback controllers, and it implements an algorithm which minimizes the induced L_2 norm (based on results by Packard [5], and Apkarian and Gahinet [6]).

The LPV controller synthesis conditions lead to a set of Linear Matrix Inequalities (LMIs) which must be solved in order to generate a controller. These LMIs suffer from similar computational issues to the LPV analysis conditions, and their complexity also grows with $O(2^{n_\rho})$.

References

1. G. Becker, "Quadratic Stability and Performance of Linear Parameter Dependent Systems," Ph.D. Dissertation, University of California, Berkeley, 1993.
2. F. Wu, "Control of Linear Parameter Varying Systems," PhD thesis, University of California, Berkeley, 1993.
3. F. Wu, X. Yang, A. Packard, and G. Becker, "Induced L2 norm control for LPV systems with bounded parameter variation rates," *International Journal of Nonlinear and Robust Control*, vol. 6, pp. 983-998, 1996.
4. L. H. Lee, "Identification and Robust Control of Linear Parameter-Varying Systems," Ph.D. Dissertation, University of California at Berkeley, 1997, doi:10.1.1.55.2269.
5. A. Packard, "Gain scheduling via linear fractional transformations," *Systems and Control Letters*, vol. 22, no. 2, pp. 79-92, 1994.
6. P. Apkarian and P.Gahinet, "A convex characterization of gain-scheduled Hinfinity controllers," *IEEE Transactions on Automatic Control*, vol. 40, no. 5, pp. 853-864, 1995.

LPV Synthesis Commands

LPVTools provides the following functions to design controllers for multiinput-multioutput (MIMO) LPV models:

LPVSYN	Synthesize a LPV controller
LPVNCFSYN	Normalized coprime factor LPV controller synthesis
LPVLOOPSHAPE	LPV loop-shaping synthesis
LPVMIXSYN	LPV mixed-sensitivity synthesis
LPVSFSYN	Synthesize a LPV state-feedback controller
LPVESTSYN	Synthesize a LPV state estimator
LPVSTOCHSYN	Synthesize stochastic LPV controller
LPVSYNOPTIONS	Create options object for LPV synthesis and analysis

Examples and How To

- Tutorial: Creating basis functions
- Tutorial: Synthesis for gridded LPV systems
- Tutorial: Synthesis for LFT LPV systems
- Example: Stochastic LPV control of spinning mass
- Example: LPV control of spinning mass using LFT framework

Concepts

- Permissible Parameter Trajectories
- Stability and Induced Gain
- Characterizing Closed-loop Performance Objectives

LTI synthesis capabilities

Overloaded LTI synthesis function from the Control Systems Toolbox and the Robust Control Toolbox are provided for LPV systems. (e.g. `lqr`, `hinfsyn`, `h2syn`, `loopsyn`, `ncfsyn`, and `mixsyn`). These functions perform the controller synthesis pointwise in the parameter domain of the controller. In each case the resulting controller is not a LPV controller (i.e. one that satisfies the LPV analysis conditions), but a collection of LTI controllers - one for each point.

Published with MATLAB® R2014b

Model Reduction for LPV systems

A primer on model reduction in the LPV framework.

Contents

Introduction

LPVTools provides tools for LPV model reduction. LPV model reduction is different from Linear Time-Invariant (LTI) model reduction techniques which act on a single point, because they perform the model reduction for all values of the scheduling parameter simultaneously. The resulting reduced order model is still a LPV model with consistent state, input and output vectors. If LTI model reduction techniques (e.g. `balreal`) are applied to a LPV model, the resulting model may lose state consistency and the resulting reduced order model is no longer a LPV system. LPVTools provides two functions for LPV model reduction. `lpvbalancmr` performs balanced trunctation, and provides the option of weighting different frequency bands in the model reduction to emphasize accuracy for some dynamics while de-emphasizing others.. However it is restricted to stable LPV systems. `lpvncfmr` performs a contractive coprime factorization of a LPV system, and can handle unstable LPV systems.

Further Reading

1. G. D. Wood, "Control of parameter-dependent mechanical systems," Ph.D. Dissertation, University of Cambridge, 1995.
2. G. D. Wood, P. J. Goddard, and K. Glover, "Approximation of linear parameter-varying systems," *IEEE Conference on Decision and Control*, Vol. 1, pp 406-411, 1996.
3. R. Widowati, R. Bambang, R. Sagari, S. M. and Nababan, “Model reduction for unstable LPV system based on coprime factorizations and singular perturbation,” *5th Asian Control Conference*, Vol. 2, pp. 963-970, Melbourne, Australia, 2004.

LPV Model Reduction Commands

LPVGRAM	Compute Gramians for PSS objects.
LPVBALREAL	Perform Gramian-based balancing for PSS objects.
LPVBALANCMR	Balanced truncation model reduction.
LPVNCFMR	Balanced truncation model reduction through contractive coprime factorization.

Examples and How To

- LTI Model Reduction
- Model Reduction for a stable LPV system
- Model Reduction for an unstable LPV system

Concepts

- State Consistency

Published with MATLAB® R2014b

LPV Model Reduction for a Stable LPV System:

Consider a third-order parameter dependent system:

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \dot{x}_{act} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -\omega_n^2 & -2\zeta(\rho)\omega_n & 1 \\ 0 & 0 & -100 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_{act} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 100 \end{bmatrix} u \qquad (1)$$
$$y = x_1$$

where the parameter ρ lies in the interval $[1, 5]$, and the coefficients $\zeta(\rho)$ and ω_n are defined as:

$$\zeta(\rho) = \frac{\sqrt{6-p}}{10} \qquad (2)$$

$$\omega_n = 5 \qquad (3)$$

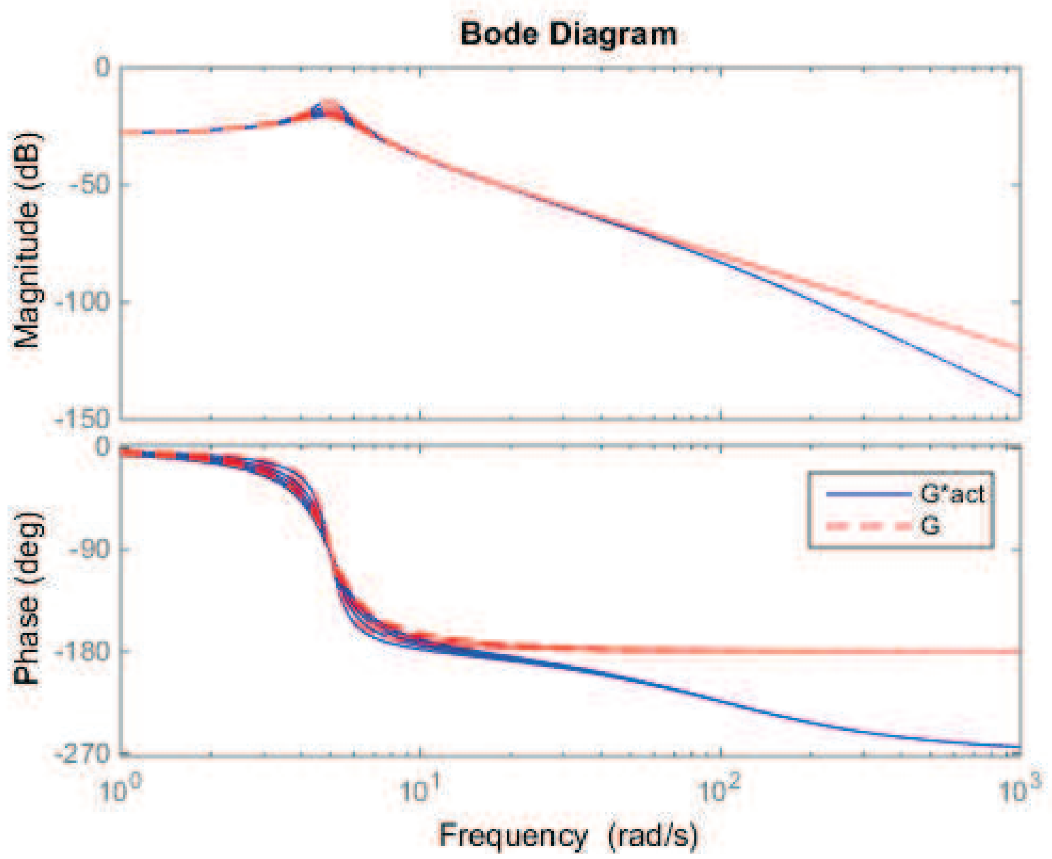
We note that the system consists of a second order system, with damping coefficient $\zeta(\rho)$ and natural frequency ω_n, that is in series with a first-order actuator at its input. The actuator has a pole at 100 rad/s, which is two orders of magnitude higher than the natural frequency of the second-order system. Hence, if the dynamics at low frequency are the main object of interest, it is possible to remove the actuator state from this 3 state model, with minimal effect in the frequency band where the second-order dynamics take place. Lets do this using LPVTools.

```
% Define the time-varying rho parameter as a gridded real parameter:
p = pgrid('p',1:5);

% Define the second-order system:
zet = sqrt(6-p)/10;
wn = 5;
G = ss([0 1;-wn^2 -2*zet*wn],[0;1],[1 0],0);

% Define the first order actuator:
act = ss(-100,100,1,0);

% Define the plant model which consists of the the second-order system and
% the actuator in series:
sys = G*act
```

```
PSS with 3 States, 1 Outputs, 1 Inputs, Continuous System.
The PSS consists of the following blocks:
  p: Gridded real, 5 points in [1,5], rate bounds [-Inf,Inf].
```

Lets compare the frequency response of `sys` and G at the five grid points: $\rho = [1, 2, 3, 4, 5]$

```
freq = linspace(1,1e3,5000);
bode(sys,'b',freq)
hold on
bode(G,'r--',freq)
legend('G*act','G')
hold off
```



We note the actuator pole kicking in at 100 rad/s, and that its effect on the second order dynamics is negligable. Hence, it should be safe to remove 1 state from the model if we are only interested in the system's dynamics at low frequency.

`lpvbalancmr` will compute a balanced realization of the LPV system `sys` and then remove those states that contribute least to its input/output behaviour. `sys` has 3 states, we will call on `lpvbalancmr` to remove 1 state and generate a balanced realization with only 2 states.

```
% The first input to |lpvbalancmr| is the system to be reduced. The second input
% is the desired state order of the output:
[sys_red,info] = lpvbalancmr(sys,2);
```

`sys_red` is the 2 state reduced-order model:

```
sys_red
```

```
PSS with 2 States, 1 Outputs, 1 Inputs, Continuous System.
The PSS consists of the following blocks:
  p: Gridded real, 5 points in [1,5], rate bounds [-Inf,Inf].
```

`info` stores results for the model reduction. Lets look at the relative size of the Hankel singular values associated with the states in the balanced version of `sys`:

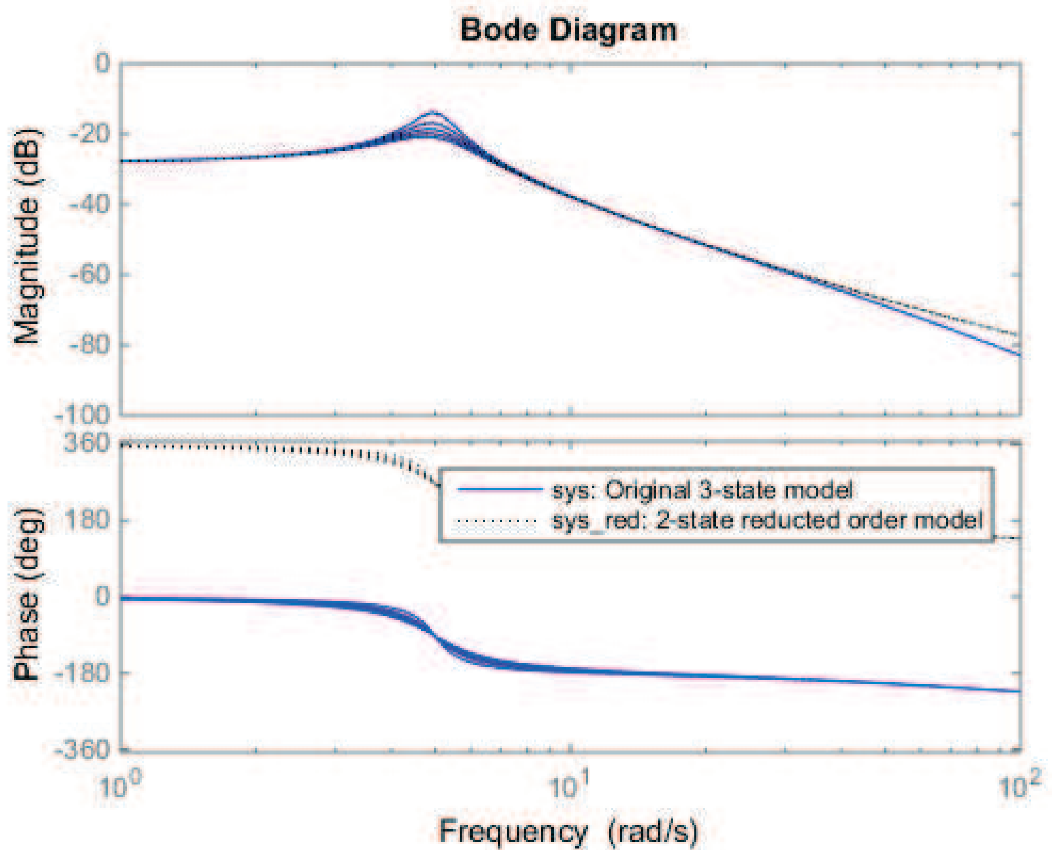
```
info.StabSV
```

```
ans =
    0.1115
    0.0915
    0.0006
```

We see that one of the states has a Hankel Singular value that is two orders of magnitude smaller than the smallest Hankel Singular value of the other two. This indicates that it removing this state will have a minimal effect on the accruacy of the resulting reduced order model.

Lets compare the frequency response of the original three state system `sys`, and the reduced order second-order system `sys_red`.

```
freq = linspace(1,1e2,5000);
bode(sys,'b',freq)
hold on;
bode(sys_red,'k:',freq)
legend('sys: Original 3-state model',...
'sys_red: 2-state reducted order model','location','northeast')
```



We note that the frequency response in of the original three state system and the reduced order system is identical up to approximatly 30 rad/s.

References

1. G. D. Wood, "Control of parameter-dependent mechanical systems," Ph.D. Dissertation, University of Cambridge, 1995.
2. G. D. Wood, P. J. Goddard, and K. Glover, "Approximation of linear parameter-varying systems," *IEEE Conference on Decision and Control*, Vol. 1, pp 406-411, 1996.
3. R. Widowati, R. Bambang, R. Sagari, S. M. and Nababan, “Model reduction for unstable LPV system based on coprime factorizations and singular perturbation,” *5th Asian Control Conference*, Vol. 2, pp. 963-970, Melbourne, Australia, 2004.

LPV Model Reduction for a Unstable LPV System:

Contents

LPV Model Reduction for a Unstable LPV System

Consider a third-order parameter dependent system:

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \dot{x}_{act} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ -\omega_n^2 & -2\zeta(\rho)\omega_n & 1 \\ 0 & 0 & -100 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_{act} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 100 \end{bmatrix} u \qquad (1)$$
$$y \qquad = x_1$$

where the parameter ρ lies in the interval $[1, 5]$, and the coefficients $\zeta(\rho)$ and ω_n are defined as:

$$\zeta(\rho) = \frac{2-p}{10} \qquad (2)$$

$$\omega_n = 5 \qquad (3)$$

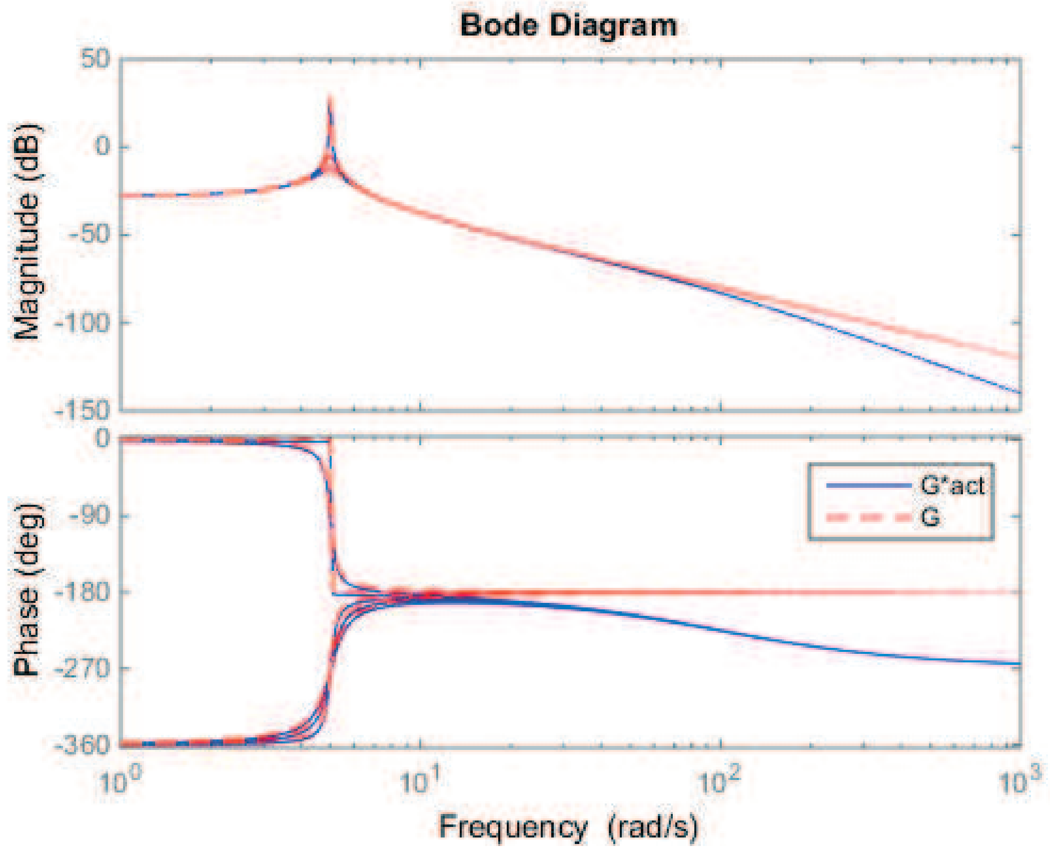
We note that the system consists of a second order system, with damping coefficient $\zeta(\rho)$ and natural frequency ω_n, that is in series with a first-order actuator at its input. The second order system is unstable for $\rho > 2$ and marginally stable for $\rho = 2$. The actuator has a pole at 100 rad/s, which is two orders of magnitude higher than the natural frequency of the second-order system. Hence, if the unstable dynamics at low frequency are the main object of interest, it is possible to remove the actuator state from this 3 state model, with minimal effect in the frequency band where the unstable second-order dynamics take place. Lets do this using LPVTools.

```
% Define the time-varying rho parameter as a gridded real parameter:
p = pgrid('p',1:5);

% Define the second-order system:
zet = (2-p)/30;
wn = 5;
G = ss([0 1;-wn^2 -2*zet*wn],[0;1],[1 0],0);

% Define the first order actuator:
act = ss(-100,100,1,0);

% Define the plant model which consists of the the second-order system and
% the actuator in series:
sys = G*act
```

```
PSS with 3 States, 1 Outputs, 1 Inputs, Continuous System.
The PSS consists of the following blocks:
  p: Gridded real, 5 points in [1,5], rate bounds [-Inf,Inf].
```

Lets compare the frequency response of `sys` and `G` at the five grid points: $\rho = [1,2,3,4,5]$

```
freq = linspace(1,1e3,10000);
bode(sys,'b',freq)
hold on
bode(G,'r--',freq)
legend('G*act','G')
hold off
```



We note the actuator pole kicking in at 100 rad/s, and that its effect on the second order dynamics is negligable. Hence, it should be safe to remove 1 state from the model if we are only interested in the system's dynamics at low frequency.

`lpvncfmr` will compute a contractive coprime factorization of the LPV system `sys`, which removes those states that contribute least to its input/output behaviour. `sys` has 3 states, we will call on `lpvncfmr` to % remove 1 state and generate a realization with only 2 states. Note that the function `lpvbalancmr` can not be used in this case, because the LPV system is unstable and it requires a stable LPV system. `lpvncfmr` can handle both stable and unstable LPV systems.

```
% The first input to |lpvncfmr| is the system to be reduced. The second
% input is the desired state order of the output:
sys_red = lpvncfmr(sys,2);
```

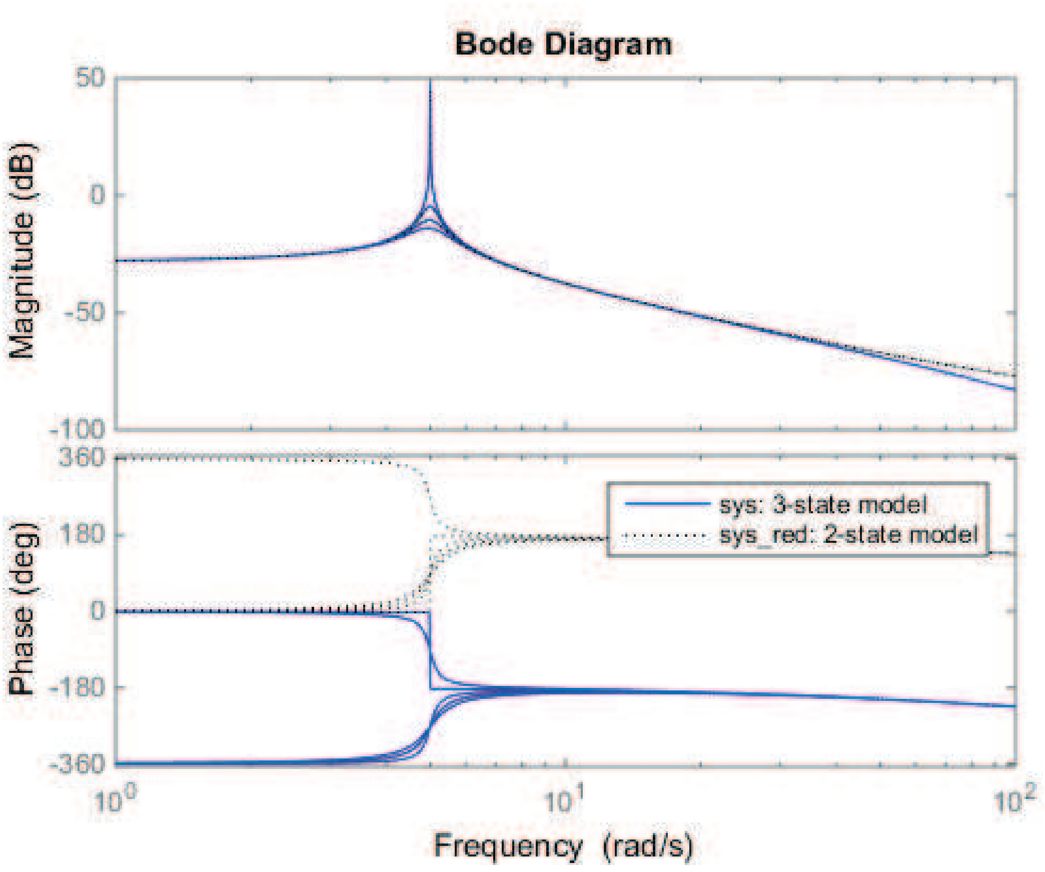
`sys_red` is the 2 state reduced-order model:

```
sys_red
```

```
PSS with 2 States, 1 Outputs, 1 Inputs, Continuous System.
The PSS consists of the following blocks:
  p: Gridded real, 5 points in [1,5], rate bounds [-Inf,Inf].
```

Lets compare the frequency response of the original three state system `sys`, and the reduced order second-order system `sys_red`.

```
freq = linspace(1,1e2,5000);
bode(sys,'b',freq)
hold on;
bode(sys_red,'k:',freq)
legend('sys: 3-state model',...
'sys_red: 2-state model','location','northeast')
```



We note that the frequency response in of the original three state system and the reduced order system is identical up to approximatly 30 rad/s.

