LMI control toolbox user's guide

Book · May 1995

Source: IEEE Xplore

CITATIONS
1,736

READS
18,290

4 authors, including:

Pascal M Gahinet
The MathWorks, Inc
69 PUBLICATIONS **18,789** CITATIONS

SEE PROFILE

Arkadi Nemirovski
Georgia Institute of Technology
223 PUBLICATIONS **30,544** CITATIONS

SEE PROFILE

LMI Control Toolbox

For Use with MATLAB®

Pascal Gahinet
Arkadi Nemirovski
Alan J. Laub
Mahmoud Chilali

- Computation
- Visualization
- Programming

User's Guide
Version 1

How to Contact The MathWorks:

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup

`support@mathworks.com`	Technical support
`suggest@mathworks.com`	Product enhancement suggestions
`bugs@mathworks.com`	Bug reports
`doc@mathworks.com`	Documentation error reports
`service@mathworks.com`	Order status, license renewals, passcodes
`info@mathworks.com`	Sales, pricing, and general information

508-647-7000 Phone

508-647-7001 Fax

The MathWorks, Inc. Mail
3 Apple Hill Drive
Natick, MA 01760-2098

For contact information about worldwide offices, see the MathWorks Web site.

LMI Control Toolbox User's Guide

Printing History: May 1995 First printing New for Version 1

Contents

Preface

About the Authors viii

Acknowledgments ix

1 Introduction

Linear Matrix Inequalities 1-2

Toolbox Features 1-3

LMIs and LMI Problems 1-4
The Three Generic LMI Problems 1-5

Further Mathematical Background 1-9

References 1-10

2 Uncertain Dynamical Systems

Linear Time-Invariant Systems 2-3
SYSTEM Matrix 2-3

Time and Frequency Response Plots 2-6

Interconnections of Linear Systems 2-9

Model Uncertainty **2-12**

Uncertain State-Space Models **2-14**
Polytopic Models **2-14**
Affine Parameter-Dependent Models **2-15**
Quantification of Parameter Uncertainty **2-17**
Simulation of Parameter-Dependent Systems **2-19**
From Affine to Polytopic Models **2-20**
Example **2-21**

Linear-Fractional Models of Uncertainty **2-23**
How to Derive Such Models **2-23**
Specification of the Uncertainty **2-26**
From Affine to Linear-Fractional Models **2-32**

References **2-35**

3 Robustness Analysis

Quadratic Lyapunov Functions **3-3**
LMI Formulation **3-4**
Quadratic Stability **3-6**
Maximizing the Quadratic Stability Region **3-8**
Decay Rate **3-9**
Quadratic $H\infty$ Performance **3-10**

Parameter-Dependent Lyapunov Functions **3-12**
Stability Analysis **3-14**

μ Analysis **3-17**
Structured Singular Value **3-17**
Robust Stability Analysis **3-19**
Robust Performance **3-21**

The Popov Criterion **3-24**
Real Parameter Uncertainty **3-25**

Example **3-28**

References **3-32**

4 State-Feedback Synthesis

Multi-Objective State-Feedback **4-3**

Pole Placement in LMI Regions **4-5**

LMI Formulation **4-7**
Extension to the Multi-Model Case **4-9**

The Function msfsyn **4-11**

Design Example **4-13**

References **4-18**

5 Synthesis of H∞ Controllers

H∞ **Control** **5-3**
Riccati- and LMI-Based Approaches **5-7**

H∞ **Synthesis** **5-10**
Validation of the Closed-Loop System **5-13**

Multi-Objective *H*∞ **Synthesis** **5-15**
LMI Formulation **5-16**
The Function hinfmix **5-20**
Loop-Shaping Design with hinfmix **5-20**

References **5-22**

6 Loop Shaping

The Loop-Shaping Methodology **6-2**

The Loop-Shaping Methodology **6-3**

Design Example **6-5**

Specification of the Shaping Filters **6-10**
Nonproper Filters and sderiv **6-12**

Specification of the Control Structure **6-14**

Controller Synthesis and Validation **6-16**

Practical Considerations **6-18**

Loop Shaping with Regional Pole Placement **6-19**

References **6-24**

7 Robust Gain-Scheduled Controllers

Gain-Scheduled Control **7-3**

Synthesis of Gain-Scheduled H_∞ Controllers **7-7**

Simulation of Gain-Scheduled Control Systems **7-9**

Design Example **7-10**

References **7-15**

8 The LMI Lab

Background and Terminology **8-3**

Overview of the LMI Lab **8-6**

Specifying a System of LMIs **8-8**
A Simple Example **8-9**
setlmis and getlmis **8-11**
lmivar **8-11**
lmiterm **8-13**
The LMI Editor lmiedit **8-16**
How It All Works **8-18**

Retrieving Information **8-21**
lmiinfo **8-21**
lminbr and matnbr **8-21**

LMI Solvers **8-22**

From Decision to Matrix Variables and Vice Versa **8-28**

Validating Results **8-29**

Modifying a System of LMIs **8-30**
dellmi **8-30**
dellmi **8-30**
setmvar **8-31**