References

1. G. D. Wood, "Control of parameter-dependent mechanical systems," Ph.D. Dissertation, University of Cambridge, 1995.
2. G. D. Wood, P. J. Goddard, and K. Glover, "Approximation of linear parameter-varying systems," *IEEE Conference on*

Decision and Control, Vol. 1, pp 406-411, 1996.

3. R. Widowati, R. Bambang, R. Sagari, S. M. and Nababan, “Model reduction for unstable LPV system based on coprime factorizations and singular perturbation,” *5th Asian Control Conference*, Vol. 2, pp. 963-970, Melbourne, Australia, 2004.

Published with MATLAB® R2014b

Simulation of LPV systems

A primer on simulation in LPVTools.

Contents

Introduction

LPVTools provides a set of tools to perform time-domain simulation of LPV systems. The tools can be split into two parts: First, there are overloaded versions of Linear Time-Invariant (LTI) simulation tools from the Control Systems Toolbox (`lsim`, `step`, `impulse`, `initial`). These functions can be used to evaluate the pointwise behaviour of the LPV system when the scheduling parameter (ρ) is held constant. Second, there are LPV simulation tools that enable simulation of the LPV system when the parameter is allowed to vary with time. These LPV simulation tools are able to capture the time-varying nature of the LPV system's dynamics, and allow the system's behaviour to be evaluated for different parameter trajectories.

LPV Simulation Commands

LPVLSIM	Simulate parameter dependent time response of a LPV system.
LPVSTEP	Simulate parameter dependent step response of a LPV system.
LPVINITIAL	Simulate initial conditions response of a LPV system.
LPVIMPULSE	Simulate parameter dependent impulse response of a LPV system.

Examples and How To

- Tutorial: Simulate a Quasi-LPV System
- Tutorial: Analysis of a grid-based LPV system
- Tutorial: Control of a grid-based LPV system
- Example: Stochastic LPV control of spinning mass
- Example: LPV control of spinning mass using LFT framework

Concepts

- Permissible Parameter Trajectories
- Quasi-LPV Models

Published with MATLAB® R2014b

Simulating Quasi-LPV Systems

Contents

Introduction

LPVTools provides command line simulation tools that enable simulation of quasi-LPV systems, i.e. systems in which one of the states is also the scheduling parameter. An accurate simulation of a quasi-LPV system requires that the parameter trajectory during simulation is a function of the system state. This is achieved using Function Handles.

The syntax for LPV simulations is:

```
lpvlsim(G,PTRAJ,UIN,TIN,X0)
```

where `G` is the system to be simulated, `PTRAJ` is a structure that defines the parameter trajectory, `UIN` is the input to the system, `TIN` a vector of simulation time values, and `X0` is the initial value of the system.

The LPV simulation command requires the user to specify the parameter trajectory in the structure `PTRAJ`. To use function handles to specify the parameter trajectory, `PTRAJ` must be specified as a structure with the field \|PTRAJ.Functions| that specifies function handles for evaluating the parameters $\rho = f(x,u,t)$. A second field `PTRAJ.Names` provides a cell array list of the parameter names corresponding to each function handle. The following code provides an example demonstrating the simulation of a simple nonlinear system using the quasi-LPV command line simulation approach:

Example

Create system: $\dot{x} = -(1+\rho)x + u$; $y = x$;

```
rho = pgrid('rho',linspace(0,50,100));
sys = ss(-1-rho,1,1,0)
```

```
PSS with 1 States, 1 Outputs, 1 Inputs, Continuous System.
The PSS consists of the following blocks:
  rho: Gridded real, 100 points in [0,50], rate bounds [-Inf,Inf].
```

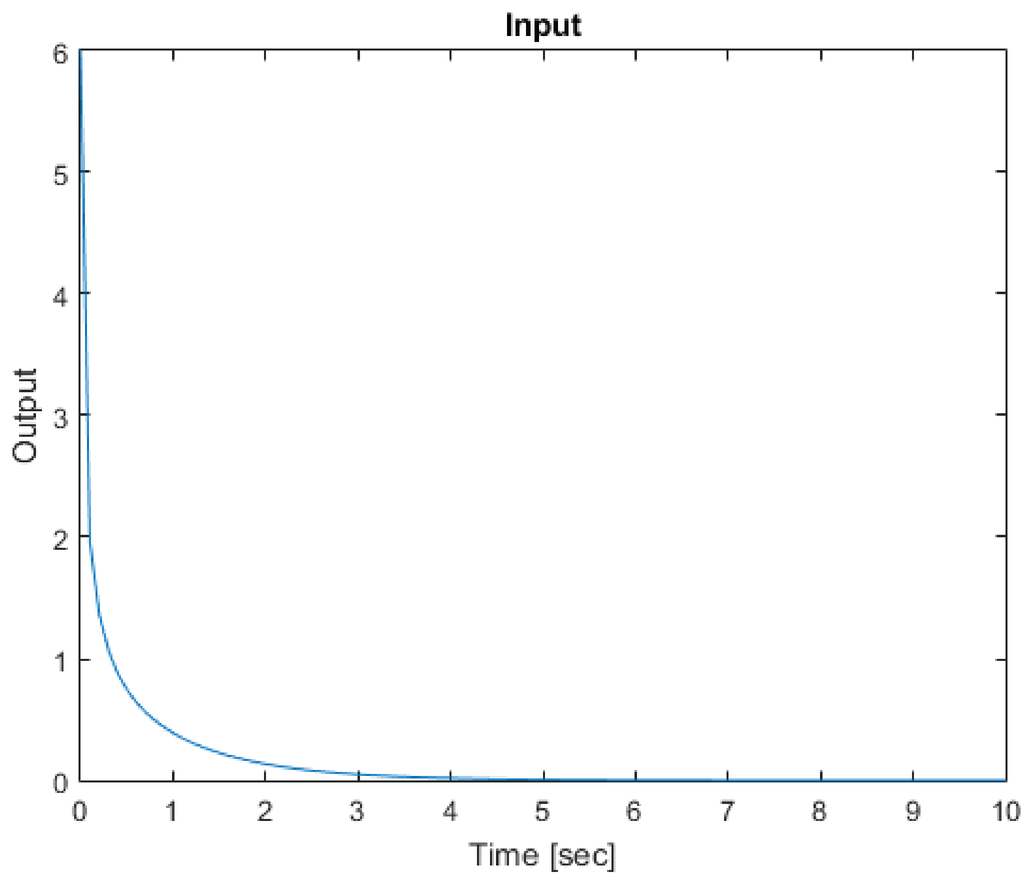
Make $\rho(x,u,t) = x^2$ such that the nonlinear system is $\dot{x} = -x - x^3 + u$

```
pFHN.Functions = @(x,u,t) x^2;
pFHN.Names = {'rho'}
```

```
pFHN =
    Functions: @(x,u,t)x^2
        Names: {'rho'}
```

Simulate from initial condition x0=6; u=0 Response starts very fast ($\dot{x} = -37x$), becomes slow ($\dot{x} = -x$)

```
Nt=100;
tVec = linspace(0,10,Nt);
uVec = zeros(Nt,1);
x0 = 6;
lpvlsim(sys,pFHN,uVec,tVec,x0);
```



The plot shows the response generated for this quasi-LPV example. It is important to emphasize that this simulation captures the nonlinear dependence $\rho(x) = x^2$. Hence this function-handle approach enables simulation of nonlinear systems represented in the quasi-LPV form. The functions `lpvstep`, `lpvimpulse`, and `lpvinitial` were also extended to allow for quasi-LPV simulations.

It should be noted that the Simulink blocks in LPVTools naturally allow for quasi-LPV simulations $\rho(x,u,t)$ because the parameter trajectory at the ``Parameter'' input port can be generated via Simulink blocks operating on any signal in the model.

Published with MATLAB® R2015a

Tutorials

The following tutorials demonstrate some of the key features of LPVTools.

Examples

- Stochastic LPV control of spinning mass
- LPV control of spinning mass using LFT framework

Tutorials

- Constructing grid-based LPV models
- Modeling and Control of LFT LPV Systems
- Conversion between LFT and LPV models
- Creating a grid-based LPV model from analytical linearization
- Creating a grid-based LPV model from a nonlinear model
- Creating basis functions
- Analysis and simulation of gridded LPV systems
- Worst-case LPV analysis using lpvwcgain
- Synthesis for gridded LPV systems
- Simulate a Quasi-LPV System
- Model Reduction for a stable LPV system
- Model Reduction for an unstable LPV system

Control Design for Stochastic LPV Systems

```
The following example from Wu [1] describes a control design for a
mechanical system in the stochastic LPV framework.
```

Contents

Problem Statement: Coupled Spinning Disks

A pair of rotating disks is shown in Figure 1. The slotted disks rotate at rates Ω_1 rad/s and Ω_2 rad/s. The disks contain masses M_1 and M_2, which can move in the horizontal plane, and slide in the radial direction The two masses are connected by a wire, which can transmit force in both compression and tension. The wire acts as a spring with spring constant k. A coupler and pulley system handles the wire routing between the two disks. The effect of friction due to the sliding motion of each mass is modeled by a damping coefficient b.

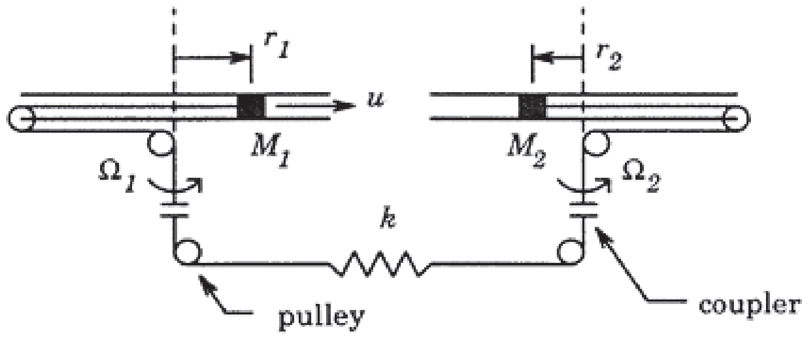


Figure 1: A pair of spinning disk with masses M_1 and M_2 [1].

The problem is to control the position of the two masses: r_1 for mass M_1 and r_2 for M_2. The control input is a radial force f_u acting on mass M_1, while there is a radial disturbance force acting on each mass: f_1 acting on M_1 and f_2 acting on M_2. The equations of motion for this problem are:

$$\begin{aligned} M_1\left(\ddot{r}_1 - \Omega_1^2 r_1\right) + b\dot{r}_1 + k(r_1+r_2) &= f_u + f_1 \\ M_2\left(\ddot{r}_2 - \Omega_2^2 r_2\right) + b\dot{r}_2 + k(r_1+r_2) &= f_2 \end{aligned} \qquad (1)$$

where:

- $M_1 = 1$ kg is the mass of the sliding mass in disk 1.
- $M_2 = 0.5$ kg is the mass of the sliding mass in disk 2.
- $b = 1$ kg is the damping coefficient due to fricition (kg/sec).

- $k = 200$ N/m is the spring constant of the wire connecting the masses.
- $r_1(t)$ is the position of the M_1 sliding mass relative to the center of disk 1 (m).
- $r_2(t)$ is the position of the M_2 sliding mass relative to the center of disk 2 (m).
- $\Omega_1(t)$ is the rotational rate of disk 1 (rad/s).
- $\Omega_2(t)$ is the rotational rate of disk 2 (rad/s).
- f_u is the control force acting radially on mass M_1 (N).
- f_1 is the disturbance force acting radially on mass M_1 (N).
- f_2 is the disturbance force acting radially on mass M_2 (N).

The rotational rates of the spinning disk are allowed to vary: $\Omega_1 \in [0,3]$ rad/s and $\Omega_2 \in [0,5]$ The rotational rates are not known in advance but are measured and available for control design.

The objective of the control design is to command the radial position of mass M_2. Note that the control input is applied to mass M_1, and is transmitted to mass M_2 through the wire connecting the two disks.

LPV System Formulation:

The system in Equation (1) is already processing the inputs and outputs linearly, and the only nonlinear elements in Equation (1) are the rotation rates Ω_1 and Ω_2. Lets choose the rotation rates as the scheduling parameters and transform the system into a LPV model. Define $\rho_1 = \Omega_1^2$ and $\rho_2 = \Omega_2^2$, such that $\rho_1 \in [0,25]$ and $\rho_2 \in [09]$. Furthermore, let

$$\begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix} = \begin{bmatrix} r_1(t) \\ r_2(t) \\ \dot{r}_1(t) \\ \dot{r}_2(t) \end{bmatrix} \qquad (2)$$

and

$$\begin{bmatrix} u(t) \\ d_1(t) \\ d_2(t) \end{bmatrix} = \begin{bmatrix} f_u(t) \\ f_1(t) \\ f_2(t) \end{bmatrix} \qquad (3)$$

And rewrite the system in Equation (1) as the LPV system:

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \dot{x}_3(t) \\ \dot{x}_4(t) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \rho_1 - \frac{k}{M_1} & -\frac{k}{M_1} & -\frac{b}{M_1} & 0 \\ -\frac{k}{M_2} & \rho_2 - \frac{k}{M_2} & 0 & -\frac{b}{M_2} \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{1}{M_1} & \frac{0.1}{M_1} & 0 \\ 0 & 0 & \frac{0.1}{M_2} \end{bmatrix} \begin{bmatrix} f_u(t) \\ f_1(t) \\ f_2(t) \end{bmatrix} \qquad (4)$$

$$y(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix} \qquad (5)$$

The following commands will generate the LPV system in Equations (5)-(6)

```
% Define system parameters
```

```
m1 = 1;
m2 = 0.5;
k = 200;
b = 1;

% Define timve-varying parameters
rho1 = pgrid('rho1',[0 4.5 9]);
rho2 = pgrid('rho2',[0 12.5 25]);

% Define system matrices:
A = [   0              0             1       0    ; ...
        0              0             0       1    ; ...
    rho1-k/m1    -k/m1        -b/m1     0    ; ...
      -k/m2         rho2-k/m2   0     -b/m2  ];

B = [    0      0       0     ; ...
         0      0       0     ; ...
      1/m1   .1/m1      0     ; ...
         0      0     .1/m2 ];

C = [ 0 1 0 0];

D = [0 0 0];

% Define the LPV system:
sys = ss(A,B,C,D)
```

```
PSS with 4 States, 1 Outputs, 3 Inputs, Continuous System.
The PSS consists of the following blocks:
  rho1: Gridded real, 3 points in [0,9], rate bounds [-Inf,Inf].
  rho2: Gridded real, 3 points in [0,25], rate bounds [-Inf,Inf].
```

LPV Control Design

We will design an LPV controller that optimizes the stochastic bound

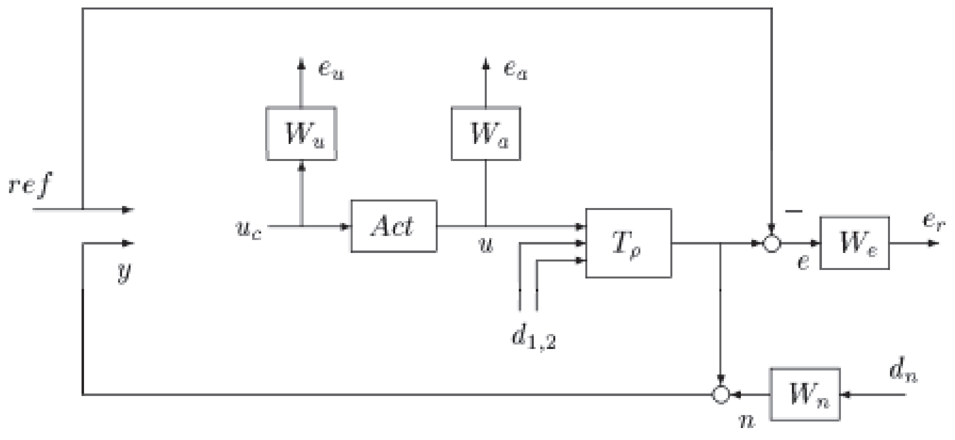


Figure 2: The weighted design interconnection [1].

The weighted interconnection shown in Figure 2 will be used for synthesis. With appropriate weights selected, this interconnection, defines the desired performance of the LPV control law. The controller will be synthesized to yield optimized performance in terms of this specification (additional details on formulating design problems in this way are provided in the Robust Control Toolbox's documentation on characterizing closed-loop performance objectives.)

The weights for the interconnection are chosen as:

```
% Define weights:
We = tf(2,[1 0.04]);
Wu = 1/50;
Wa = 0.00001;
Wn = tf([1,0.4],[0.01 400]);
act = ss(-100,100,1,0);
```

and we can form the weighted interconnection shown in Figure 2:

```
% Form synthesis interconnection:
systemnames = 'sys act We Wu Wa Wn';
inputvar = '[ref;d{2};dn;u]';
outputvar = '[Wu;Wa;We;ref;sys+Wn]';
input_to_sys = '[act;d]';
input_to_act = '[u]';
input_to_Wa = '[act]';
input_to_Wu = '[u]';
input_to_Wn = '[dn]';
input_to_We = '[sys-ref]';
G = sysic;
```

We will use the function `lpvstochsyn` to synthesize a LPV controller which minimizes the <..\..\Concepts\StabilityAndInducedGain\html\StabilityAndInducedGain.html stochastic LPV bound > of the weighted interconnection G in Figure 2. This synthesis will assume that the rate of variation in ρ_1 and ρ_2 are unbounded, i.e. there is no limit to fast the parameters can change with time.

```
% Synthesize two-degree of freedom controller.
nmeas = 2;                                      % # of measurements
ncont = 1;                                      % # of controls
[Knr,Gamma,Info] = lpvstochsyn(G,nmeas,ncont);
```

The control design is successfull, and the controller `Knr` is guarenteed to achieve a stochastic LPV bound that is less than or equal to `Gamma`

```
Gamma
```

```
PMAT with 1 rows and 1 columns.

Gamma =
    1.5998
```

Computing a Tighter Bound on the Stochastic LPV Bound

The value reported in `Gamma` is an only an upper bound on the stochastic LPV bound. We will now compute a tighter bound on the stochastic LPV bound achieved by `Knr`. We will insert `Knr` into the weighted interconnection and use this closed-loop weighted interconnection to compute the H2 norm for the Linear Time-Invariant (LTI) system obtained by holding the parameters constant at each grid point in the domain of the LPV system: $[\rho_1, \rho_2] = [0, 4.5, 9] \times [0, 12.5, 25]$. The stochastic LPV bound will always be greater or equal to the largest H2 norm on the grid of parameter values. Hence, the largest H2 norm we compute will provide a lower bound on the stochastic LPV bound.

```
% Insert Knr into the weighted interconnection:
CLICnr = lft(G,Knr);

% Compute the H2 norm for the
LTI_CLnorm = norm(CLICnr,2);

% extract the largest H2 norm on the grid of parameter values:
lpvmax(LTI_CLnorm)
```

```
PMAT with 1 rows and 1 columns.

ans =
    1.5506
```

The results indicate that the stochastic LPV bound achieved by `Knr` is between 1.5506 and 1.5998.

Evaluating Pointwise Performance

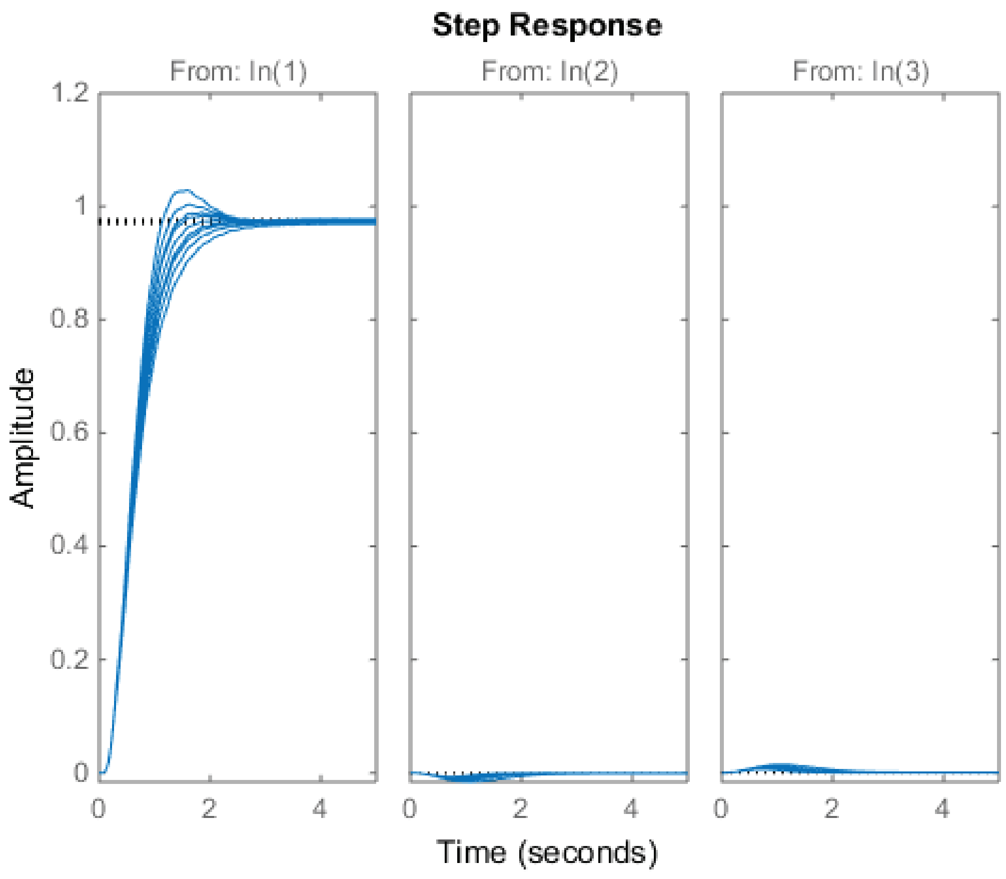
Lets look at the closed-loop response for the original LPV system in the loop with the controller `Knr`:

```
% Form closed-loop sytem
systemnames = 'sys act Knr';
inputvar = '[r;d{2}]';
outputvar = '[sys]';
input_to_sys = '[act;d]';
input_to_act = '[Knr]';
input_to_Knr = '[r;sys]';
CL = sysic;
```

We will start by applying LTI analysis techniques to evaluate the performance of the LPV controller. We will evalaluate the LTI closed-loop system obtained at each grid point in the domain $[\rho_1, \rho_2] = [0, 4.5, 9] \times [0, 12.5, 25]$, when the parameter is held constant.

We can start by studying the pointwise step response. The Control System Toolbox's command `step` is overloaded to work with the `pss` object. It will compute the step response at each point in the grid, and plot them together on the same plot:

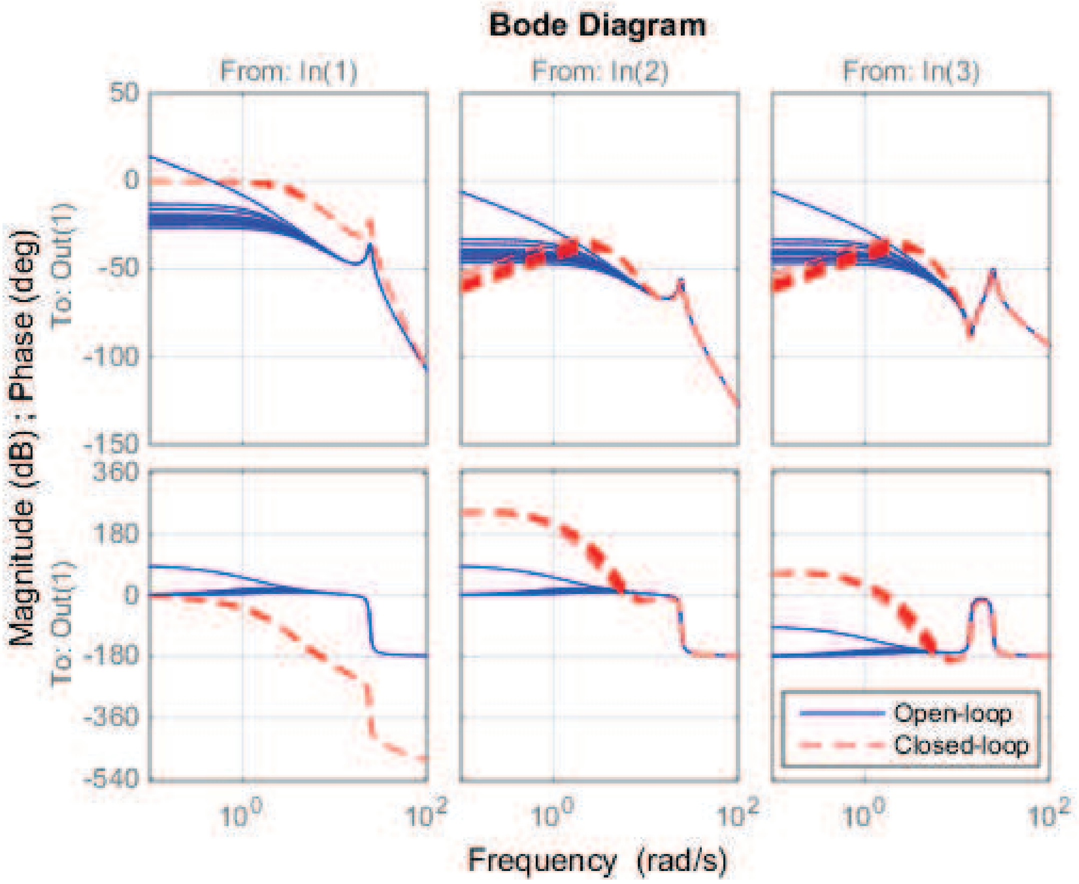
```
step(CL)
```



The controller achives good tracking with minimal overshoot, approximatly 2.5 sec settling time and less than 3% steady-state tracking error. Furthermore, the effect of the disturbances is minimal.

Similarly we can create pointwise frequency responses using the `bode` command. Lets compare the pointwise frequency response of the open- and closed-loop system at frequencies between 0.1 to 20 rad/s

```
bode(sys,{0.1,100},'b')
hold on
bode(CL,{0.1,100},'r--')
legend('Open-loop','Closed-loop','location','best')
grid minor
hold off
```

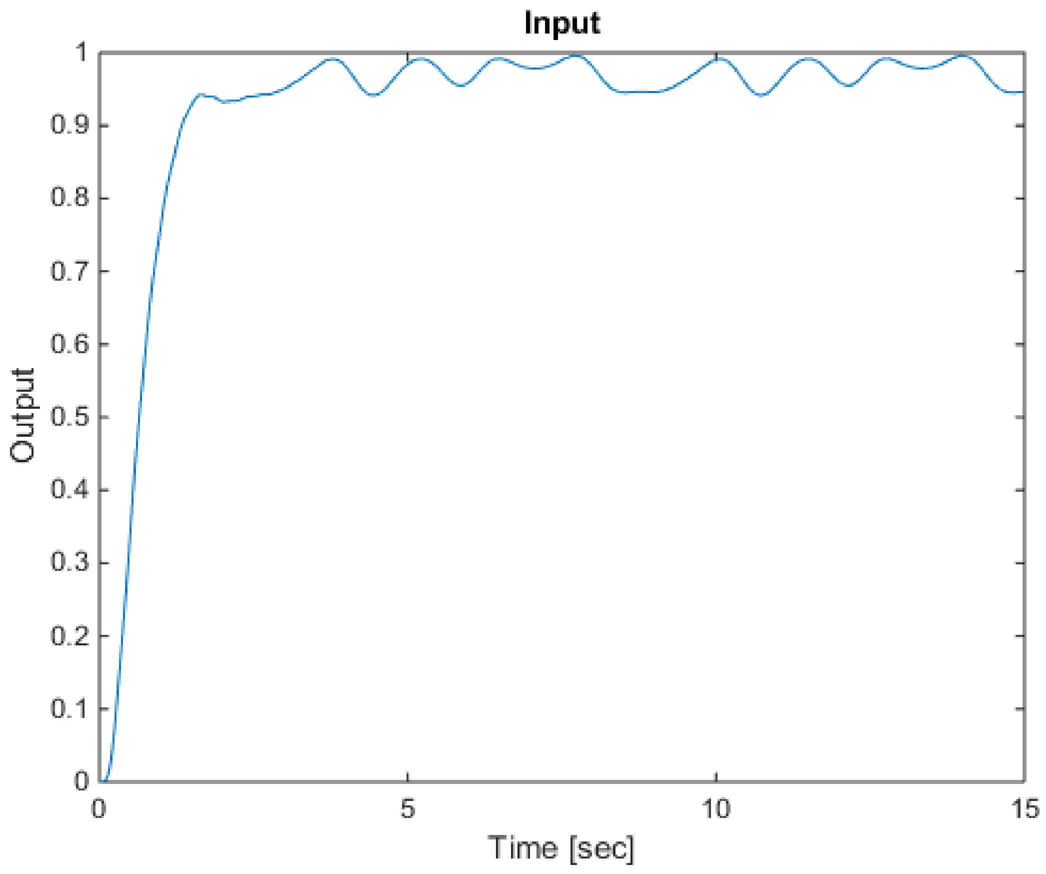


The closed-loop system has a bandwidth of approximatly 3 rad/s.

Evaluating LPV Time-Domain Performance

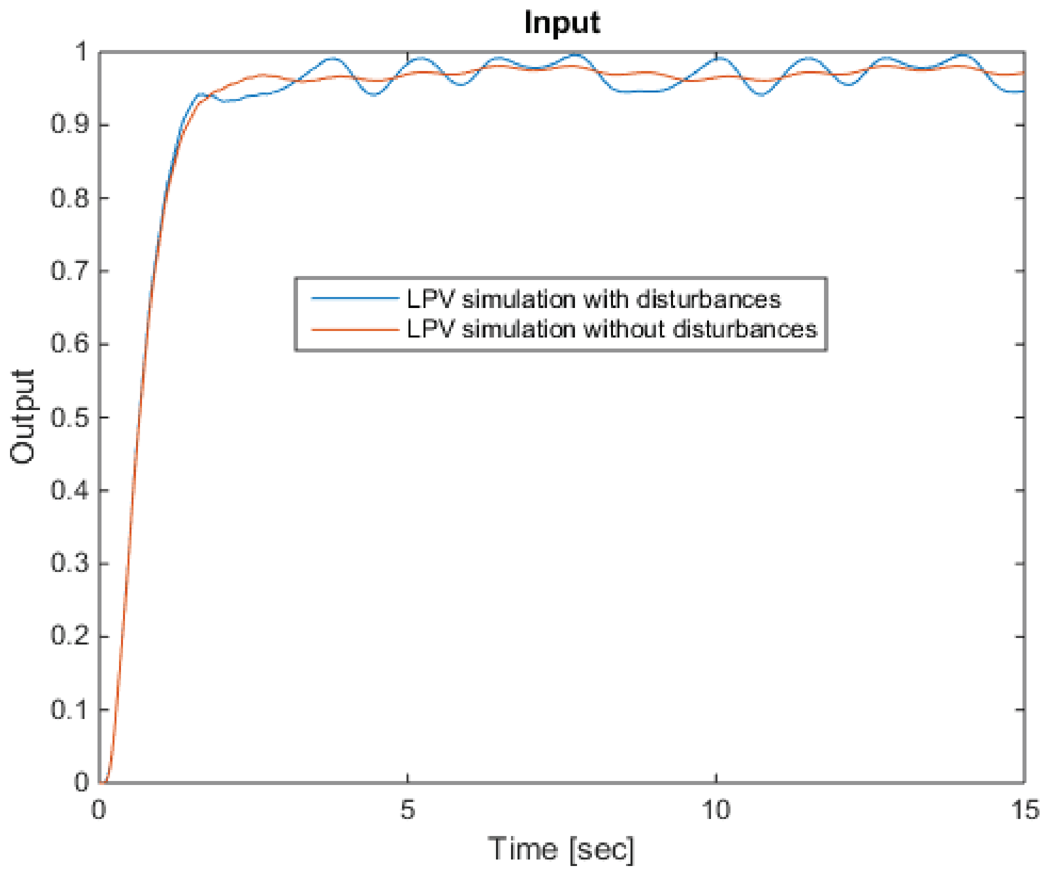
We will use the LPV simulation capabilities to inspect the performance of `Knr`. The parameters ρ_1 and ρ_2 will be made to vary with time: $\rho_1(t) = sin(t) + 1.5$ and $\rho_2(t) = 0.5 * cos(5 * t) + 3$, while the system tracks a unit step response and rejects disturbances $d_1 = cos(3t) + sin(5t) + cos(8t)$ and $d_2 = cos(t) + sin(2t) + cos(4t)$.

```
% Define the inputs to the system:
t = [0:0.01:15]';
u = ones(size(t));
d1 = cos(3*t)+sin(5*t)+cos(8*t);
d2 = cos(t)+sin(2*t)+cos(4*t);


% Define the trajectories of the parameters:
ptraj.time = t;
ptraj.rho1 = sin(t)+1.5;
ptraj.rho2 = .5*cos(5*t)+3;

% Perform LPV simulation:
lpvlsim(CL,ptraj,[u,d1,d2],t);
```



Lets evaluate the impact that the disturbances had on the response by repeating the simulation without disturbances:

```
lpvlsim(CL,ptraj,[u,d1,d2],t);
hold on
lpvstep(CL(:,1),ptraj);
legend('LPV simulation with disturbances',...
       'LPV simulation without disturbances',...
       'location','best')
```



The error due to the distrubances is on the order of 5%, and they are confined to an approximately 1-2 Hz oscillation about the nominal. This frequency range is incidentally where the system's pointwise frequency response analysis indicated that disturbances would have the greatest effect on the output signal.

References

1. F. Wu, "Control of Linear Parameter Varying Systems," Ph.D. dissertation, Department of Mechanical Engineering, University of California at Berkeley, CA, May 1995.

Published with MATLAB® R2014b

Modeling and Control of a LFT-based LPV System

Contents

Problem Statement: Coupled Spinning Disks

A pair of rotating disks is shown in Figure 1. The slotted disks rotate at rates Ω_1 rad/s and Ω_2 rad/s. The disks contain masses M_1 and M_2, which can move in the horizontal plane, and slide in the radial direction The two masses are connected by a wire, which can transmitt force in both compression and tension. The wire acts as a spring with spring constant k. A coupler and pulley system handles the wire routing between the two disks. The effect of friction due to the sliding motion of each mass is modeled by a damping coefficient b.

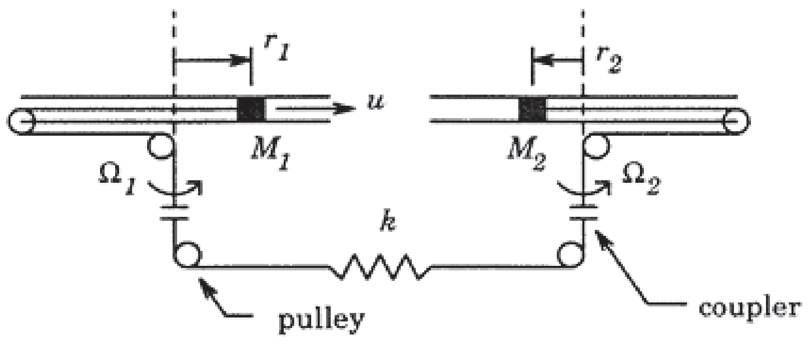


Figure 1: A pair of spinning disk with masses M_1 and M_2 [1].

The problem is to control the position of the two masses: r_1 for mass M_1 and r_2 for M_2. The control input is a radial force f_u acting on mass M_1, while there is a radial disturbance force acting on each mass: f_1 acting on M_1 and f_2 acting on M_2. The equations of motion for this problem are:

$$\begin{aligned} M_1\left(\ddot{r}_1 - \Omega_1^2 r_1\right) + b\dot{r}_1 + k(r_1+r_2) = f_u + f_1 \\ M_2\left(\ddot{r}_2 - \Omega_2^2 r_2\right) + b\dot{r}_2 + k(r_1+r_2) = f_2 \end{aligned} \qquad (1)$$

where:

- $M_1 = 1$ kg is the mass of the sliding mass in disk 1.
- $M_2 = 0.5$ kg is the mass of the sliding mass in disk 2.
- $b = 1$ kg is the damping coefficient due to fricition (kg/sec).
- $k = 200$ N/m is the spring constant of the wire connecting the masses.
- $r_1(t)$ is the position of the M_1 sliding mass relative to the center of disk 1 (m).
- $r_2(t)$ is the position of the M_2 sliding mass relative to the center of disk 2 (m).

- $\Omega_1(t)$ is the rotational rate of disk 1 (rad/s).
- $\Omega_2(t)$ is the rotational rate of disk 2 (rad/s).
- f_u is the control force acting radially on mass M_1 (N).
- f_1 is the disturbance force acting radially on mass M_1 (N).
- f_2 is the disturbance force acting radially on mass M_2 (N).

The rotational rates of the spinning disk are allowed to vary: $\Omega_1 \in [0,3]$ rad/s and $\Omega_2 \in [0,5]$ The rotational rates are not known in advance but are measured and available for control design.

The objective of the control design is to command the radial position of mass M_2. Note that the control input is applied to mass M_1, and is transmitted to mass M_2 through the wire connecting the two disks.

LFT-Based LPV System Formulation

The system in Equation (1) is already processing the inputs and outputs linearly, and the only nonlinear elements in Equation (1) are the rotation rates Ω_1 and Ω_2. Lets choose the rotation rates as the scheduling parameters and transform the system into a LFT-based LPV model. Define $\rho_1 = \Omega_1^2$ and $\rho_2 = \Omega_2^2$, such that $\rho_1 \in [0,25]$ and $\rho_2 \in [09]$. Furthermore, let

$$\begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix} = \begin{bmatrix} r_1(t) \\ r_2(t) \\ \dot{r}_1(t) \\ \dot{r}_2(t) \end{bmatrix} \qquad (2)$$

and

$$\begin{bmatrix} u(t) \\ d_1(t) \\ d_2(t) \end{bmatrix} = \begin{bmatrix} f_u(t) \\ f_1(t) \\ f_2(t) \end{bmatrix} \qquad (3)$$

And rewrite the system in Equation (1) as the LPV system:

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \dot{x}_3(t) \\ \dot{x}_4(t) \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ \rho_1 - \frac{k}{M_1} & -\frac{k}{M_1} & -\frac{b}{M_1} & 0 \\ -\frac{k}{M_2} & \rho_2 - \frac{k}{M_2} & 0 & -\frac{b}{M_2} \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{1}{M_1} & \frac{0.1}{M_1} & 0 \\ 0 & 0 & \frac{0.1}{M_2} \end{bmatrix} \begin{bmatrix} f_u(t) \\ f_1(t) \\ f_2(t) \end{bmatrix} \qquad (4)$$

$$y(t) = \begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix} \qquad (5)$$

The following commands will generate the LPV system in Equations (5)-(6)

```
% Define system parameters
m1 = 1;
m2 = 0.5;
k = 200;
b = 1;
```

```
% Define timve-varying parameters
rho1 = tvreal('rho1',[0 9]);
rho2 = tvreal('rho2',[0 25]);

% Define system matrices:
A = [   0              0             1       0    ; ...
        0              0             0       1    ; ...
    rho1-k/m1    -k/m1        -b/m1     0    ; ...
      -k/m2         rho2-k/m2   0     -b/m2  ];

B = [    0      0       0     ; ...
         0      0       0     ; ...
      1/m1   .1/m1      0     ; ...
         0      0     .1/m2 ];

C = [ 0 1 0 0];

D = [0 0 0];

% Define the parameter-varying LFT system:
sys = ss(A,B,C,D)
```

```
Continuous-time PLFTSS with 1 outputs, 3 inputs, 4 states.
The model consists of the following blocks:
  rho1: Time-varying real, range = [0,9], rate bounds = [-Inf,Inf], 1 occurrences
  rho2: Time-varying real, range = [0,25], rate bounds = [-Inf,Inf], 1 occurrences
```

LPV Control Design

We will design an LPV controller that optimizes the induced L_2 norm of the weighted interconnection shown in Figure 2.

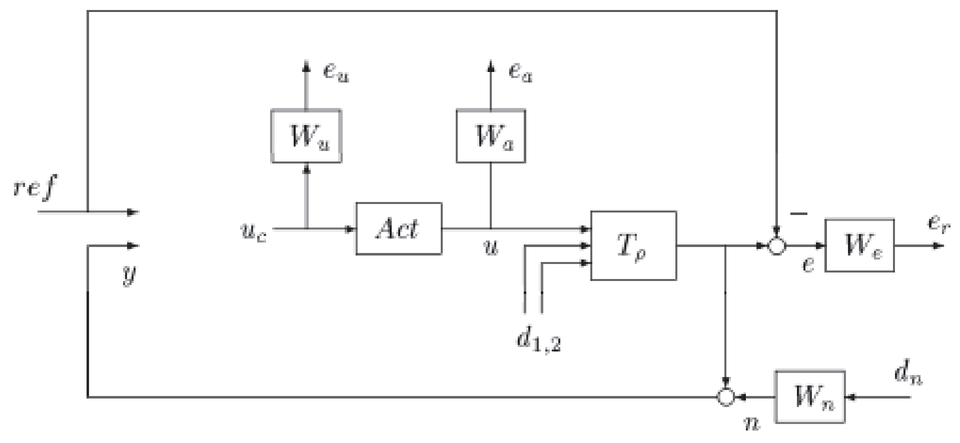


Figure 2: The weighted design interconnection [1].

The weights for the interconnection are chosen as [1]:

```
% Define weights:
We = tf([0.3 1.2],[1 0.04]);
```

```
Wu = tf([1 0.1],[0.01 125]);
Wa = 0.00001;
Wn = tf([1,0.4],[0.01 400]);
act = tf(1,[0.01 1]);
```

and we can form the weighted interconnection shown in Figure 2:

```
% Form synthesis interconnection:
systemnames = 'sys act We Wu Wa Wn';
inputvar = '[ref;d{2};dn;u]';
outputvar = '[Wu;Wa;We;ref;sys+Wn]';
input_to_sys = '[act;d]';
input_to_act = '[u]';
input_to_Wa = '[act]';
input_to_Wu = '[u]';
input_to_Wn = '[dn]';
input_to_We = '[sys-ref]';
G = sysic;
```

We will use the function `lpvsyn` to synthesize a LFT-based LPV controller which minimizes the <..\..\Concepts\StabilityAndInducedGain\html\StabilityAndInducedGain.html stochastic LPV bound > of the weighted interconnection `G` in Figure 2. This synthesis will assume that the rate of variation in ρ_1 and ρ_2 are unbounded, i.e. there is no limit to fast the parameters can change with time.

```
% Synthesize two-degree of freedom controller.
nmeas = 2;                                    % # of measurements
ncont = 1;                                    % # of controls
[Knr,Gamma,Info] = lpvsyn(G,nmeas,ncont);
```

The control design is successfull, and the controller `Knr` is guarenteed to achieve an induced L_2 norm that is less than or equal to `Gamma`

```
Gamma
```

```
Gamma =
    1.3361
```

Evaluating Pointwise Performance

Lets look at the closed-loop response for the original LPV system `sys` in the loop with the controller `Knr`:

```
% Form closed-loop sytem
systemnames = 'sys act Knr';
inputvar = '[r;d{2}]';
outputvar = '[sys]';
input_to_sys = '[act;d]';
input_to_act = '[Knr]';
input_to_Knr = '[r;sys]';
```

```
CL = sysic;
```

We will start by applying LTI analysis techniques to evaluate the performance of the LFT-based LPV controller. The LFT-based LPV system can be transformed into a LTI system by holding the parameters at a constant value. We will evalaluate the closed-loop LPV system on a 3x3 grid of parameter values defined by: $[\rho_1, \rho_2] = [0, 4.5, 9] \times [0, 12.5, 25]$.

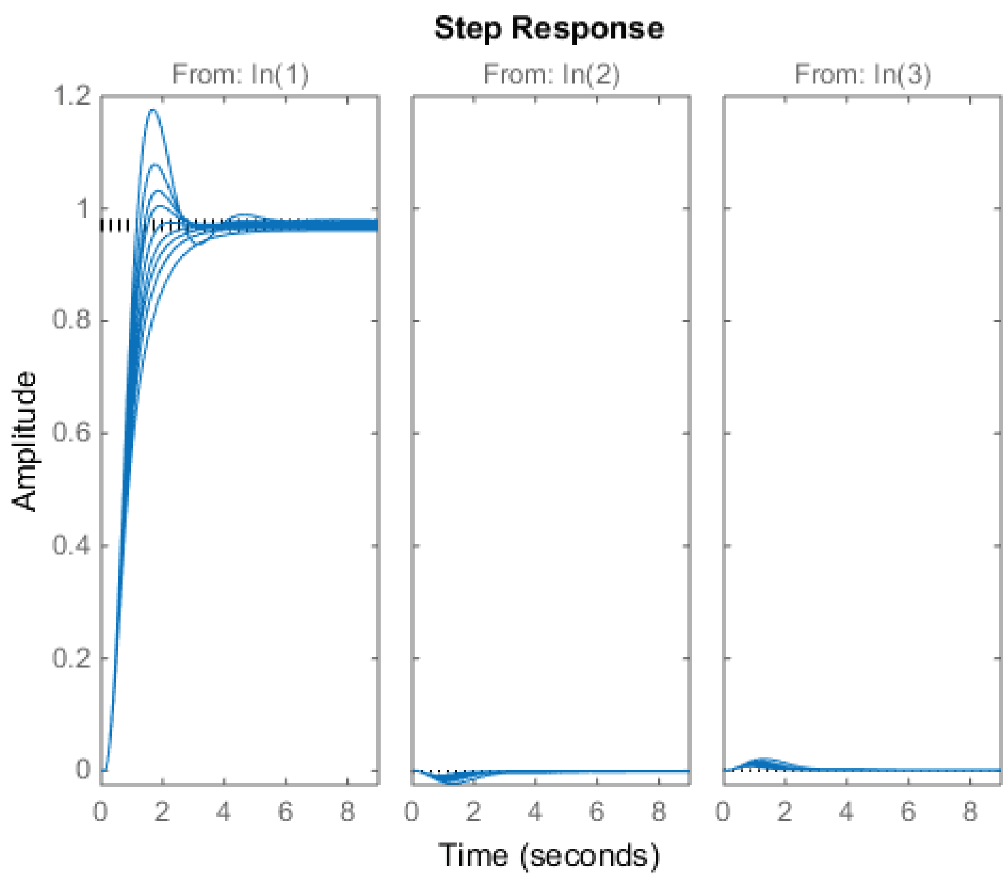
The syntax to perform pointwise LTI analysis requires the user to pass in a `rgrid` object that specifies the grid of parameter values that the LFT-based LPV system should be evaluated at:

```
% Define the grid of parameter values:
Domain = rgrid({'rho1','rho2'},{[0 4.5 9],[0 12.5 25]})
```

```
RGRID with the following parameters:
  rho1: Gridded real, 3 points in [0,9], rate bounds [-Inf,Inf].
  rho2: Gridded real, 3 points in [0,25], rate bounds [-Inf,Inf].
```

We can start by studying the pointwise step response. The Control System Toolbox's command `step` is overloaded to work with the `plftss` object. It will compute the step response at each point in the grid, and plot them together on the same plot:

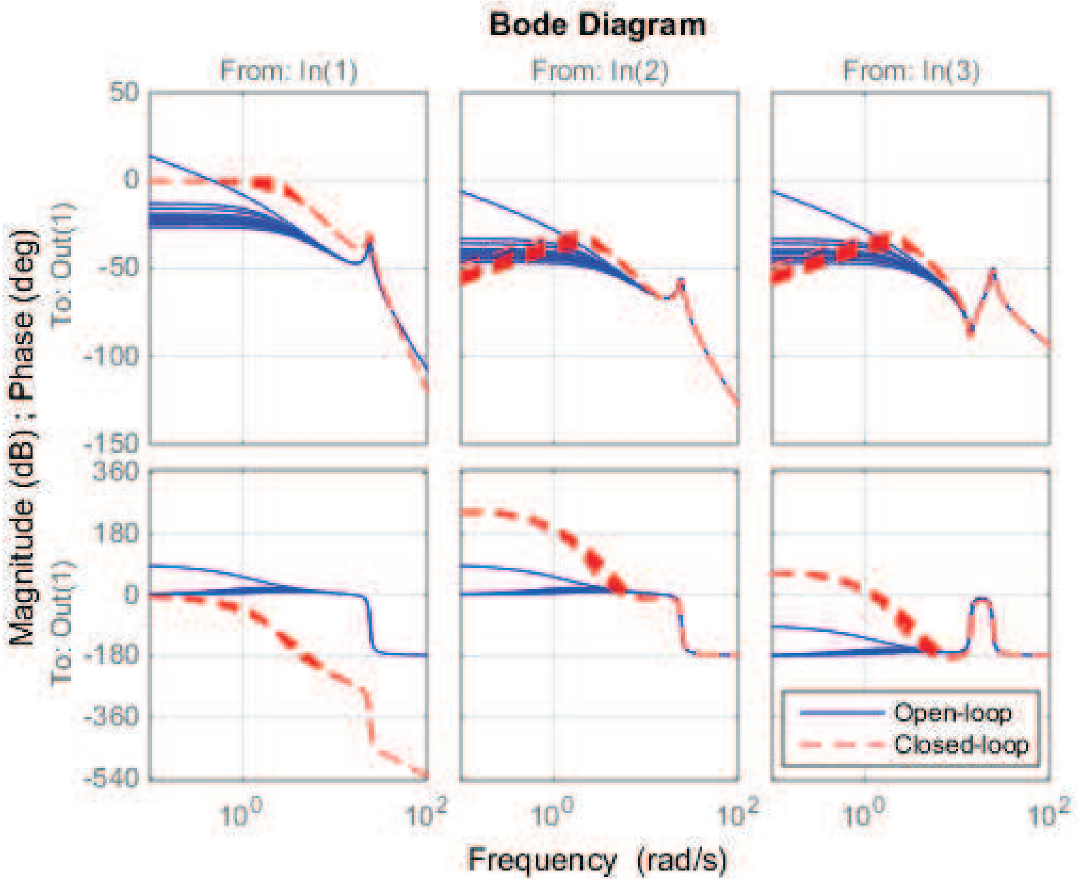
```
step(CL,Domain)
```



The controller achives approximatly 2.5 sec settling time and less than 3% steady-state tracking error. However, it suffers from large overshoot of approximatly 17% at $(\rho_1, \rho_2) = [9, 25]$. The effect of the disturbances is minimal.

Similarly we can create pointwise frequency responses using the `bode` command. Lets compare the pointwise frequency response of the open- and closed-loop system at frequencies between 0.1 to 20 rad/s

```
bode(sys,{0.1,100},'b',Domain)
hold on
bode(CL,{0.1,100},'r--',Domain)
legend('Open-loop','Closed-loop','location','best')
grid minor
hold off
```



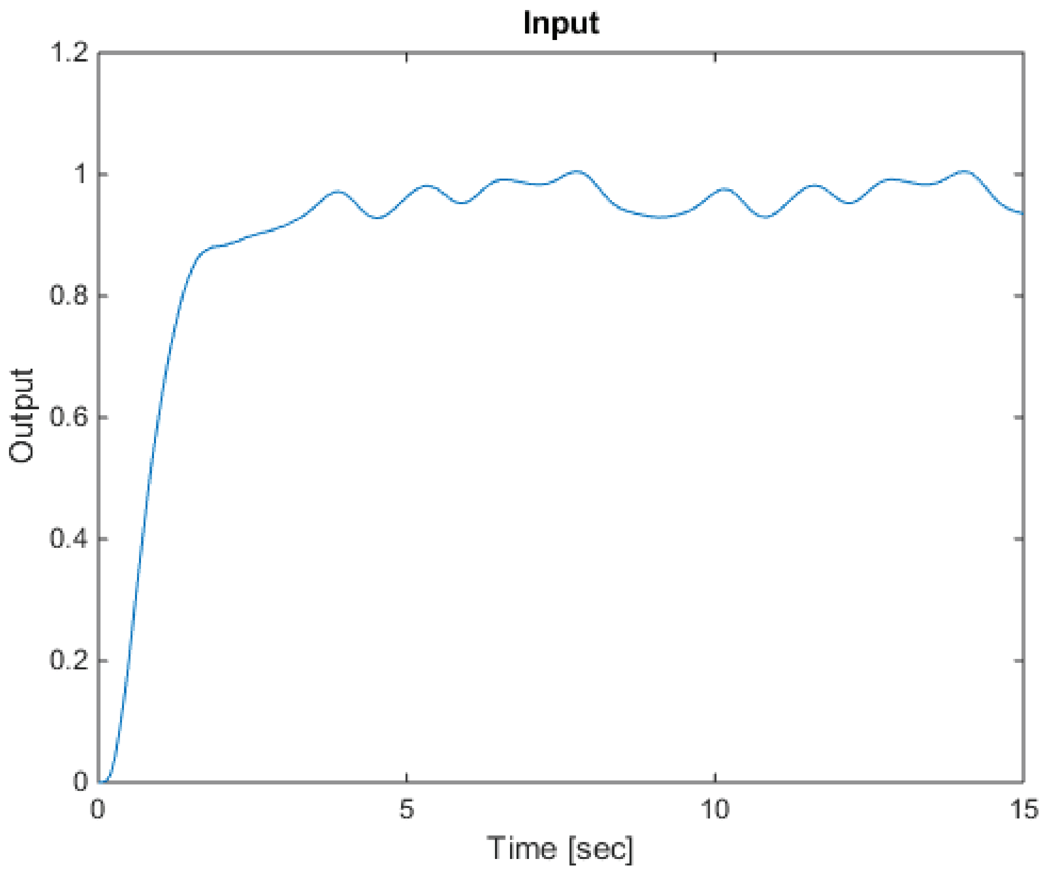
The closed-loop system has a bandwidth of approximatly 3 rad/s.

Evaluating LPV Time-Domain Performance

We will use the LPV simulation capabilities to inspect the performance of `Knr`. The parameters ρ_1 and ρ_2 will be made to vary with time: $\rho_1(t) = sin(t) + 1.5$ and $\rho_2(t) = 0.5 * cos(5 * t) + 3$, while the system tracks a unit step response and rejects disturbances $d_1 = cos(3t) + sin(5t) + cos(8t)$ and $d_2 = cos(t) + sin(2t) + cos(4t)$.

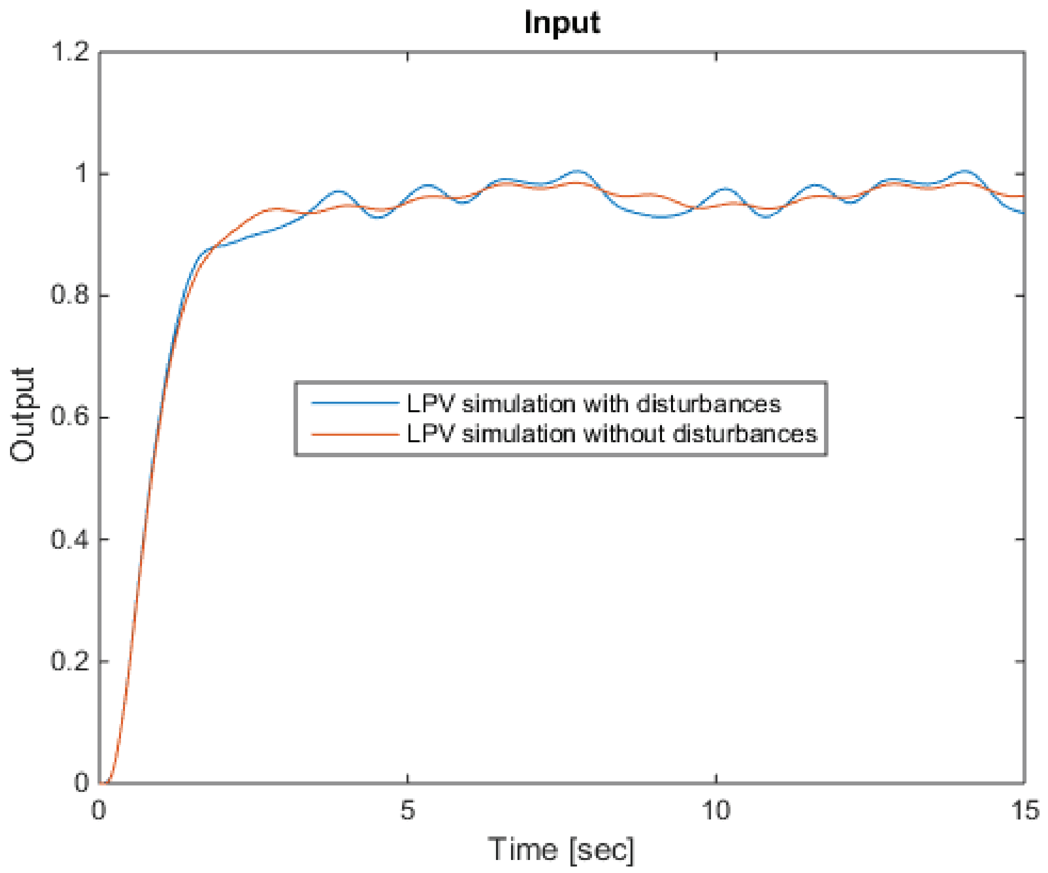
```
% Define the inputs to the system:
t = [0:0.01:15]';
u = ones(size(t));
d1 = cos(3*t)+sin(5*t)+cos(8*t);
d2 = cos(t)+sin(2*t)+cos(4*t);
```

```
% Define the trajectories of the parameters:
ptraj.time = t;
ptraj.rho1 = sin(t)+1.5;
ptraj.rho2 = .5*cos(5*t)+3;

% Perform LPV simulation:
lpvlsim(CL,ptraj,[u,d1,d2],t);
```



Lets evaluate the impact that the disturbances had on the response by repeating the simulation without disturbances:

```
lpvlsim(CL,ptraj,[u,d1,d2],t);
hold on
lpvstep(CL(:,1),ptraj);
legend('LPV simulation with disturbances',...
       'LPV simulation without disturbances',...
       'location','best')
```



The tracking performance is not very good. It is both oscillatory, and the steady-state error is noticable. The error due to the added distrubances is on the order of 5%, and they are confined to an approximately 1-2 Hz oscillation about the nominal. This frequency range is incidentally where the system's pointwise frequency response analysis indicated that disturbances would have the greatest effect on the output signal.

References

1. F. Wu, "Control of Linear Parameter Varying Systems," Ph.D. dissertation, Department of Mechanical Engineering, University of California at Berkeley, CA, May 1995.

Published with MATLAB® R2014b

List of Classes in the LPVTools Toolbox

Contents

Grid based parameter dependent matrices and systems

- PGRID - Time-varying real parameter defined on a grid of points.
- RGRID - Rectangular grid of parameter values.
- PMAT - Parameter-varying dependent matrix.
- PSS - Parameter-varying state-space model.
- PFRD - Parameter-varying frequency response data model.
- UPMAT - Parameter-varying uncertain matrix.
- UPSS - Parameter-varying uncertain state-space model.
- UPFRD - Parameter-varying uncertain frequency response data model.
- BASIS - Parameter-varying basis function (for synthesis).
- PSTRUCT - Parameter-varying structure.

LFT based parameter dependent matrices and systems

- TVREAL - Time-varying real parameter.
- PLFTMAT - Parameter-varying matrix in LFT framework.
- PLFTSS - Parameter-varying ss array in LFT framework.

RGRID - Rectangular grid object

Contents

Syntax

```
R = rgrid(IVName,IVData)
R = rgrid(IVName,IVData,IVRateBounds)
```

Description

`R = RGRID(IVName,IVData,IVRateBounds)` creates a rectangular grid object with independent variables and grid data specified by `IVName`, `IVData` and `IVRateBounds`. For an N-dimensional rectangular grid, `IVName` is an N-by-1 cell array of characters that specify the names of the independent variables. `IVData` is a N-by-1 cell array of column vectors that specify the grid data along each dimension. `IVRateBounds` is a N-by-1 double array with two columns, where each row corresponds to a parameter listed in `IVNames`, and each elements in the first column specifies a lower rate bound and each element in the second column specifies a upper rate bound. Each `IVData{i}` should be a vector of sorted, real data. If the `rgrid` contains only one independent variable then `IVName` can be specified as a single char, `IVData` can be specified as a single vector of sorted real data, and `IVRateBounds` can be specified as a 1-by-2 row vector of real numbers. `R = RGRID(IVName,IVData)` creates a `rgrid` with no limits on the rate bounds of the parameter, i.e. $\pm\infty$.

Example

```
% Create an RGRID object with independent variable 'a', grid data 4:10.
r1 = rgrid('a',4:10)
```

```
RGRID with the following parameters:
  a: Gridded real, 7 points in [4,10], rate bounds [-Inf,Inf].
```

```
% Create an RGRID object with independent variable 'a', grid data 4:10
% and parameter rate bounds [-1 1].
r1 = rgrid('a',4:10,[-1,1])
```

```
RGRID with the following parameters:
  a: Gridded real, 7 points in [4,10], rate bounds [-1,1].
```

```
% Create a 2-dimensional RGRID object
r2 = rgrid( {'a', 'b'}, {linspace(-2,2,12), 1:5},[-1 1;-4 8] )
```

```
RGRID with the following parameters:
  a: Gridded real, 12 points in [-2,2], rate bounds [-1,1].
  b: Gridded real, 5 points in [1,5], rate bounds [-4,8].
```

```
% Access independent variable name along first dimension
r2.IVName{1}
```

```
ans =
a
```

```
% Access independent variable rate bounds for parameter 'a'
r2.a.IVRateBounds
```

```
ans =
    -1     1
```

```
% Replace independent variable rate bounds for parameter 'b'
r2.b.IVRateBounds = [-10 10]
```

```
RGRID with the following parameters:
  a: Gridded real, 12 points in [-2,2], rate bounds [-1,1].
  b: Gridded real, 5 points in [1,5], rate bounds [-10,10].
```

Published with MATLAB® R2014b

PGRID - Real parameter defined on a grid of points

Contents

Syntax

```
A = pgrid(Name,GridData)
A = pgrid(Name,GridData,RateBounds)
```

Description

`A = pgrid(Name,GridData,RateBounds)` specifies the name, grid points, and ratebounds for a time-varying real parameter where:

- `Name` is a character string specifying the name.
- `GridData` is a N-by-1 column vector of sorted values that specify the parameter grid.
- `RateBounds` is a 1-by-2 row vector specifying lower and upper bounds on the derivative of the parameter with respect to time. Set RateBounds(1)=-inf and/or RateBounds(2)=+inf to denote an unbounded rate of change. RateBounds are optional, and a two argument call: `A = pgrid(Name,GridData)` will set them to a default value of [-inf,+inf].

The `pgrid` object describes a time-varying real parameter in the grid-based LPV framework. It is defined on a grid of real values, as seen in Figure 1. The rate bounds of the parameter specify how fast the parameter's value can change with time.

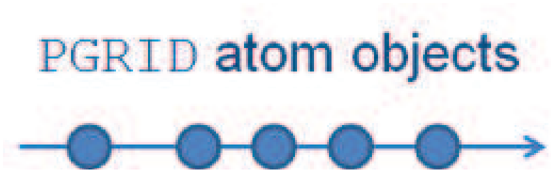


Figure 1: The `pgrid` object.

`pgrid` is used to specify parameter varying matrices and systems using analytical expressions. In this regard it is analogous to the `ureal` object in the Robust Control Toolbox, which is used to construct uncertain matrices and systems.

Example: Defining a `pgrid`

Define a time-varying real parameter q with values between $1 \leq q \leq 30$ and ratebounds $-10 \leq \dot{q} \leq 20$. The following commands model q as a `pgrid` with 30 grid points: 1,2,3,...,30. The first argment to \texttt{pgrid} is the name of the parameter, the second is a vector of grid points, and the third is a vector containing the upper and lower limits on the parameter's rate of variation.

```
q = pgrid('q',1:30, [-10 20])
```

```
Gridded real parameter "q" with 30 points in [1,30] and rate bounds [-10,20].
```

Properties of `pgrid`

`pgrid` objects have the following properties:

Name	string specifying the name of the parameter.
GridData	N-by-1 column vector of sorted values that specify the parameter grid.
Range	1-by-2 row vector specifying the lower and upper bounds on the parameter values.
RateBounds	1-by-2 row vector specifying lower and upper bounds on the derivative of the parameter with respect to time.

Example: Accessing and setting `pgrid` properties

The following commands will create a `pgrid` object and demonstrate how to access and set its properties:

Define a parameter with default rate bounds ($\pm\infty$)

```
p = pgrid('p',1:10)
```

```
Gridded real parameter "p" with 10 points in [1,10] and rate bounds [-Inf,Inf].
```

Change the name of the paramter from p to z

```
p.Name = 'z'
```

```
Gridded real parameter "z" with 10 points in [1,10] and rate bounds [-Inf,Inf].
```

Set the ratebounds to be +/- 5

```
p.RateBounds = [-5 5]
```

```
Gridded real parameter "z" with 10 points in [1,10] and rate bounds [-5,5].
```

Retreive the grid points that define the parameter

```
p.GridData
```

```
ans =
     1
     2
     3
     4
     5
     6
     7
     8
     9
    10
```

Published with MATLAB® R2014b

PMAT - Parameter-varying matrix

Contents

Syntax

```
M = pmat(Data,Domain)
```

Description

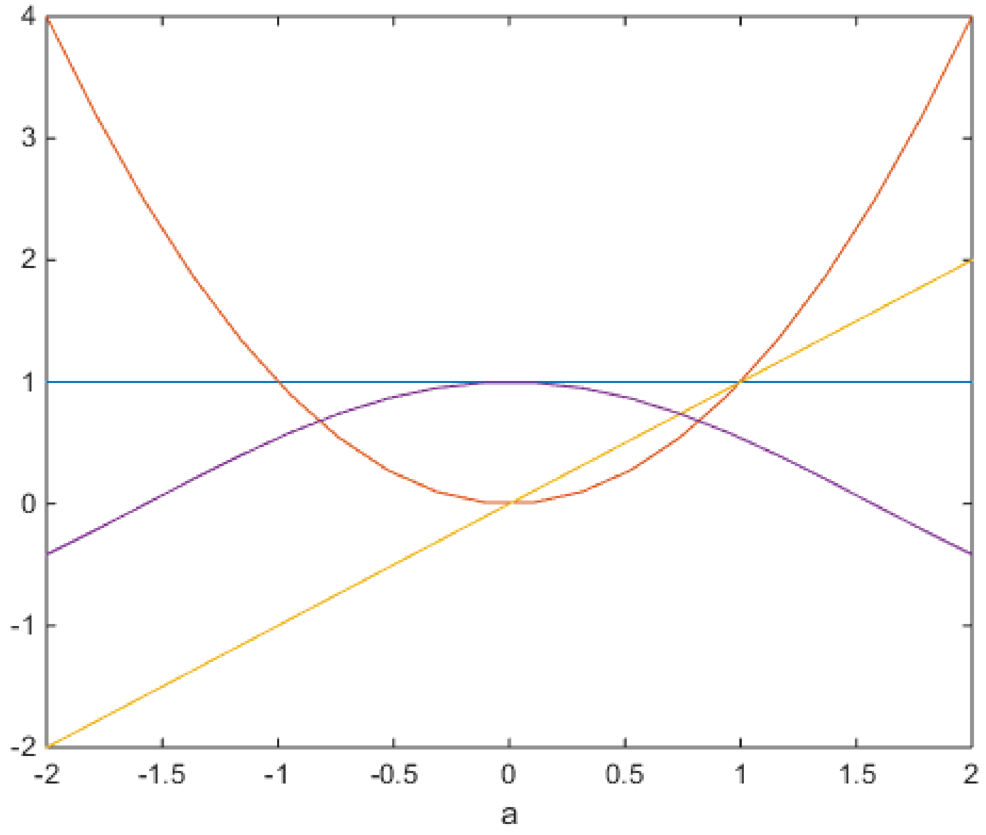
`M = pmat(Data,Domain)` creates a parameter-varying matrix defined on an N-dimensional rectangular grid. Domain is an `rgrid` object that specifies the N independent variables and the rectangular grid domain. Data is an (N+2) dimensional double array that specifies the matrix data. `Data(:,:,i1,...,iN)` is the value of the matrix evaluated at the point `Domain(i1,....,iN)`.

Example

```
% Create a 2-by-2 matrix defined on a 1-dimensional grid
IVData = linspace(-2,2,20);
Domain = rgrid('a',IVData);
for i=1:length(IVData)
  Data(1:2,1:2,i) = [1 IVData(i); IVData(i)^2 cos(IVData(i))];
end
M = pmat(Data,Domain)
```

```
PMAT with 2 rows and 2 columns.
The PMAT consists of the following blocks:
  a: Gridded real, 20 points in [-2,2], rate bounds [-Inf,Inf].
```

```
% Plot entries of M versus the independent variable
lpvplot(M);
```



Published with MATLAB® R2014b

PSS - Parameter-varying state-space model

Contents

Syntax

`S = pss(Data,Domain)`

Description

`S = pss(Data,Domain)` creates a parameter-varying state-space model defined on an N-dimensional rectangular grid. `Domain` is an `rgrid` object that specifies the N independent variables and the rectangular grid domain. `Data` is an N-dimensional state-space array that specifies the state space data. `Data(:,:,i1,...,iN)` is the model evaluated at the point `Domain(i1,....,iN)`.

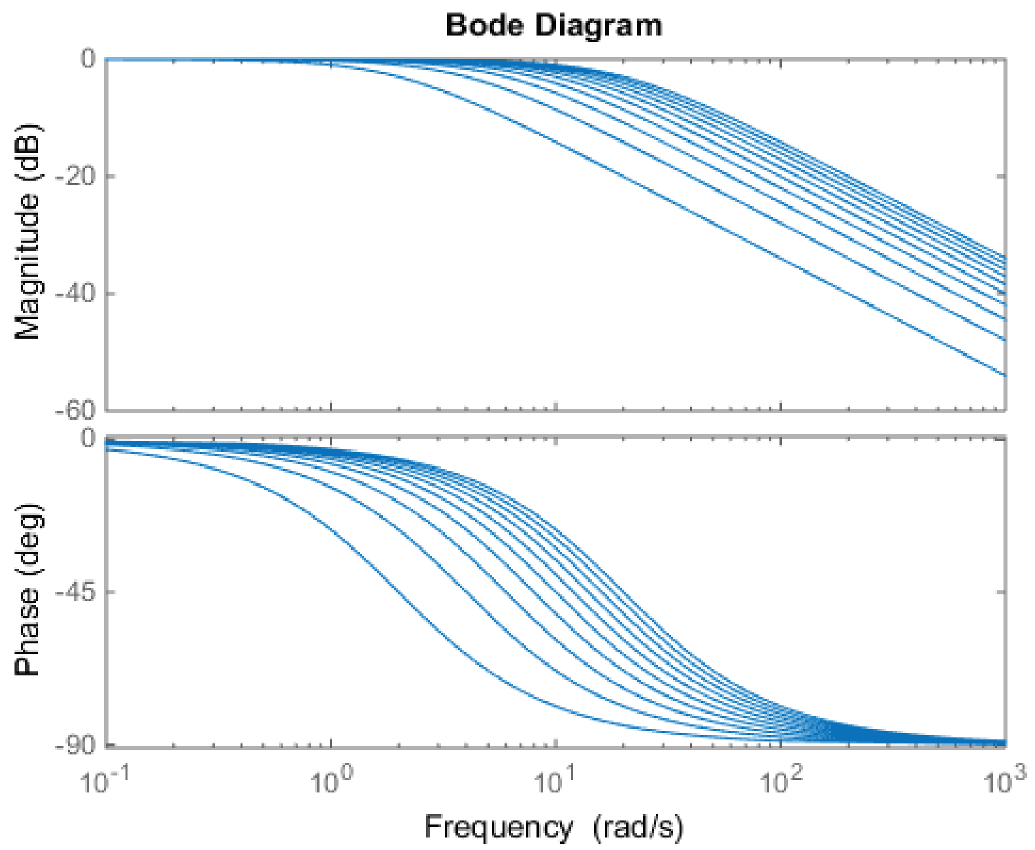
Example

```
% Create a 1-by-1 state-space model defined on a 1-dimensional grid
IVData = linspace(2,20,10);
Domain = rgrid('a',IVData);
for i=1:length(IVData)
  Data(1,1,i) = ss(-IVData(i),IVData(i),1,0);
end
S = pss(Data,Domain)
```

```
PSS with 1 States, 1 Outputs, 1 Inputs, Continuous System.
The PSS consists of the following blocks:
  a: Gridded real, 10 points in [2,20], rate bounds [-Inf,Inf].
```

Overlay Bode plots at each independent variable

```
bode(S)
```



Published with MATLAB® R2014b

PFRD - Parameter-varying frequency response data model

Contents

Syntax

`S = pfrd(Data,Domain)`

Description

`S = pfrd(Data,Domain)` creates a parameter-varying frequency response data model defined on an N-dimensional rectangular grid. `Domain` is an `rgrid` object that specifies the N independent variables and the rectangular grid domain. `Data` is an N-dimensional frequency response data (`frd`) array. `Data(:,:,i1,...,iN)` is the frequency response data evaluated at the point `Domain(i1,....,iN)`.

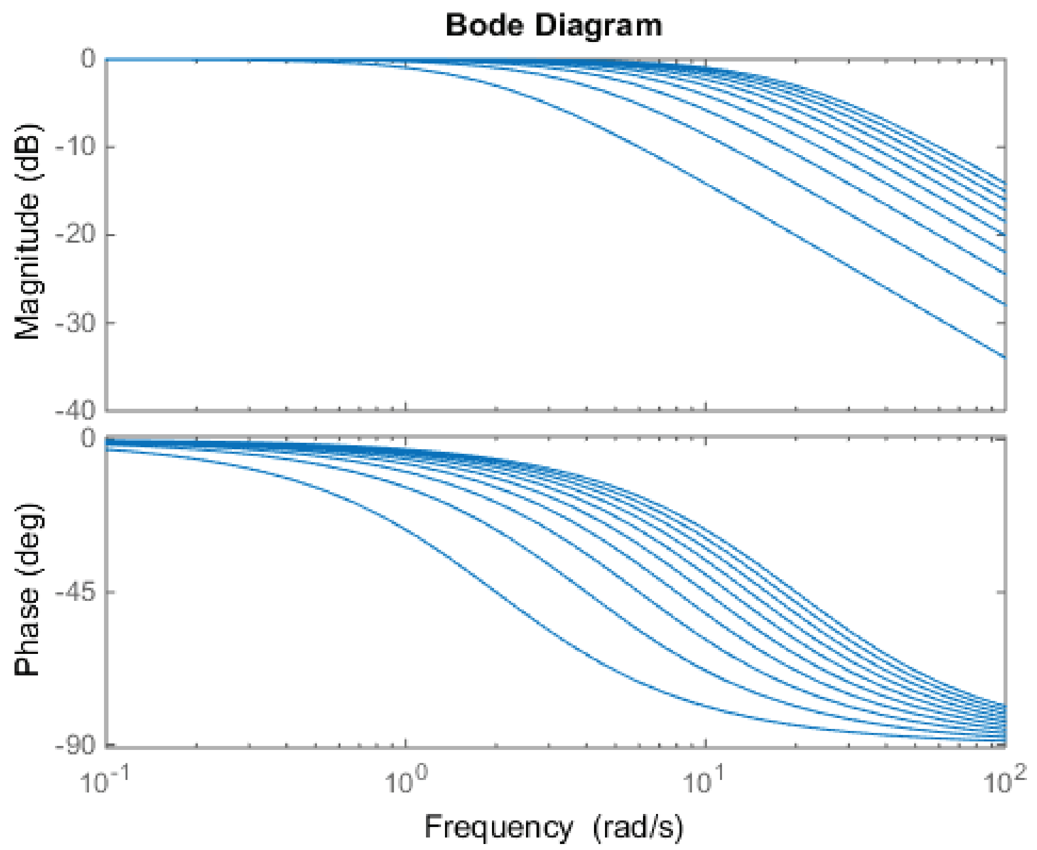
Example

```
% Create a 1-by-1 frequency response model defined on a 1-dimensional grid
IVData = linspace(2,20,10);
Domain = rgrid('a',IVData);
omeg = logspace(-1,2,30);
for i=1:length(IVData)
  sys = ss(-IVData(i),IVData(i),1,0);
  Data(1,1,i) = frd(sys,omeg);
end
S = pfrd(Data,Domain)
```

```
PFRD with 1 Outputs, 1 Inputs, Continuous System, 30 Frequency points.
The PFRD consists of the following blocks:
  a: Gridded real, 10 points in [2,20], rate bounds [-Inf,Inf].
```

Overlay Bode plots at each independent variable

```
bode(S);
```



Published with MATLAB® R2014b

UPMAT - Uncertain parameter-varying matrix

Contents

Syntax

`M = upmat(Data,Domain)`

Description

`M = upmat(Data,Domain)` creates an uncertain parameter-varying matrix defined on an N-dimensional rectangular grid. `Domain` is an `rgrid` object that specifies the N independent variables and the rectangular grid domain. `Data` is an (N+2) dimensional double array that specifies the matrix data. `Data(:,:,i1,...,iN)` is the value of the matrix evaluated at the point `Domain(i1,....,iN)`.

Example

```
% Create a 2-by-2 matrix defined on a 1-dimensional grid
IVData = linspace(-2,2,20);
Domain = rgrid('a',IVData);
au = ureal('au',-2.3);
for i=1:length(IVData)
    Data(1:2,1:2,i) = [1 au*IVData(i); IVData(i)^2 cos(IVData(i))];
end
M = upmat(Data,Domain)
```

```
UPMAT with 2 rows and 2 columns.
The UPMAT consists of the following blocks:
  a: Gridded real, 20 points in [-2,2], rate bounds [-Inf,Inf].
  au: Uncertain real, nominal = -2.3, variability = [-1,1], 1 occurrences
```

Published with MATLAB® R2014b

UPSS - Uncertain parameter-varying state-space model

Contents

Syntax

`S = upss(Data,Domain)`

Description

`S = upss(Data,Domain)` creates an uncertain parameter-varying state-space model defined on an N-dimensional rectangular grid. `Domain` is an `rgrid` object that specifies the N independent variables and the rectangular grid domain. `Data` is an N-dimensional uncertain state-space array that specifies the uncertain state-space data. Note that `Data` must contain the same uncertainty structure across the array dimensions. `Data(:,:,i1,...,iN)` is the model evaluated at the point `Domain(i1,....,iN)`.

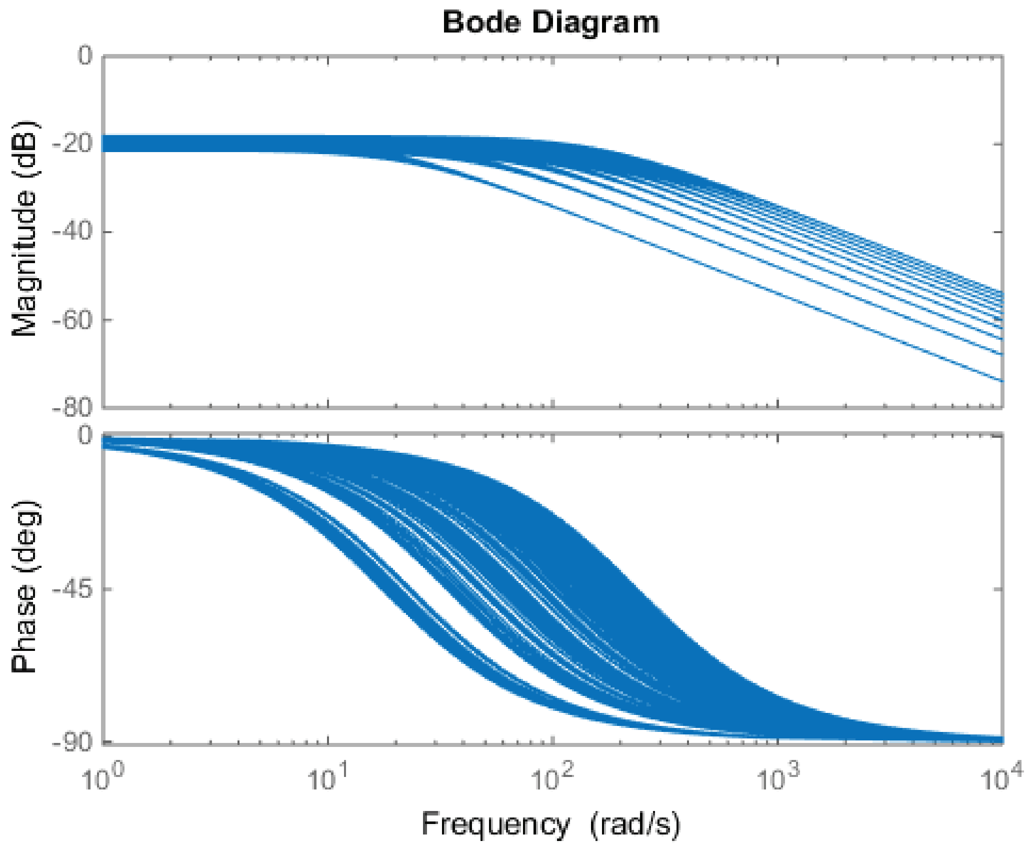
Example:

```
% Create a 1-by-1 uncertain, state-space model defined on a
% 1-dimensional grid
IVData = linspace(2,20,10);
Domain = rgrid('a',IVData);
theta = ureal('theta', 10,'Range',[8 12]);
for i=1:length(IVData)
  Data(1,1,i) = ss(-IVData(i)*theta,IVData(i),1,0);
end
US = upss(Data,Domain)
```

```
UPSS with 1 States, 1 Outputs, 1 Inputs, Continuous System.
The UPSS consists of the following blocks:
  a: Gridded real, 10 points in [2,20], rate bounds [-Inf,Inf].
  theta: Uncertain real, nominal = 10, range = [8,12], 1 occurrences
```

Overlay Bode plots at each independent variable

```
bode(US);
```



Published with MATLAB® R2014b

UPFRD - Uncertain parameter-varying frequency response data model

Contents

Syntax

`S = upfrd(Data,Domain)`

Description

`S = upfrd(Data,Domain)` creates an uncertain parameter-varying frequency response data model defined on an N-dimensional rectangular grid. `Domain` is an `rgrid` object that specifies the N independent variables and the rectangular grid domain. `Data` is an N-dimensional uncertain frequency response data (`ufrd`) array. `Data(:,:,i1,...,iN)` is the uncertain frequency response data evaluated at the point `Domain(i1,....,iN)`.

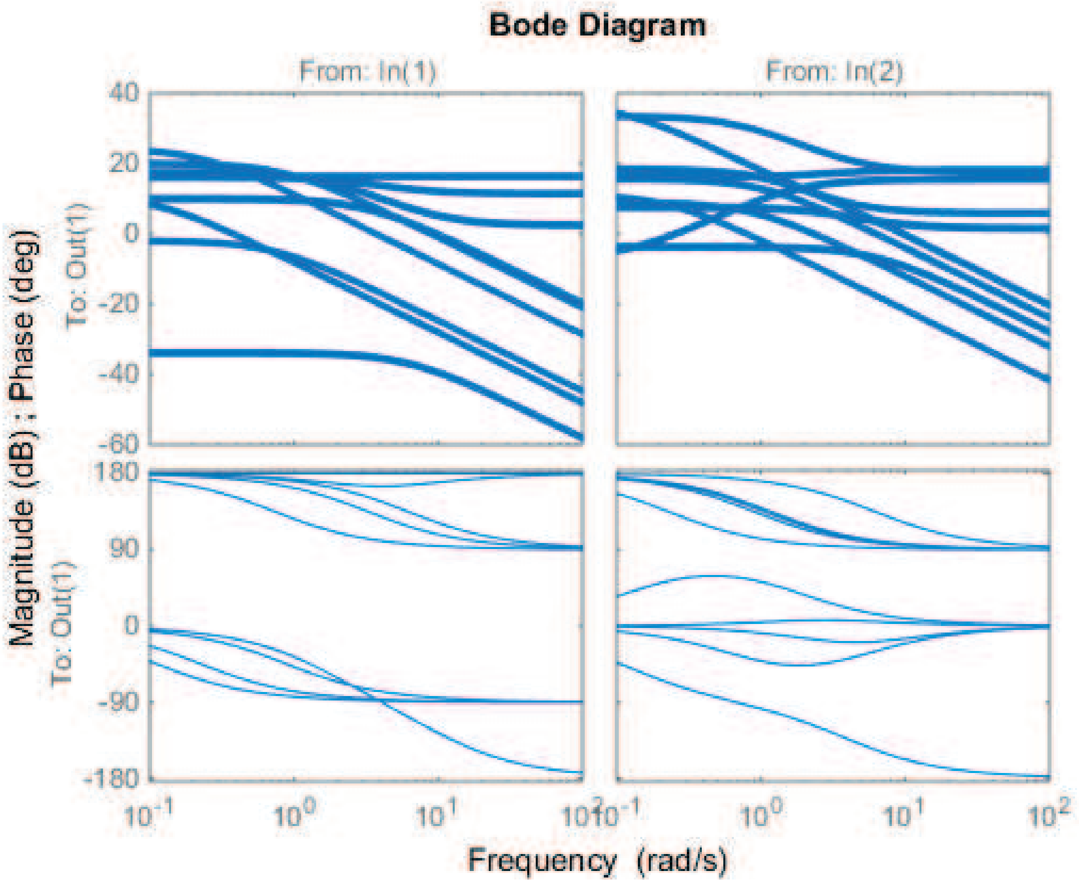
Example

```
% Create a 1-by-1 UFRD defined on a 1-dimensional grid
IVData = linspace(2,20,10);
Domain = rgrid('a',IVData);
omeg = logspace(-1,2,30);
unc = ureal('unc',10)
usys = rss(1,1,2,10)*unc;
Data = ufrd(usys,omeg);
S = upfrd(Data,Domain)
```

```
unc =

  Uncertain real parameter "unc" with nominal value 10 and variability [-1,1].

UPFRD with 1 Outputs, 2 Inputs, Continuous System, 30 Frequency points.
The UPFRD consists of the following blocks:
  a: Gridded real, 10 points in [2,20], rate bounds [-Inf,Inf].
  unc: Uncertain real, nominal = 10, variability = [-1,1], 1 occurrences
```

Overlay Bode plots at each independent variable

```
bode(S);
```



Published with MATLAB® R2014b

PSTRUCT - Parameter-varying structure

Contents

Syntax

`M = pstruct(Data,Domain)`

Description

`M = pstruct(Data,Domain)` creates a parameter-varying structure defined on an N-dimensional rectangular grid. `Domain` is an `rgrid` object that specifies the N independent variables and the rectangular grid domain. `Data` is an (N+2) dimensional structured array that specifies the data. `Data(:,:,i1,...,iN)` is the value of the struct array evaluated at the point `Domain(i1,....,iN)`.

Note: Use M.FieldName to access the field named 'FieldName' in M. If possible, the content of the field is returned as an object (e.g. pmat, pss), otherwise it is returned as a cell array.

Published with MATLAB® R2014b

BASIS - Basis function object

Contents

Syntax

```
b = basis(F,NAME1,PARTIAL1,NAME2,PARTIAL2,...)
b = basis(F,DERIVATIVE)
```

Description

`basis` functions are needed for rate-bounded LPV analysis and synthesis. These are functions of the independent variables present in the system being analyzed. `basis` functions are specified as scalar `pmat`. All partial derivatives must also be provided by the user.

`b = basis(F,NAME1,PARTIAL1,NAME2,PARTIAL2,...)` creates a `basis` function object with the function F, which is a scalar `pmat`. If there are N independent variables in F, then 2*N additional arguments are specified, which are pairs of (1) an independent variable name (as `char`), and (2) the corresponding partial derivative (as `pmat`) of F with respect to that independent variable.

`basis(F,DERIVATIVE)` is the same as `basis(F,NAME1,DERIVATIVE)` if F has only one independent variable.

Example

```
theta = pgrid('theta',0:linspace(0,2*pi,20));
psi = pgrid('psi',linspace(0,2*pi,10));
F = cos(theta)*sin(2*psi);
pTheta = -sin(theta)*sin(2*psi);
pPsi = 2*cos(theta)*cos(2*psi);
B = basis(F,'theta',pTheta,'psi',pPsi)
```

```
BASIS: 1 basis functions and 2 partial derivatives with respect to 2 PGRIDs
The BASIS object consists of the following blocks:
  psi: Gridded real, 10 points in [0,6.28], rate bounds [-Inf,Inf].
  theta: Gridded real, 1 points in [0,0], rate bounds [-Inf,Inf].
```

Published with MATLAB® R2014b

TVREAL - Time-varying real parameter

Contents

Syntax

```
A = tvreal(Name,Range)
A = tvreal(Name,Range,RateBounds)
```

Description

`A = tvreal(Name,Range,RateBounds)` specifies the name, range, and ratebounds for a time-varying real parameter, where * `Name` is a character string specifying the name * `Range` is a 1-by-2 row vector specifying the lower and upper limits for the `tvreal`. * `RateBounds` is a 1-by-2 row vector specifying lower and upper bounds on the derivative of the parameter with respect to time. Set `RateBounds(1)=-inf` and/or `RateBounds(2)=+inf` to denote an unbounded rate of change. `RateBounds` are optional, and a two argument call: `A = tvreal(Name,GridData)` will set them to a default value of `[-inf,+inf]`.

Example

```
% Create a tvreal "a" which has range [-2 2] and ratebounds [-1 1].
a = tvreal('a',[-2 2],[-1 1])
```

```
Time-varying real parameter "a" with range [-2,2] and rate bounds [-1,1].
```

```
% Create a tvreal "b" which has range [2 20] and default
% ratebounds [-inf inf].
b = tvreal('b',[2 20])
```

```
Time-varying real parameter "b" with range [2,20] and rate bounds [-Inf,Inf].
```

```
% Use tvreal as a building block: Create a 2-by-2 matrix that depends
% on the tvreal "a".
M = [1, a;a^2, -a]
```

```
PLFTMAT with 2 rows and 2 columns.
The PLFTMAT consists of the following blocks:
  a: Time-varying real, range = [-2,2], rate bounds = [-1,1], 4 occurrences
```

```
% Use tvreal as a building block to build a plftss: Create a 1-by-1
% state-space model that depends on tvreals "a" and "b".
S = ss(-a,b,1,0)
```

```
Continuous-time PLFTSS with 1 outputs, 1 inputs, 1 states.
The model consists of the following blocks:
  a: Time-varying real, range = [-2,2], rate bounds = [-1,1], 1 occurrences
  b: Time-varying real, range = [2,20], rate bounds = [-Inf,Inf], 1 occurrences
```

Published with MATLAB® R2014b

PLFTMAT - Parameter-varying matrix in LFT framework

Contents

Syntax

`M = plft(Data,RateBounds)`

Description

`M = plft(Data,RateBounds)` creates a parameter-varying matrix. `Data` is a `umat`. `RateBounds` is a N-by-2 cell array listing the rate bound information for each independent variable in the `plftmat`. `RateBounds{i,1}` is the character string name of the i-th independent variable and `RateBounds{i,2}` is a sorted real vector of form [Low, High] specifying its rate bounds. `RateBounds` must only contain names of `ureal` objects that exist in `Data` and this indicates that those `ureal` are actually `tvreal` representing the independent variables.

Example

```
% Create a 2-by-2 matrix that depends on TVREAL a.
a = tvreal('a',[-2 2],[-1 1]);
M = [1, a;a^2, -a]
```

```
PLFTMAT with 2 rows and 2 columns.
The PLFTMAT consists of the following blocks:
  a: Time-varying real, range = [-2,2], rate bounds = [-1,1], 4 occurrences
```

Published with MATLAB® R2014b

PLFTSS - Parameter-varying state-space model in LFT framework

Contents

Syntax

`M = plftss(Data,RateBounds)`

Description

`M = plftss(Data,RateBounds)` creates a parameter-varying state-space model. `Data` is a `uss`. `RateBounds` is a N-by-2 cell array listing the rate bound information for each independent variable in the `plftss`. `RateBounds{i,1}` is the character string name of the i-th independent variable and `RateBounds{i,2}` is a sorted real vector of form [Low, High] specifying its rate bounds. `RateBounds` must only contain names of `ureal` objects that exist in `Data` and this indicates that these `ureal` are actually `tvreal` representing the independent variables.

Example

```
% Create a 1-by-1 state-space model S that depends on tvreal b.
b = tvreal('b',[2 20]);
S = ss(-b,b,1,0)
```

```
Continuous-time PLFTSS with 1 outputs, 1 inputs, 1 states.
The model consists of the following blocks:
  b: Time-varying real, range = [2,20], rate bounds = [-Inf,Inf], 1 occurrences
```

Published with MATLAB® R2014b

List of Function in the LPVTools Toolbox

Contents

Manipulation of parameter dependent models

- DOMUNION - Map LPV objects onto a common domain.
- GRDI2LFT - Transform a grid-based LPV model into a LFT model.
- LFT2GRID - Transform a LFT model into a grid-based LPV model.
- LPVSPLIT - Extract grid-based LPV model data based on IV range.
- LPVINTERP - Interpolate a grid-based LPV object.
- LPVSUBS - Substitutes values for parameters.
- LPVELIMIV -Eliminate singleton independent variables.
- LPVSAMPLE - Sample a LPV model at randomly chosen points parameter values.
- LPVBALANCE - Diagonal scaling for PSS objects.
- LPVPLOT - Plot PMAT data as a function of the parameter.

Model order reduction

- LPVGRAM - Compute Gramians for PSS objects.
- LPVBALREAL - Perform Gramian-based balancing for PSS objects.
- LPVBALANCMR - Balanced truncation model reduction.
- LPVNCFMR - Balanced normalized coprime factor model reduction.

Robustness and worst-case analysis.

- LPVNORM - Bound on induced L2 norm for PSS systems.
- LPVWCGAIN - Worst-case gain of an uncertain LPV system.

Controller synthesis.

- LPVSYN - Synthesize a LPV controller.
- LPVNCFSYN - Normalized coprime factor LPV controller synthesis.
- LPVMIXSYN - LPV loop-shaping synthesis.
- LPVLOOPSHAPE - LPV mixed-sensitivity synthesis.
- LPVSFSYN - Synthesize a LPV state feedback controller.
- LPVESTSYN - Synthesize a LPV state estimator.
- LPVSTOCHSYN - LPV controller synthesis for stochastic LPV systems.

- LPVSYNOPTIONS - Options object for LPV synthesis and analysis.

Time-domain analysis

- LPVLSIM - Simulate parameter dependent time response of a PSS.
- LPVSTEP - Simulate parameter dependent step response of a PSS.
- LPVINITIAL - Simulate initial conditions response of a PSS.
- LPVIMPULSE - Simulate parameter dependent impulse response of a PSS.

DOMUNION - Map LPV objects on a common domain

Contents

Syntax

```
[Aext,Bext]=domunion(A,B)
[A1ext,...,ANext] = domunion(A1,...,AN)
```

Description

Let `A` and `B` be grid-based LPV objects. If `A` depends on independent variables (X,Y) and `B` depends on independent variables (X,Z) then `[Aext,Bext]=domunion(A,B)` returns grid-based LPV objects Aext and Bext, of the same class, that have a common domain with independent variables (X,Y,Z). `Aext` evaluated at point (x,y,z) is given by `A` evaluated at (x,y). `Bext` evaluated at point (x,y,z) is given by `B` evaluated at (x,z).

Given grid-based LPV objects `A1`,|A2|,...,|AN|, the syntax `[A1ext,...,ANext] = domunion(A1,...,AN)` constructs `A1ext`, `A2ext`,..., `ANext` that are defined on a common domain.

Published with MATLAB® R2014b

GRID2LFT - Transform a grid-based LPV model into LFT

Contents

Syntax

```
[L,C,E] = grid2lft(G)
[L,C,E] = grid2lft(G,N)
[L,C,E] = grid2lft(G,NAMES,DEGREEMAT)
```

Description

Transform a grid-based LPV model into a LFT model with polynomial parameter dependence. Use linear regression to fit a polynomial in the parameters to the grid based data.

`L = grid2lft(G)` fits the elements of the matrix or state-space data in G with a linear parameter dependence. G is a grid based LPV model (e.g. pmat or upss) and L is an LFT based LPV model (e.g. plftmat, plftss).

`L = grid2lft(G,N)` fits the elements of the matrix or state-space data in G with polynomial parameter dependence of degree N.

`L = grid2lft(G,NAMES,DEGREEMAT)` fits the matrix or state-space data in G with polynomials, using a linear combination of monomials specified by the data in DEGREEMAT. NAMES is a 1-by-P cell array of chars, consisting of the P names of every independent variable in G. DEGREEMAT is a D-by-P matrix of nonnegative integers, each 1-by-P row corresponding to a monomial, defined by the nonnegative exponents associated with each independent variable.

`[L,C] = grid2lft(G,...)` returns|C|, the matrix of polynominal coefficients used in the transformation from grid-LPV to LFT. If G is a M-by-N matrix that is being fit with a polynominal with B terms, then C is a M-by-N-by-B `double` array, in which elements (i,k,:) correspond to (i,k)-th matrix element in G, and elements (:,:,r) correspond to the r-th basis function.

`[L,C,E] = grid2lft(G,...)` returns E, the root mean square error of the linear fit.

Example: Transform a PMAT to a PLFTMAT

```
% Create PMATs M and M2 with two independent variables x and y.
x = pgrid('x',linspace(-2,2,12),[-1 1]);
y = pgrid('y',1:5,[-4 8] );
M = [x+y-x*y x;3*y -2*x*y];
M2 = sqrt(1+x.^2)*y;

% Transform both M and M2 into LFT based LPV objects. Use a polynomial
% containing the factors (1,x,y,x*y) to perform the fitting for M, and
% a polynomial (1,x,y,x^2,x*y,x^2*y) to perform the fitting for M2.

% Call grid2lft and specify that the fitting of M should use the
% polynomial (1,x,y,x*y)
```

```
[Mlft,C,E] = grid2lft(M,{'x','y'},[0 0;1 0;0 1;1 1]);
```

```
Mlft
```

```
PLFTMAT with 2 rows and 2 columns.
The PLFTMAT consists of the following blocks:
  x: Time-varying real, range = [-2,2], rate bounds = [-1,1], 2 occurrences
  y: Time-varying real, range = [1,5], rate bounds = [-4,8], 2 occurrences
```

```
C
```

```
C(:,:,1) =
   1.0e-14 *
   -0.0097    0.0074
    0.1022   -0.0665
C(:,:,2) =
    1.0000    1.0000
   -0.0000    0.0000
C(:,:,3) =
    1.0000   -0.0000
    3.0000    0.0000
C(:,:,4) =
   -1.0000    0.0000
    0.0000   -2.0000
```

```
E

% Call grid2lft and specify that the fitting of M2 should use the
% polynomial (1,x,y,x^2,x*y,x^2*y)
[M2lft,C2,E2] = grid2lft(M2,{'x','y'},[0 0;1 0;0 1;2 0;1 1;2 1]);
```

```
E =
   1.0150e-15
```

```
M2lft
```

```
PLFTMAT with 1 rows and 1 columns.
The PLFTMAT consists of the following blocks:
  x: Time-varying real, range = [-2,2], rate bounds = [-1,1], 2 occurrences
  y: Time-varying real, range = [1,5], rate bounds = [-4,8], 1 occurrences
```

```
C2
```

```
C2(:,:,1) =
   4.6282e-16
C2(:,:,2) =
  -3.3723e-16
C2(:,:,3) =
    1.0646
C2(:,:,4) =
  -1.2078e-16
C2(:,:,5) =
   1.0533e-16
C2(:,:,6) =
    0.3058
```

```
E2
```

```
E2 =
    0.1491
```

Example: Transform a UPSS to a PLFTSS

```
% Create UPSS M that depends on two independent variables x and y.
x = pgrid('x',linspace(-2,2,12),[-1 1]);
y = pgrid('y',1:5,[-4 8] );
u = ureal('u',1);
M = ss(x+y-x*y*u, x+3*y,-2*x*y,pmat(0));

% Transform M into a LFT based LPV object. Use a polynomial containing
% the factors (1,x,y,x^2,x*y,x^2*y) to perform the fitting for M.

% Call grid2lft and specify that the fitting of M should use the
% polynomial (1,x,y,x^2,x*y,x^2*y)
[Mlft,C] = grid2lft(M,{'x','y'},[0 0;1 0;0 1;2 0;1 1;2 1]);
```

```
Mlft
```

```
Continuous-time PLFTSS with 1 outputs, 1 inputs, 1 states.
The model consists of the following blocks:
  u: Uncertain real, nominal = 1, variability = [-1,1], 1 occurrences
  x: Time-varying real, range = [-2,2], rate bounds = [-1,1], 3 occurrences
  y: Time-varying real, range = [1,5], rate bounds = [-4,8], 2 occurrences
```

```
C
```

```
C(:,:,1) =
    0.0000   -0.0000    0.0000
    1.0000         0         0
```

```
   -0.0000         0         0
  C(:,:,2) =
    1.0000    0.0000    1.0000
    0.0000         0         0
    0.0000         0         0
  C(:,:,3) =
    1.0000    0.0000    3.0000
   -0.0000         0         0
    0.0000         0         0
  C(:,:,4) =
   1.0e-14 *
   -0.0204    0.0052   -0.1099
   -0.0064         0         0
    0.0103         0         0
  C(:,:,5) =
   -1.0000   -1.0000   -0.0000
   -0.0000         0         0
   -2.0000         0         0
  C(:,:,6) =
   1.0e-15 *
    0.0431   -0.0120   -0.2844
    0.0000         0         0
   -0.0240         0         0
```

Published with MATLAB® R2014b

LFT2GRID - Transfrom LFT into grid-based LPV object

Contents

Syntax

```
M = lft2grid(L)
M = lft2grid(L,N)
M = lft2grid(L,DOMAIN)
```

Description

Transform a `tvreal`, `plftmat` or `plftss` object into a grid based LPV object `pmat`, `pss`, `upmat`, or `upss`. The transformation is performed by evaluating the PLFT object at a grid of parameter values.

`M = lft2grid(L)` evaluates `L` at 10 values of each independent parameter, sampled uniformly from the range of each parameter.

`M = lft2grid(L,N)` evaluates `L` at `N` values of each independent parameter, sampled uniformly from the range of each parameter.

`M = lft2grid(L,DOMAIN)` evaluates `L` at each point containted in `DOMAIN`. `DOMAIN` is an `rgrid` object that must contain the same independent variables as `L`.

Published with MATLAB® R2014b

LPVSPLIT - Extract data based on independent variable range

Contents

Syntax

```
B = lpvsplit(A,NAME1,RANGE1,NAME2,RANGE2,....)
B = lpvsplit(A,NAME1,INDEX1,NAME2,INDEX2,....,'index')
B = lpvsplit(A,NAME1,VALUES1,NAME2,VALUES2,....,'value')
B = lpvsplit(A,NAME,RANGE)
B = lpvsplit(A,NAME,INDEX,'index')
B = lpvsplit(A,NAME,VALUES,'value')
B = lpvsplit(A,DOMAIN)
```

Description

`B = lpvsplit(A,NAME1,RANGE1,NAME2,RANGE2,....)` extracts data from the grid-based LPV object A on the domain specified by the NAME / RANGE pairs. Each RANGE is a 1-by-2 row vector [min, max], that specifies the values of the independent variable to be extracted along the domain direction NAME. The data at all points in `A.Parameter."NAME".GridData` which lies inside RANGE is extracted. If RANGE is a scalar then the data is extracted where the variable NAME is exactly equal to RANGE. If an independent variable of A is not listed in the inputs then all values along this domain direction are retained in B.

`B = lpvsplit(A,NAME1,INDEX1,NAME2,INDEX2,....,'index')` extracts data from the `pmat` A on the domain specified by the NAME / INDEX pairs. Each INDEX is a vector of integers or a logical array that specifies the indices of `A.Parameter."NAME".GridData` to be extracted.

`B = lpvsplit(A,NAME1,VALUES1,NAME2,VALUES2,....,'value')` extracts data from the `pmat` A on the domain specified by the NAME / VALUES pairs. VALUES specifies the `A.Parameter."NAME".GridData` to be extracted.

`B = lpvsplit(A,NAME,RANGE)` is an alternative syntax. NAME is an N-by-1 cell array of characters and RANGE is an N-by-1 cell array of ranges. `B = lpvsplit(A,NAME,INDEX,'index')` and `B = lpvsplit(A,NAME,VALUES,'value')` also apply for INDEX or VALUES as a cell array.

`B = lpvsplit(A,DOMAIN)` is an another alternative syntax. DOMAIN is an `rgrid` object. This extracts data from the `pmat` A based on the independent variables and data ranges in DOMAIN.

Published with MATLAB® R2014b

LPVINTERP - Interpolate a grid-based LPV object

Contents

Syntax

```
B = lpvinterp(A,NAME,VALUES)
B = lpvinterp(A,NAME1,VALUES1,NAME2,VALUES2,...)
B = lpvinterp(A,NAME1,VALUES1,NAME2,VALUES2,....,METHOD)
```

Description

`B = lpvinterp(A,NAME1,VALUES1,NAME2,VALUES2,...)` interpolates a grid-based LPV system `A` on the domain specified by the `NAME` / `VALUES` pairs. Each `VALUE` is a vector of values at which to interpolate `A` along the domain direction of the corresponding `NAME`. If an independent variable of `A` is not listed in the inputs, then all values along this domain direction are retained in the output `B`.

`B = lpvinterp(A,NAME,VALUES)` is an alternative syntax. `NAME` is an N-by-1 cell array of characters and `VALUES` is an N-by-1 cell array of values.

`B = lpvinterp(A,NAME1,VALUES1,NAME2,VALUES2,....,METHOD)` includes a final input argument called `METHOD`, which specifies the interpolation method to be used. `METHOD` can be: `'nearest'`, `'linear'`, `'spline'`, or `'cubic'`. The default is `'linear'`.

Published with MATLAB® R2014b

LPVSUBS - Evaluate LPV object and demote to LTI

Contents

Syntax

```
B = lpvsubs(A,NAME,VALUES)
B = lpvsubs(A,NAME,VALUES,METHOD)
```

Description

`lpvsubs` evaluates a grid-based LPV object at points in the domain, and demotes the results to a standard LTI object `double`, `ss`, or `frd`.

`B = lpvsubs(A,NAME,VALUES,METHOD)` evaluates a grid-based LPV object A at the domain points specified by the NAME / VALUES pair. NAME is an N-by-1 cell array of characters. NAME must contain all of the independent variable names in A, namely `A.IVName`, but may also contain others but they will be ignored. VALUES is an N-by-NPTS double array of the corresponding values. B is a double array, with row and column dimensions from A. The 3rd dimension of B is NPTS. METHOD is an optional input that specifies the interpolation method and can be: `'nearest'`, `'linear'`, `'spline'`, or `'cubic'`. The default is `'linear'`.

Published with MATLAB® R2014b

LPVELIMIV - Eliminate parameters which only have a single grid point

Contents

Syntax

```
E = lpvelimiv(M)
```

Description

`E = lpvelimiv(M)` eliminates all singleton independent variables from the grid-based LPV object `M`, i.e. the i^th independent variable of `M` is removed if `M.IVData{i}` has length one.

Published with MATLAB® R2014b

LPVSAMPLE - Sample a LPV object

Contents

Syntax

```
S=lpvsample(G,N)
S=lpvsample(G,N,OPT)
```

Description

If `R` is an `rgrid`, then `S=LPVSAMPLE(R,N)` returns a sample of `N` points from the rectangular grid `R`. `S` is an Niv-by-N matrix with each column containing one sample of the rectangular grid. The rows of `S` correspond to the ordering of independent variables in `R.IVName`.

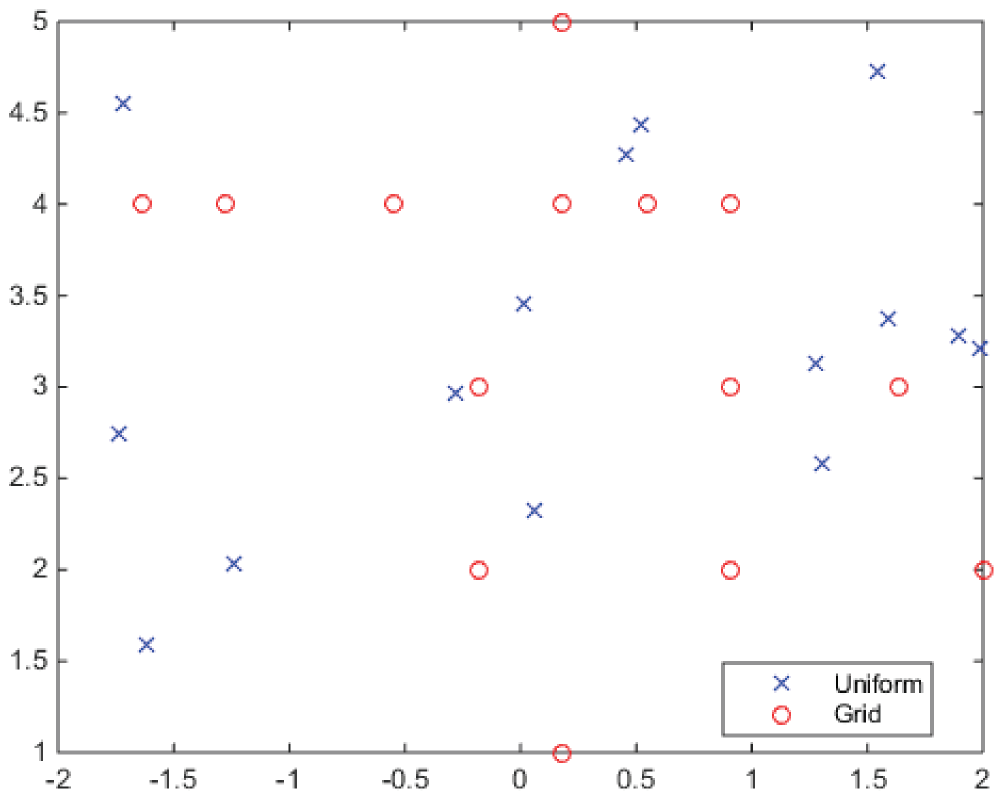
Otherwise, if `G` is a `pmat`, `pss`, `pfrd`, `upmat`, `upss` or `upfrd`, then `S=lpvsample(G,N)` returns `N` samples of the LPV object `G`. Each sample of `G` is generated by evaluating `G` at a randomly chosen point in the domain of `G`. The output `S` is an array of size: [`size(G)`, `N`].

`S=lpvsample(G,N,OPT)` allows the user to specify the sampling algorithm to be used. `OPT` is a `char` that specifies the type of sampling: * `'grid'`: Draws points drawn randomly (possibly with repeats) from the rectangular grid of `G.Domain`. * `'uniform'` (default): Draws points uniformly from the hypercube specified by the limits of `G.Domain`. * `'LHC'`: Does a Latin Hypercube sample of the `G.Domain`.

For `'uniform'` and `'LHC'`, the samples are not, in general, elements of the rectangular grid.

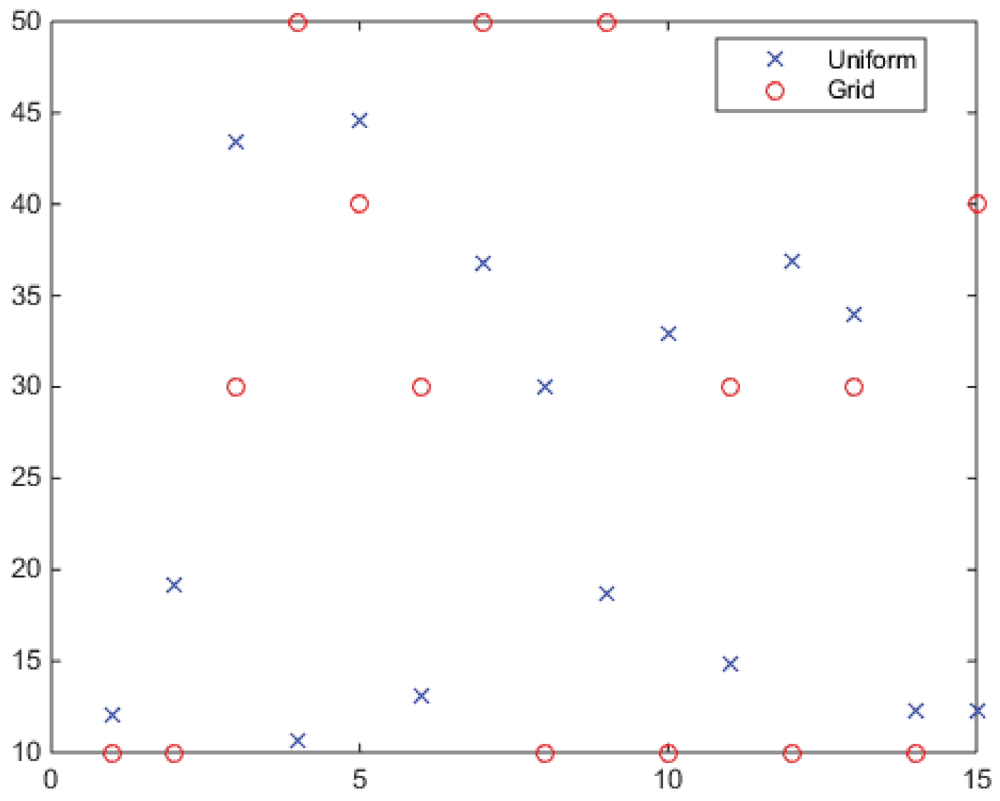
Example: Sample a 2-dimensional `rgrid` object

```
R = rgrid( {'a', 'b'}, {linspace(-2,2,12), 1:5} );
Su = lpvsample(R,15);   % Uniform sample
Sg = lpvsample(R,15,'grid');   % Sample from grid
plot(Su(1,:),Su(2,:),'bx',Sg(1,:),Sg(2,:),'ro');
legend('Uniform','Grid','Location','Best')
```



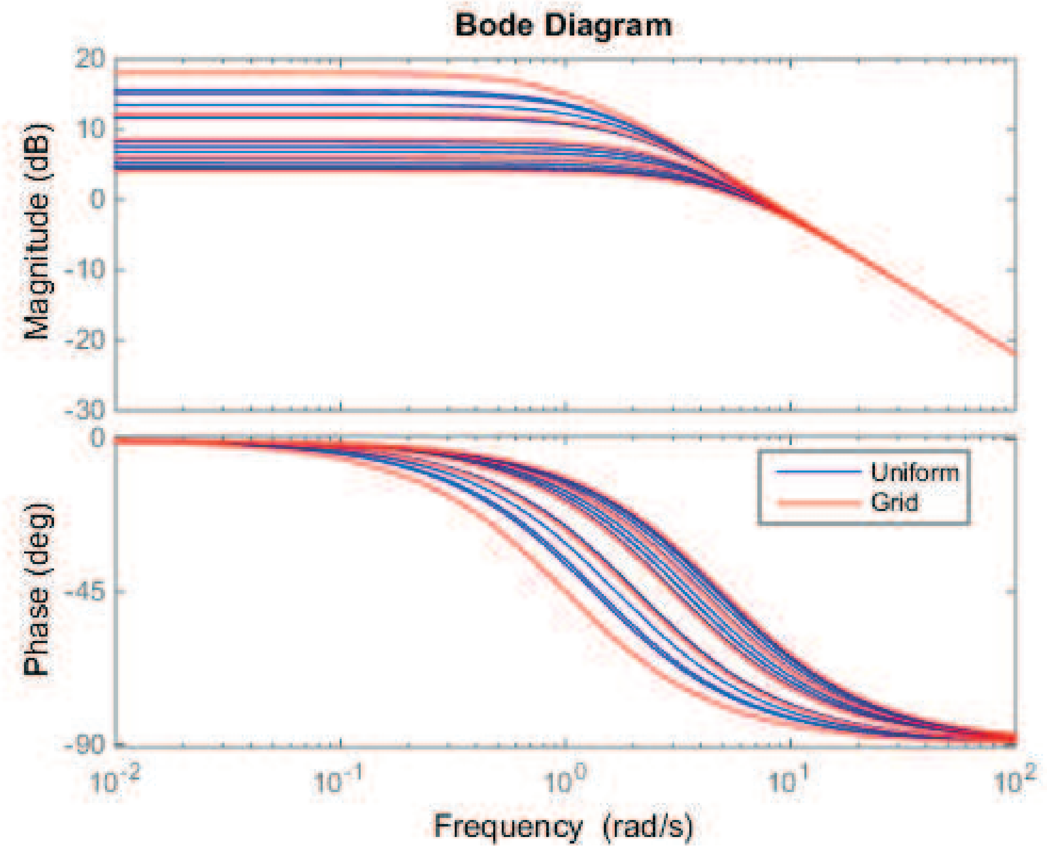
Example: Sample a 2-by-2 `pmat` object

```
a = pgrid('a',1:5);
M = 10*a;
Su = lpvsample(M,15);   % Uniform sample
Sg = lpvsample(M,15,'grid');   % Sample from grid
plot(1:15,Su(:),'bx',1:15,Sg(:),'ro')
legend('Uniform','Grid','Location','Best')
```



Example: Sample a 1-by-1 `pss` object

```
a = pgrid('a',1:5);
M = ss(-a,2,4,0);
Su = lpvsample(M,15);   % Uniform sample
Sg = lpvsample(M,15,'grid');   % Sample from grid
bode(Su,'b',Sg,'r')
legend('Uniform','Grid','Location','Best')
```



Published with MATLAB® R2014b

LPVBALANCE - Diagonal scaling of a `pmat` or `pss`

Contents

Syntax

```
[B,D] = lpvbalance(A)
[B,DR,DC] = lpvbalance(A,BLK)
```

Description

`lpvbalance` computes a diagonal scaling for LPV objects to improve their numerical conditioning. The algorithm used to accomplish this uses a generalized version of Osborne's iteration.

`lpvbalance` for `pmat`

`[B,D] = lpvbalance(A)` computes a single diagonal similarity transformation to improve the conditioning of the N-by-N `pmat` A at all points in the domain. The transformation D is computed using a generalized version of Osborne's iteration. D is returned as an N-by-N diagonal, `double` matrix. The scaled `pmat` B is `D*A*inv(D)`. This applies the transformation D at each point in the domain of A.

`[B,DR,DC] = lpvbalance(A,BLK)` computes structured, diagonal transformations for the N-by-M `pmat` A. The scaled `pmat` B is DR`*`A`*`DC where DR is an N-by-N matrix and and DC is an M-by-M matrix. BLK is a K-by-2 matrix that specifies the block partitioning dimensions of DR and DC. If `BLK = [c1 r1; ... ; ck rk]` then DR and DC are partitioned as:

```
DR = blkdiag( d1*I_r1, ...,  dk*I_rk )
DC = blkdiag( (1/d1)*I_c1, ...,  (1/dk)*I_ck )
```

where the notation `I_r1= eye(r1)`, represents the r1-by-r1 identity matrix. The block partitioning must be consistent with the dimensions of A, i.e. the sum across the rows of BLK should equal [M N].

`lpvbalance` for `pss`

`[P,D] = lpvbalance(S)` computes a single diagonal similarity transformation to improve the conditioning of the Ny-by-Nu `pss` S at all points in the domain. The transformation D is computed using a generalized version of Osborne's iteration. D is returned as an Nx-by-Nx diagonal, double matrix where Nx is the state dimension of S. The scaled `pss` P is obtained by applying the similarity transformation D at each point in the domain of S.

`[P,DR,DC] = lpvbalance(S,BLK)` computes diagonal transformations applied to the states and input/output channels. The state matrices of the scaled `pss` P are obtained from DR`*[A B; C D]*`DC where A, B, C, D are the state matrices of S. BLK is a K-by-2 matrix that specifies the partitioning dimensions of DR and DC. If `BLK = [c1 r1; ... ; ck rk]` then DR and DC are partitioned as:

```
DR = blkdiag( D, d1*I_r1, ...,  dk*I_rk )
DC = blkdiag( inv(D), (1/d1)*I_c1, ...,  (1/dk)*I_ck )
```

where D is a diagonal, Nx-by-Nx matrix, and the notation `I_r1= eye(r1)`, represents the r1-by-r1 identity matrix. The block partitioning must be consistent with the input/output dimensions of S, i.e. the sum across the rows of BLK should equal [Nu Ny].

Published with MATLAB® R2014b

LPVPLOT - Plot PMAT data as a function of the independent variable.

Contents

Syntax

```
lpvplot(PMAT)
lpvplot(PLOT_TYPE,PMAT1,PMAT2,PMAT3, ...)
lpvplot(PLOT_TYPE,PMAT1,'linetype1',PMAT2,'linetype2',...)
```

Description

`lpvplot` plots the data contained in a `pmat` against the independent variable it depends on. The syntax is identical to the `uplot` command in the Robust Control Toolbox, except the data are `pmat`.

`lpvplot(PLOT_TYPE,PMAT1,PMAT2,PMAT3, ..., PMATN)` plots the value of `PMAT1`, `PMAT2`, `PMAT3`, ..., `PMATN` against the independent variable. `PMAT1`, `PMAT2`, `PMAT3`, ... can only depend on a single independent variable, and they must all depend on the same independent variable.

The (optional) PLOT_TYPE argument must be one of:

- `'iv,d'` matin .vs. independent variable (default option)
- `'iv,m'` magnitude .vs. independent variable
- `'iv,lm'` log(magnitude) .vs. independent variable
- `'iv,p'` phase .vs. independent variable
- `'liv,d'` matin .vs. log(independent variable)
- `'liv,m'` magnitude .vs. log(independent variable)
- `'liv,lm'` log(magnitude) .vs. log(independent variable)
- `'liv,p'` phase .vs. log(independent variable)
- `'nyq'` real .vs. imaginary (parametrized by indep variable)
- `'nic'` Nichols chart
- `'bode'` Bode magnitude and phase plots

`lpvplot(PLOT_TYPE,PMAT1,'linetype1',PMAT2,'linetype2',...)` enables the user to set the linetype for the data associated with each `pmat` input.

Published with MATLAB® R2014b

LPVGRAM - Compute Gramians for PSS objects

Contents

Syntax

```
Wc = LPVGRAM(SYS,'c')
Wo = LPVGRAM(SYS,'o')
W = LPVGRAM(SYS,OPTION,WEIGHT)
W = LPVGRAM(SYS,...,INVERT)
```

Description

`Wc = LPVGRAM(SYS,'c')` computes the controllability gramian of the `pss sys`. The output `Wc` is a constant `double` matrix, which satisfies the LMI: `A*Wc+Wc*A' +B*B' < 0` at each point in the domain of `SYS`, where `A` is the state matrix of `SYS` and `B` is its input matrix.

`Wo = LPVGRAM(SYS,'o')` computes the observability gramian of the `pss SYS`. The output `Wo` is a constant `double` matrix, which satisfies the LMI: `A'*Wo+Wo*A +C'*C < 0` at each point in the domain of `SYS`, where `A` is the state matrix of `SYS` and `C` is its output matrix.

`W = LPVGRAM(SYS,OPTION,WEIGHT)` applies a matrix weighting `WEIGHT` when solving for the gramian. For a controllabilty gramian the LMI becomes:

```
A*WEIGHT*Wc+Wc*WEIGHT*A' +B*B' < 0
```

For a observability gramian the LMI becomes:

```
A'*WEIGHT*Wo+Wo*WEIGHT*A +C'*C < 0
```

If no `WEIGHT` is specified a default value of `eye(size(A))` is used, and the resulting gramian is diagonal.

`W = LPVGRAM(SYS,...,INVERT)` provides an alternative implementation of the algorithm which solves for the gramians. If `INVERT` is `True` the LMI conditions are changed to solve for the inverse of the gramians, which can improve the accuracy of the solution for certain systems. The default implementation assumes `INVERT=FALSE`.

Published with MATLAB® R2014b

LPVBALREAL - Gramian-based balancing for PSS objects

Contents

Syntax

```
SYSB = LPVBALREAL(SYS)
[SYSB,G] = LPVBALREAL(SYS)
[SYSB,G,T,Ti] = BALREAL(SYS)
[SYSB,G,T,Ti] = LPVBALREAL(SYS,...,INVERSE)
```

Description

`SYSB = LPVBALREAL(SYS)` computes a balanced realization of the parameter varying system `SYS`. `SYSB` is dervied by computing a single balancing transformation for `SYS` and applying it at every point in its domain.

`[SYSB,G] = LPVBALREAL(SYS)` returns `G`, a vector of singular values describing the input-output mapping of `SYSB` (comparable to Hankel singular values for LTI systems).

`[SYSB,G,T,Ti] = BALREAL(SYS)` returnes the balancing transformation `T`, and its inverse `Ti`.

`[SYSB,G,T,Ti] = LPVBALREAL(SYS,...,INVERSE)` provides an option to use a alternative implementation of the algorithm which computes the balancing transformation. If `INVERT` is True the alternative formulation is used. It can improve the accuracy of the solution for certain systems. The default implementation assumes `INVERT=FALSE`.

Published with MATLAB® R2014b

LPVBALANCMR - Balanced truncation of quadratically stable `pss` models

Contents

Syntax

```
[PRED,INFO] = lpvbalancmr(SYS,N)
[PRED,INFO] = lpvbalancmr(SYS,N,OPTION1,VAL1,OPTION2,VAL2,...)
```

Description

`[PRED,INFO] = lpvbalancmr(SYS,N)` performs balanced truncation model reduction on a PSS SYS. A `pss` `SYS`, with Nx states, is balanced by computing a single balancing transformation for SYS and applying it at every point in its domain. The output PRED has N states and is obtained by truncating from the balanced system the `(Nx-N)` states which contribute least to its input-output mapping. INFO contains two fields `'StabSV'` and `'ErrorBound'`. `INFO.StabSV` is a vector of singular values describing the input-output mapping of SYSB (comparable to Hankel singular values for LTI systems). `INFO.ErrorBound` contains the L_2 norm of the difference between SYS and PRED: `INFO.ErrorBound` = induced L_2 norm of SYS - PRED.

Note that `lpvbalancmr` only works for quadratically stable systems. For unstable `pss` models use `lpvncfmr`.

`[PRED,INFO] = lpvbalancmr(SYS,N,OPTION1,VAL1,OPTION2,VAL2,...)` provides additional options for the balanced truncation model reduction. The current implementation supports the following options:

OPTION	VAL	Explanation
'weight'	{Wout,Win}	LTI weights on input (Win) and output (Wout). Used to emphasize accuracy in different I/O and frequency ranges. Must be invertable if method 'invgram' is used.
'method'	'gram' or 'invgram'	Solve using either gramians or the inverse gramians.

Published with MATLAB® R2014b

LPVNCFMR - Contractive coprime factor model reduction of a `pss`

Contents

Syntax

```
[Pred,INFO] = lpvncfmr(P)
[Pred,INFO] = lpvncfmr(P,ORDER)
```

Description

`lpvncfmr` performs balanced truncation model reduction through contractive coprime factorization of a `pss`.

`[Pred,INFO] = lpvncfmr(P,ORDER)` finds a balanced contractive coprime factorization of the LPV system `P` (analogous to a normalized coprime factorization for LTI systems), and performs a balanced truncation to remove those states that contribute the least to the input-output mapping of the balanced LPV system. If `P` has M states, then the reduced order system Pred will have `ORDER` states, with M - `ORDER` states removed using the balanced truncation. `INFO.hsv` contains a vector of singular values describing the input-output mapping of `SYSB` (comparable to Hankel singular values for LTI systems).

`[Pred,INFO] = lpvncfmr(P)` computes the balanced contractive coprime factorization of the LPV system `P`, and outputs it as `Pred`. This is equivalent to the call `[Pred,INFO] = LPVNCFMR(P,Nx)` for a system `P` with `Nx` states.

Published with MATLAB® R2014b

LPVNORM - Compute bound on norm for `pss` systems.

Contents

Syntax

```
[Gamma,X] = lpvnorm(P)
[Gamma,X] = lpvnorm(P,'L2')
[Gamma,X] = lpvnorm(P,'LQG')
[Gamma,X] = lpvnorm(P,Xb)
[Gamma,X] = lpvnorm(P,Xb,'L2')
[Gamma,X] = lpvnorm(P,Xb,'LQG')
```

Description

`lpvnorm` computes a bound on the norm of a `pss` system over the set of all permissible trajectories of the independend variables which the `pss` depends on.

`[Gamma,X] = lpvnorm(P,'L2')` computes an upper bound `Gamma` on the induced L_2 norm of the `pss` `P`. The upper bound `Gamma` and a constant (parameter independent) matrix `X` are computed to satisfy the induced norm linear matrix inequality (LMI) condition. `X` is returned as a `pmat`. The L_2 norm bound is valid for arbitrarily fast variations of the system parameters.

`[Gamma,X] = lpvnorm(P,'LQG')` computes an upper bound `Gamma` on the stochastic LPV bound. The stochastic LPV bound is defined as the expected value of the average instantaneous power of the output of `P`, assuming its inputs are zero mean, white-noise processes with unit intensity.

`[Gamma,X] = lpvnorm(P,Xb,ALG)` computes a tighter (less conservative) bound on the norm by using a parameter dependent matrix `X` = $X(\rho)$ and bounds on the parameter rates of variation. The basis functions used to construct $X(\rho)$ are specified with the `basis` object `Xb`. `ALG` can be either `'L2'` or `'LQG'`. A call without the `ALG` argument is equivalent to `[Gamma,X] = lpvnorm(P,Xb,'L2')`.

Published with MATLAB® R2014b

LPVWCGAIN - Worst-case bound on induced L2 norm for `pss` and `plftss`

Contents

Syntax: `lpvwcgain` for `pss`

```
[GAM,X,INFO] = lpvwcgain(P)
[GAM,X,INFO] = lpvwcgain(P,Xb)
[GAM,X,INFO] = lpvwcgain(P,...,POLELIST)
```

Syntax: `lpvwcgain` for `plftss`

```
[WCG,WCU,INFO] = lpvwcgain(P)
[WCG,WCU,INFO] = lpvwcgain(P,OMEGA)
[WCG,WCU,INFO] = lpvwcgain(P,...,POLELIST)
```

Description

`lpvwcgain` computes the worst-case bound on the induced L_2 norm for `pss` and `plftss` systems. The grid based and LFT objects have minor differences in syntax, explained below.

`lpvwcgain` for `pss`

`[GAM,X,INFO] = lpvwcgain(P)` computes the upper-bound on the worst-case induced L_2 norm of the LPV system P, assuming no rate-bounds on the independent variables of P. "Worst-case" refers to all the modeled uncertainty and parameter-dependence (including parameter rate-bounds). `GAM` is the upper bound on the worst-case induced L_2 norm. The upper bound `GAM` and a constant (parameter independent) matrix `X` are computed to satisfy the induced L_2 norm linear matrix inequality (LMI) condition.

`[GAM,X,INFO] = lpvwcgain(P,Xb)` computes the upper-bound on the worst-case induced L_2 norm of the PSS P. The upper bound `GAM` and a parameter-varying matrix `X` are computed to satisfy the induced L_2 norm linear matrix inequality (LMI) condition. `Xb` is a `basis` object that defines the basis functions which describe the assumed parameter dependence of `X`. `INFO` is a structure containing additional information about the solution to the LMI.

`lpvwcgain` for `plftss`

`[WCG,WCU,INFO] = lpvwcgain(P)` computes the upper-bound on the worst-case induced L_2 norm of the `plftss` P. "Worst-case" refers to all the modeled uncertainty and parameter-dependence. `WCG` is the upper bound on the worst-case induced L_2 norm. `INFO` is a structure containing additional information about the solution, including an estimate of the lower bound on the worst-case induced L_2 norm, based on LTI worst-case gain analysis. `WCU` is value of the uncertainty associated with the lower-bound of the induced L_2 norm, which is based on LTI worst-case gain analysis.

`[WCG,WCU,INFO] = lpvwcgain(P,OMEGA)` allows the user to specify a custom frequency vector for the analysis. `OMEGA` is the chosen vector of frequency values used in the analysis.

lpvwcgain(P,...,POLELIST) allows the user to define weighting functions for the Integral Quadratic Constraints (IQC) used to bound the uncertainty when formulating the LMI to be solved. `POLELIST` is a 1xN `double` row vector, of negative values. Each value in `POLELIST` corresponds to a pole of a stable transfer function that is used as a weight on all signals in the IQCs. A default `POLELIST`, with three pole values, is used when a `POLELIST` is not supplied by the user. The three pole values are selected automatically from the frequency range of the system dynamics.

Published with MATLAB® R2014b

LPVSYN - Parameter-dependent controller synthesis for LPV systems

Contents

Syntax

```
[K,GAM,INFO] = lpvsyn(P,NMEAS,NCON)
[K,GAM,INFO] = lpvsyn(P,NMEAS,NCON,Xb,Yb)
[K,GAM,INFO] = lpvsyn(P,NMEAS,NCON,...,OPT)
```

Description

`[K,GAM,INFO] = lpvsyn(P,NMEAS,NCON)` computes a parameter-varying controller K which minimizes the induced L_2 norm of the interconnection defined by `lft(P,K)`. K is a `pss` or `plftss` with NMEAS inputs and NCON outputs, defined on same domain as P. GAM is the induced L_2 norm of `lft(P,K)`. This three argument call assumes that the rate-bounds of the independent variables in P are `[-inf,inf]`. INFO is a structure containing data from the Linear Matrix Inequalities that are solved to obtain K.

`[K,GAM,INFO] = lpvsyn(P,NMEAS,NCON,Xb,Yb)` computes the rate-bounded parameter-varying controller K for a system P. K is the controller which minimizes the induced L_2 norm of `lft(P,K)` when the rate-bounds of the independent variables of P are incorporated into the synthesis. Xb and Yb are `basis` objects, which describe the assumed parameter dependence of the lyapunov matrices used in solving for K.

`[K,GAM,INFO] = lpvsyn(P,NMEAS,NCON,...,OPT)` allows the user to pass in a `lpvsynoptions` object.

The default algorithm for `lpvsyn` will solve the given synthesis problem twice. The first iteration attempts to find a solution that minimizes the induced L_2 norm of `lft(P,K)`. The second iteration will solve the optimization problem again, with the caveat that any solution that is % found to lie within 15% of the optimal induced L_2 norm of `lft(P,K)` from the first iteration, is satisfactory. This formulation has been found to yield controllers that are better numerically conditioned. The back-off factor of 15% can be reset to a different value in `lpvsynoptions`.

Published with MATLAB® R2014b

LPVNCFSYN - Parameter-dependent Glover-McFarlane loop-shaping for LPV systems

Contents

Syntax

```
[K,CL,GAM,INFO]=lpvncfsyn(G,W1,W2)
[K,CL,GAM,INFO]=lpvncfsyn(G,W1,W2,'ref')
[K,CL,GAM,INFO]=lpvncfsyn(G,W1,W2,Xb,Yb,...)
[K,CL,GAM,INFO]=lpvncfsyn(G,...,OPT)
```

Description

`[K,CL,GAM,INFO]=lpvncfsyn(G,W1,W2)` synthesizes a parameter-dependent Glover-McFarlane loop-shaping controller K for the shaped plant `Gs=W2*G*W1`.

```
                 ^ z1       ^ z2
      ____       |   ____   |        ____    ____
   ->| W2 |-> + ---->| Ks |--> + -->| W1 |--| G  |---
+ |   ----    ^       ----     ^     ----    ----    |
  |           |                |                     |
  |        w1 |            w2 |                     |
  |__________________________________________________|
```

`G` is a `pss`, while `W1` and `W2` can be `pss`, `pmat` or `double`. K is a parameter-varying controller which minimizes the induced L_2 norm of the loop-shaping interconnection defined by `G`, `W1` and `W2`. K is a `pss` defined on same domain as `P`. K has as many inputs as `G` has outputs, and as many outputs as `G` has inputs. `GAM` is the induced L_2 norm of the loop-shaping interconnection. `CL` is the system taking in [w1; w2] and outputting [z1; z2]. `INFO` is a structure containing data from the Linear Matrix Inequalities that are solved to obtain `K`. A call to `lpvncfsyn` without a `basis` function argument generates a controller assuming no bounds on the parameter rate of variation.

`[K,CL,GAM,INFO]=lpvncfsyn(G,W1,W2,'ref')` synthesizes a parameter-dependent Glover-McFarlane loop-shaping controller K for the shaped plant `Gs=W2*G*W1`, assuming a reference command. `CL` is the system taking in [w1; w2; ref] and outputting [z1; z2], and `GAM` is its induced L_2 norm.

```
                       ^ z1         ^ z2
          ____         |   ____     |     ____    ____
   ----->| W2 |-> + ----->|    |--> + -->| W1 |--| G  |---
+ |       ----    ^       | Ks }      ^   ----    ----    |
  |               |   --->|    |      |                   |
  |            w1 |   ref  ----    w2 |                   |
  |                                                       |
  |_______________________________________________________|
```

`[K,CL,GAM,INFO]=lpvncfsyn(G,W1,W2,Xb,Yb,...)` computes the rate-bounded Glover-McFarlane loop-shaping controller `K` where the rate-bounds of the independent variables of the shaped plant `Gs` are incuded in the synthesis conditions. `Xb` and `Yb` are `basis` objects, which describe the assumed parameter dependence of the lyapunov matrices used in solving for `K`.

`[K,CL,GAM,INFO]=lpvncfsyn(G,...,OPT)` allows the user to pass in a `lpvsynoptions` object.

Published with MATLAB® R2014b

LPVMIXSYN - Parameter-varying mixed-sensitivity synthesis

Contents

Syntax

```
[K,GAM,INFO]=lpvmixsyn(G,W1,W2,W3)
[K,GAM,INFO]=lpvmixsyn(G,W1,W2,W3,Xb,Yb)
[K,GAM,INFO]=lpvmixsyn(G,...,OPT)
```

Description

`[K,GAM,INFO]=lpvmixsyn(G,W1,W2,W3)` synthesizes the parameter-varying mixed-sensitivity controller `K`, which minimizes the induced L_2 norm of `W1*S`, `W2*K*S` and `W3*T`, where `S = inv(I+G*K)`, `T = G*K*inv(I+G*K)`, and `W1`, `W2` and `W3` are stable `pss`, `pmat` or `double` weights of appropriate size. `GAM` is the induced L_2 norm acheived by `K`. `INFO` is a structure containing data from the Linear Matrix Inequalities that are solved to obtain `K`. A call to `lpvmixsyn` without a `basis` function argument generates a controller assuming no bounds on the parameter rate of variation.

`[K,GAM,INFO]=lpvmixsyn(G,W1,W2,W3,Xb,Yb)` computes the rate-bounded mixed-sensitivity controller `K`, where the rate-bounds of the independent variables of `G`, `W1`, `W2` and `W3` are incuded in the synthesis conditions. `Xb` and `Yb` are `basis` objects, which describe the assumed parameter dependence of the lyapunov matrices used in solving for `K`.

`[K,GAM,INFO]=lpvmixsyn(G,...,OPT)` allows the user to pass in a `lpvsynoptions` object.

Published with MATLAB® R2014b

LPVLOOPSHAPE - Parameter-varying loop-shaping synthesis

Contents

Syntax

```
[K,GAM,INFO]=lpvloopshape(G,Wi,Wo)
[K,GAM,INFO]=lpvloopshape(G,Wi,Wo,Xb,Yb)
[K,CL,GAM,INFO]=lpvloopshape(G,...,OPT)
```

Description

`[K,GAM,INFO]=lpvloopshape(G,Wi,Wo)` synthesizes the parameter-varying controller `K`, which minimizes the induced L_2 norm for the shaped plant `Gs=Wo*G*Wi`. `GAM` is the induced L_2 norm of the closed-loop system from [d1,d2] to |[e1,e2].

```
                    ^ e1                          ^ e2
         ____       |     ____     ____     ____  |
  +---->| Ks |-- + ---->| Wi |--| G  |-->| Wo |------->+ -----
- |      ----    ^      ----     ----     ----         ^     |
  |              |                                     |     |
  |           d1 |                                  d2 |     |
  |____________________________________________________________|
```

`INFO` is a structure containing data from the Linear Matrix Inequalities that are solved to obtain `K`. A call to `lpvloopshape` without a `basis` function argument generates a controller assuming no bounds on the parameter rate of variation.

`[K,GAM,INFO]=lpvloopshape(G,Wi,Wo,Xb,Yb)` computes the rate-bounded controller `K`, where the rate-bounds of the independent variables of the shaped pland `Gs` are incuded in the synthesis conditions. `Xb` and `Yb` are `basis` objects, which describe the assumed parameter dependence of the lyapunov matrices used in solving for `K`.

`[K,CL,GAM,INFO]=lpvloopshape(G,...,OPT)` allows the user to pass in a `lpvsynOptions` object.

Published with MATLAB® R2014b

LPVSFSYN - Parameter-dependent state feedback controller synthesis

Contents

Syntax

```
[F,GAM,INFO] = lpvsfsyn(P,NCON)
[F,GAM,INFO] = lpvsfsyn(P,NCON,'L2')
[F,GAM,INFO] = lpvsfsyn(P,NCON,'LQG')
[F,GAM,INFO] = lpvsfsyn(P,NCON,Xb,Yb)
[F,GAM,INFO] = lpvsfsyn(P,NCON,Xb,Yb,'L2')
[F,GAM,INFO] = lpvsfsyn(P,NCON,Xb,Yb,'LQG')
```

Description

`[F,GAM,INFO] = lpvsfsyn(P,NCON,'L2')` computes a parameter-varying state-feedback controller for the parameter-varying system `P`. `NCON` specifies the number of available control inputs in `P`. `F` is the state-feedback controller for the plant `P`, which minimizes the L_2 norm from the input of `P` to its output. `GAM` is the minimum L_2 norm achived by `F`. `INFO` is a struct with additional data.

`[F,GAM,INFO] = lpvsfsyn(P,NCON,'LQG')` computes a parameter-varying state-feedback controller `F`, which minimizes the stochastic LPV bound. The stochastic LPV bound is defined as the expected value of the average instantaneous power of the output of `P`, assuming its inputs are zero mean, white-noise processes with unit intensity.

`[F,GAM,INFO] = lpvsfsyn(P,NCON,Xb,Yb,ALG)` performs a rate-bounded synthesis. `Xb` and `Yb` are `basis` objects specifying the basis functions to be used in the synthesis. `ALG` can be either `'L2'` or `'LQG'`. A call without the `ALG` argument is equivalent to `[F,GAM,INFO] = lpvsfsyn(P,NCON,Xb,Yb,'L2')`.

Published with MATLAB® R2014b

LPVESTSYN - Synthesize a parameter-varying estimator for LPV systems

Contents

Syntax

```
[L,GAM,INFO] = lpvestsyn(P)
[L,GAM,INFO] = lpvestsyn(P,'L2')
[L,GAM,INFO] = lpvestsyn(P,'LQG')
[L,GAM,INFO] = lpvestsyn(P,Yb)
[L,GAM,INFO] = lpvestsyn(P,Yb,'L2')
[L,GAM,INFO] = lpvestsyn(P,Yb,'LQG')
```

Description

`[L,GAM,INFO] = lpvestsyn(P,'L2')` computes a parameter-varying state estimator for the parameter-varying system `P`. `L` takes in all the outputs of `P` and outputs an estimate of the states of `P`. `L` is the constant estimation matrix for the plant `P`, which minimizes the induced L_2 norm of the error in the state-estimate. `GAM` is the minimum L_2 norm achived by `L`. `INFO` is a struct with additional data.

`[L,GAM,INFO] = lpvestsyn(P,'LQG')` computes the constant estimation matrix for the plant `P`, which minimizes the stochastic LPV bound on the state estimation error. The stochastic LPV bound on the state estimation error is defined as the expected value of the average instantaneous power of error signal, assuming system inputs are zero mean, white-noise processes with unit intensity.

`[L,GAM,INFO] = lpvestsyn(P,Yb,ALG)` performs a rate-bounded synthesis. `Yb` is a `basis` object specifying the basis functions to be used in the synthesis. `ALG` can be either `'L2'` or `'LQG'`. A call without the `ALG` argument is equivalent to `[L,GAM,INFO] = lpvestsyn(P,Yb,'L2')`

Published with MATLAB® R2014b

LPVSTOCHSYN - LPV controller synthesis for stochastic LPV systems

Contents

Syntax

```
[K,GAMMA,INFO] = lpvstochsyn(P,NMEAS,NCON)
[K,GAMMA,INFO] = lpvstochsyn(P,NMEAS,NCON,XB,YB)
[K,GAMMA,INFO] = lpvstochsyn(P,NMEAS,NCON,XB,YB,OPT)
```

Description

`[K,GAMMA,INFO] = lpvstochsyn(P,NMEAS,NCON)` computes a parameter-varying controller for the `pss` `P`. `NCON` specifies the number of available control inputs in `P`. `NMEAS` specifies the number of available measurements being output from `P`. `K` is the parameter-dependent controller for the stochastic plant `P`, which minimizes the stochastic LPV bound `GAMMA`. The stochastic LPV bound is defined as the expected value of the average instantaneous power of the output of `P`, assuming its inputs are zero mean, white-noise processes with unit intensity. `INFO` is a struct with additional data.

`[K,GAMMA,INFO] = lpvstochsyn(P,NMEAS,NCON,XB,YB)` performs a rate-bounded synthesis. `Xb` and `Yb` are `basis` objects which describe the assumed parameter dependence of the lyapunov matrices used in solving for `K`.

`[K,GAMMA,INFO] = lpvstochsyn(P,NMEAS,NCON,XB,YB,OPT)` allows the user to pass in a `lpvsynoptions` object.

The default algorithm for `lpvstochsyn` will solve the given synthesis problem twice. The first iteration attempts to find a solution that minimizes the stochastic LPV bound of `lft(P,K)`. The second iteration will solve the optimization problem again, with the caveat that any solution that is found to lie within 15% of the optimal stochastic LPV bound of `lft(P,K)` from the first iteration, is satisfactory. This formulation has been found to yield controllers that are better numerically conditioned. The back-off factor of 15% can be reset to a different value in `lpvsynoptions`.

Published with MATLAB® R2014b

LPVSYNOPTIONS - Create a options object for LPV synthesis and analysis

Contents

Syntax

```
opt = lpvsynoptions
opt = lpvsynOptions(Name1,Value1,Name2,Value2,...)
```

Description

`opt = lpvsynOptions(Name1,Value1,Name2,Value2,...)` creates a options object for parameter-varying synthesis and analysis. The `lpvsynOptions` object is used to specify the parameters of the optimization routines used in the synthesis and analysis functions: `lpvsyn`, `lpvmixsyn`, `lpvncfsyn`, `lpvloopshapesyn`, `lpvestsyn`, `lpvsfsyn`, `lpvnorm`, `lpvstochsyn`, `lpnvnorm`, `lpvwcgain`.

`opt = lpvsynOptions` creates an `lpvsynOptions` object initialized with default values.

The options are set using `NAME`, `VALUE` pairs, i.e. the options property specified by the character string `NAME` is set to `VALUE`. The setable options properties are specified below. The default choice is specified in brackets.

```
%--------------------------------------------------------------------------
%     NAME          |   VALUE     |    Description
%--------------------------------------------------------------------------
%   'Solver'        | ['lmilab']  | Optimization solver to be used.
%   ------------------------------------------------------------------------
%   'SolverOptions' | []          | Options passed directly to the solver.
%   ------------------------------------------------------------------------
%   'SolverInit'    | []          | Initial decision variables for LMI solver.
%   ------------------------------------------------------------------------
%   'Gammalb'       | [1e-6]      | Lower bound on closed-loop induced L2 norm.
%   ------------------------------------------------------------------------
%   'Gammaub'       | [1e6]       | Upper bound on closed-loop induced L2 norm.
%   ------------------------------------------------------------------------
%   'Xlb            | [1e-6]      | X Riccati variable lower bounded by Xlb*I
%   ------------------------------------------------------------------------
%   'Xub            | [1e6]       | X Riccati variable upper bounded by Xub*I
%   ------------------------------------------------------------------------
%   'Ylb            | [1e-6]      | Y Riccati variable lower bounded by Ylb*I
%   ------------------------------------------------------------------------
%   'Yub            | [1e6]       | Y Riccati variable upper bounded by Yub*I
%   ------------------------------------------------------------------------
%   'Method         | ['BackOff'] | String specifying the solution method:
%                   |             |----------------------------------------
%                   | 'MinGamma'  | Minimize the closed-loop induced L2 norm.
%                   |             |----------------------------------------
%                   | 'MaxFeas'   | Maximize the feasibility of the X Riccati
%                   |             | subject to contraints Gammalb and Gammaub
%                   |             | on the closed-loop induced L2 norm.
%                   |             |----------------------------------------
%                   | 'BackOff'    | First solve the 'MinGamma' problem for
%                   |              | GammaOpt and then solve a second stage
%                   |              | 'MaxFeas' problem with
%                   |              | Gammaub = BackOffFactor*GammaOpt.
%                   |              | This two-stage solution improves
%                   |              | the numerical conditioning of the
%                   |              | controller reconstruction.
%                   |              |----------------------------------------
%                   | 'PoleCon'    | Constrain the closed-loop poles.
%   -----------------------------------------------------------------------
%   'BackOffFactor' | [1.2]        | Multiplicative factor ( >= 1 ) to back off
%                   |              | the minimum gamma when Method = 'BackOff'.
%   -----------------------------------------------------------------------
```

Published with MATLAB® R2014b

LPVLSIM - Simulate the time response of a LPV system

Contents

Syntax

```
[Y,T,X,U,TRAJ] = lpvlsim(G,PTRAJ,UIN,TIN)
[Y,T,X,U,TRAJ] = lpvlsim(G,PTRAJ,UIN,TIN,X0)
```

Description

`[Y,T,X,U,TRAJ] = lpvlsim(G,PTRAJ,UIN,TIN)` simulates the time-response of the system `G`, subject to the input signal defined by `UIN` and `TIN`, and the parameter tracjetory defined in `PTRAJ`. `G` is a `pss` with `Ny` outputs, `Nx` states, `Nu` inputs, and `N` independent variables `IVName1,...,IVNameN`. `TIN` is a sorted column vector of time values, and `UIN` is a `length(TIN)-by-Nu` matrix of corresponding inputs. `PTRAJ` is a struct which defines the time-variation of the parameters (independent variables) in `G`. The field `PTRAJ.time` contains a sorted row vector of time-values. `PTRAJ` must also have a field for each independend variable in `G`, such that `PTRAJ.IVName1, ... ,PTRAJ.IVName` each contain a row vector of parameter trajectories corresponding to `PTRAJ.time`. `Y` is a `length(T)-by-NY` matrix whose columns correspond to the outputs of `G`, `X` is a `length(T)-by-Nx` matrix whose columns correspond to the state trajectories of `G`, `U` is a `length(T)-by-Nu` matrix whose columns correspond to the inputs of `G`, and `T` is a column vector of time values corresponding to `Y`, `X` and `U`. `TRAJ` contains the corresponding parameter trajectories.

`[Y,T,X,U,TRAJ] = lpvlsim(G,PTRAJ,UIN,TIN,X0)` simulates the time-response of the system `G` starting from the initial condition `X0`.

Published with MATLAB® R2014b

LPVSTEP - Parameter dependent step response for LPV systems

Contents

Syntax

```
lpvstep(SYS,PTRAJ)
[Y,T,X,U,PTRAJOUT] = lpvstep(SYS,PTRAJ)
[Y,T,X,U,PTRAJOUT] = lpvstep(SYS,ptraj,TFINAL)
[Y,T,X,U,PTRAJOUT] = lpvstep(SYS,ptraj,T)
```

Description

`[Y,T,X,U,PTRAJOUT] = lpvstep(SYS,PTRAJ)` computes the parameter dependent step response of `SYS`. `SYS` is a LPV system with `Ny` outputs, `Nx` states, `Nu` inputs, and `N` independent variables `IVName1,...,IVNameN`. `PTRAJ` is a struct which defines the time-variation of the parameters (independent variables) in `SYS`. The field `PTRAJ.time` contains a sorted row vector of time-values. `PTRAJ` must also have a field for each independend variable in `SYS`, such that `PTRAJ.IVName1, ... ,PTRAJ.IVNameN` each contain a row vector of parameter trajectories corresponding to `PTRAJ.time`. The output Y is a `length(T)-by-NY-by-Nu` matrix such that `Y(:,i,j)` corresponds to the i-th output of `SYS` due to a step command in the j-th input channel. Similarly `X` is a `length(T)-by-Nx-by-Nu` matrix describing the state trajectories of `SYS`, `U` is a `length(T)-by-Nu-by-Nu` matrix describing the trajectory of the inputs to `SYS`, and `T` is a column vector of time values corresponding to `Y`, `X` and `U`. `PTRAJOUT` contains the corresponding parameter trajectories.

`lpvstep(SYS,PTRAJ)` generates plots of the parameter dependent step response of `SYS`.

`[Y,T,X,U,PTRAJOUT] = lpvstep(SYS,ptraj,TFINAL)` simulates the step response up to the time `TFINAL`.

`[Y,T,X,U,PTRAJOUT] = lpvstep(SYS,ptraj,T)` simulates the step response using a user supplied time vector `T`.

Published with MATLAB® R2014b

LPVINITIAL - Parameter dependent step response for LPV systems

Contents

Syntax

```
lpvinitial(SYS,PTRAJ,X)
[Y,T,X,U,PTRAJOUT] = LPVINITIAL(SYS,PTRAJ,XO)
[Y,T,X,U,PTRAJOUT] = LPVINITIAL(SYS,ptraj,TFINAL)
[Y,T,X,U,PTRAJOUT] = LPVINITIAL(SYS,ptraj,T)
```

Description

`[Y,T,X,U,PTRAJOUT] = LPVINITIAL(SYS,PTRAJ,XO)` computes the parameter dependent response of `SYS` to an initial value `X0`. `SYS` is a LPV system with `Ny` outputs, `Nx` states, `Nu` inputs, and `N` independent variables `IVName1,...,IVNameN`. `PTRAJ` is a struct which defines the time-variation of the parameters (independent variables) in `SYS`. The field `PTRAJ.time` contains a sorted row vector of time-values. `PTRAJ` must also have a field for each independend variable in `SYS`, such that `PTRAJ.IVName1, ... ,PTRAJ.IVNameN` each contain a row vector of parameter trajectories corresponding to `PTRAJ.time`. The output `Y` is a `length(T)-by-NY-by-Nu` matrix such that `Y(:,i,j)` corresponds to the i-th output of `SYS` due to a step command in the j-th input channel. Similarly `X` is a `length(T)-by-Nx-by-Nu` matrix describing the state trajectories of `SYS`, `U` is a `length(T)-by-Nu-by-Nu` matrix describing the trajectory of the inputs to `SYS`, and `T` is a column vector of time values corresponding to `Y`, `X` and `U`. `PTRAJOUT` contains the corresponding parameter trajectories.

`lpvinitial(SYS,PTRAJ,X)` generates plots of the parameter dependent response of `SYS` to an initial value `X0`.

`[Y,T,X,U,PTRAJOUT] = LPVINITIAL(SYS,ptraj,TFINAL)` simulates the response of `SYS` to an initial value `X0` up to the time `TFINAL`.

`[Y,T,X,U,PTRAJOUT] = LPVINITIAL(SYS,ptraj,T)` simulates the response of `SYS` to an initial value `X0` using a user supplied time vector `T`.

Published with MATLAB® R2014b

LPVIMPULSE - Parameter dependent impulse response for LPV systems

Contents

Syntax

```
lpvimpulse(SYS,PTRAJ)
[Y,T,X,U,PTRAJOUT] = LPVIMPULSE(SYS,PTRAJ)
[Y,T,X,U,PTRAJOUT] = lpvimpulse(SYS,ptraj,TFINAL)
[Y,T,X,U,PTRAJOUT] = lpvimpulse(SYS,ptraj,T)
```

Description

`[Y,T,X,U,PTRAJOUT] = lpvimpulse(SYS,PTRAJ)` computes the parameter dependent impulse response of `SYS`. `SYS` is a LPV system with `Ny` outputs, `Nx` states, `Nu` inputs, and `N` independent variables `IVName1,...,IVNameN`. `PTRAJ` is a struct which defines the time-variation of the parameters (independent variables) in `SYS`. The field `PTRAJ.time` contains a sorted row vector of time-values. `PTRAJ` must also have a field for each independend variable in `SYS`, such that `PTRAJ.IVName1, ... ,PTRAJ.IVNameN` each contain a row vector of parameter trajectories corresponding to `PTRAJ.time`. The output `Y` is a `length(T)-by-NY-by-Nu` matrix such that `Y(:,i,j)` corresponds to the i-th output of `SYS` due to a step command in the j-th input channel. Similarly `X` is a `length(T)-by-Nx-by-Nu` matrix describing the state trajectories of `SYS`, `U` is a `length(T)-by-Nu-by-Nu` matrix describing the trajectory of the inputs to `SYS`, and `T` is a column vector of time values corresponding to `Y`, `X` and `U`. `PTRAJOUT` contains the corresponding parameter trajectories.

`lpvimpulse(SYS,PTRAJ)` generates plots of the parameter dependent impulse response of `SYS`.

`[Y,T,X,U,PTRAJOUT] = lpvimpulse(SYS,ptraj,TFINAL)` simulates the impulse response up to the time `TFINAL`.

`[Y,T,X,U,PTRAJOUT] = lpvimpulse(SYS,ptraj,T)` simulates the impulse response using a user supplied time vector `T`.

Published with MATLAB® R2014b

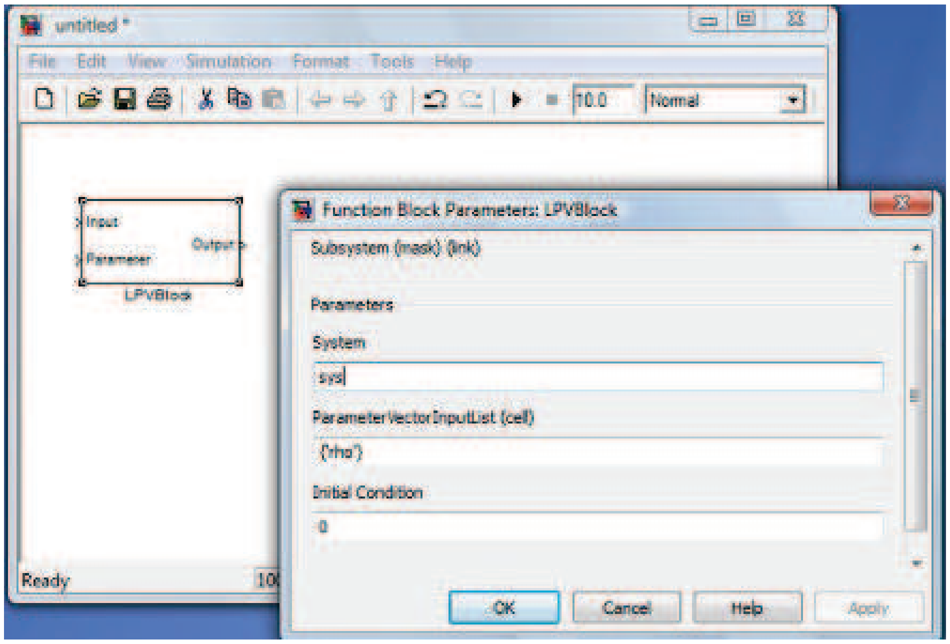
Simulink Blocks in LPVTools

LPVTools provides Simulink blocks to interface to the state-space LPV objects: `pss`, `upss` and `plftss`. The Simulink blocks enable users to include LPV systems in Simulink simulation models. One Simlink block is for systems that depend on a time-varying parameter and its derivative, as seen in Equation 1, while the other is for systems that do not depend explicitly on the derivative, as seen in Equation 2.

$$\begin{bmatrix} \dot{x}(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} A(\rho(t),\dot{\rho}(t)) & B(\rho(t),\dot{\rho}(t)) \\ C(\rho(t),\dot{\rho}(t)) & D(\rho(t),\dot{\rho}(t)) \end{bmatrix} \begin{bmatrix} x(t) \\ u(t) \end{bmatrix} \qquad (1)$$

$$\begin{bmatrix} \dot{x}(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} A(\rho(t)) & B(\rho(t)) \\ C(\rho(t)) & D(\rho(t)) \end{bmatrix} \begin{bmatrix} x(t) \\ u(t) \end{bmatrix} \qquad (2)$$

The Simulink block for the system shown in Equation 2 is shown in Figure 1.



Function Block Parameters: LPVBlock

Subsystem (mask) (link)

Parameters

System

sys

ParameterVectorInputList (cell)

{'rho'}

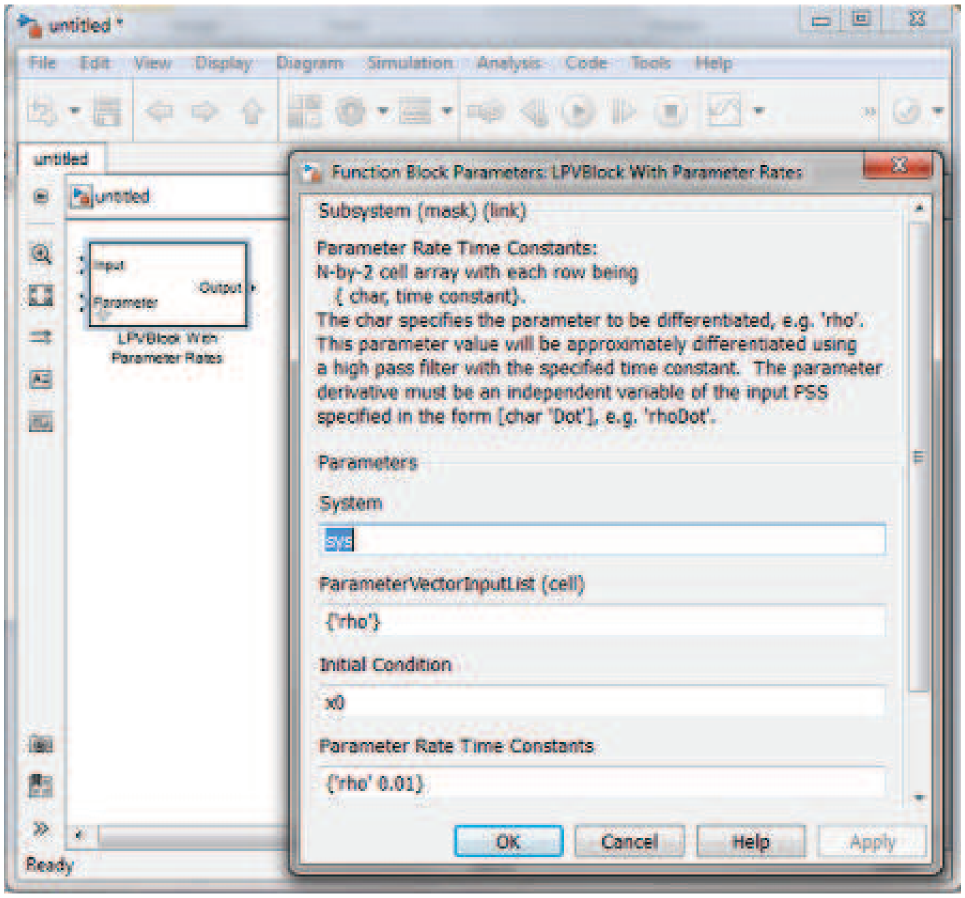
Initial Condition

0

Figure 1: Simulink LPV block and block mask.

The block in Figure 1 has inputs for the system input $u(t)$ and the parameter vector $\rho(t)$, and an output for $y(t)$. The block mask contains entries for the user to specify the system variable name, the order of the input parameter vectors ρ, and the state initial condition $x(0)$. The block is implemented as a Simulink S-function under the block mask. The block currently performs a multidimensional linear interpolation to evaluate the state-space matrices at the specified parameter vector. An efficient implementation of this linear interpolation has been coded to reduce computation and speed up the simulation time.

LPVTools also includes a block for systems that depend explicitly on both the time-varying parameter and its derivative, as seen in Equation 1. This block is shown in Figure 2.


untitled *
File Edit View Display Diagram Simulation Analysis Code Tools Help
untitled
Input
Parameter
Output
LPVBlock With Parameter Rates
Ready
OK
Cancel
Help
Apply

Function Block Parameters: LPVBlock With Parameter Rates

Subsystem (mask) (link)

Parameter Rate Time Constants:
N-by-2 cell array with each row being
{ char, time constant}.
The char specifies the parameter to be differentiated, e.g. 'rho'.
This parameter value will be approximately differentiated using
a high pass filter with the specified time constant. The parameter
derivative must be an independent variable of the input PSS
specified in the form [char 'Dot'], e.g. 'rhoDot'.

Parameters

System

sys

ParameterVectorInputList (cell)

{'rho'}

Initial Condition

x0

Parameter Rate Time Constants

{'rho' 0.01}

Figure 2: Rate-dependent Simulink LPV block and block mask.

Published with MATLAB® R2014b