Advanced Topics **8-33**
Structured Matrix Variables **8-33**
Complex-Valued LMIs **8-35**
Specifying $c^T x$ Objectives for mincx **8-38**
Feasibility Radius **8-39**

Well-Posedness Issues **8-40**
Semi-Definite B(x) in gevp Problems **8-41**
Efficiency and Complexity Issues **8-41**
Solving M + PTXQ + QTXTP < 0 . . . . . **8-42**

References **8-44**

9 Command Reference

List of Functions **9-3**

H∞ Control and Loop Shaping **9-6**

LMI Lab: Specifying and Solving LMIs **9-7**

LMI Lab: Additional Facilities **9-8**

Preface

About the Authors

Dr. **Pascal Gahinet** is a reserach fellow at INRIA Rocquencourt, France. His research interests include robust control theory, linear matrix inequalities, numerical linear algebra, and numerical software for control.

Prof. **Arkadi Nemirovski** is with the Faculty of Industrial Engineering and Management at Technion, Haifa, Israel. His research interests include convex optimization, complexity theory, and non-parametric statistics.

Prof. **Alan J. Laub** is with the Electrical and Computer Engineering Department of the University of California at Santa Barbara, USA. His research interests are in numerical analysis, mathematical software, scientific computation, computer-aided control system design, and linear and large-scale control and filtering theory.

Mahmoud Chilali is completing his Ph.D. at INRIA Rocquencourt, France. His thesis is on the theory and applications of linear matrix inequalities in control.

Acknowledgments

The authors wish to express their gratitude to all colleagues who directly or indirectly contributed to the making of the LMI Control Toolbox. Special thanks to Pierre Apkarian, Gregory Becker, Hiroyuki Kajiwara, and Anca Ignat for their help and contribution. Many thanks also to those who tested and helped refine the software, including Bobby Bodenheimer, Markus Brandstetter, Eric Feron, K.C. Goh, Anders Helmersson, Ted Iwasaki, Jianbo Lu, Roy Lurie, Jason Ly, John Morris, Ravi Prasanth, Michael Safonov, Carsten Scherer, Andy Sparks, Mario Rotea, Matthew Lamont Tyler, Jim Tung, and John Wen. Apologies, finally, to those we may have omitted.

The work of Pascal Gahinet was supported in part by INRIA.

1

Introduction

Linear Matrix Inequalities 1-2

Toolbox Features 1-3

LMIs and LMI Problems 1-4
The Three Generic LMI Problems 1-5

Further Mathematical Background 1-9

References 1-10

Linear Matrix Inequalities

Linear Matrix Inequalities (LMIs) and LMI techniques have emerged as powerful design tools in areas ranging from control engineering to system identification and structural design. Three factors make LMI techniques appealing:

- A variety of design specifications and constraints can be expressed as LMIs.
- Once formulated in terms of LMIs, a problem can be solved *exactly* by efficient convex optimization algorithms (the "LMI solvers").
- While most problems with multiple constraints or objectives lack analytical solutions in terms of matrix equations, they often remain tractable in the LMI framework. This makes LMI-based design a valuable alternative to classical "analytical" methods.

See [9] for a good introduction to LMI concepts. The LMI Control Toolbox is designed as an easy and progressive gateway to the new and fast-growing field of LMIs:

- For users mainly interested in applying LMI techniques to control design, the LMI Control Toolbox features a variety of high-level tools for the analysis and design of multivariable feedback loops (see Chapters 2 through 7).
- For users who occasionally need to solve LMI problems, the "LMI Editor" and the tutorial introduction to LMI concepts and LMI solvers provide for quick and easy problem solving.
- For more experienced LMI users, the "LMI Lab" (Chapter 8) offers a rich, flexible, and fully programmable environment to develop customized LMI-based tools.

The LMI Control Toolbox implements state-of-the-art interior-point LMI solvers. While these solvers are significantly faster than classical convex optimization algorithms, it should be kept in mind that the complexity of LMI computations remains higher than that of solving, say, a Riccati equation. For instance, problems with a thousand design variables typically take over an hour on today's workstations. However, research on LMI optimization is still very active and substantial speed-ups can be expected in the future. Thanks to its efficient "structured" representation of LMIs, the LMI Control Toolbox is geared to making the most out of such improvements.

Toolbox Features

The LMI Control Toolbox serves two purposes:

- Provide state-of-the-art tools for the LMI-based analysis and design of robust control systems
- Offer a flexible and user-friendly environment to specify and solve general LMI problems (the LMI Lab)

The control design tools can be used without *a priori* knowledge about LMIs or LMI solvers. These are dedicated tools covering the following applications of LMI techniques:

- Specification and manipulation of uncertain dynamical systems (linear-time invariant, polytopic, parameter-dependent, etc.)
- Robustness analysis. Various measures of robust stability and performance are implemented, including quadratic stability, techniques based on parameter-dependent Lyapunov functions, analysis, and Popov analysis.
- Multi-model/multi-objective state-feedback design
- Synthesis of output-feedback H_∞ controllers via Riccati- and LMI-based techniques, including mixed H_2 / H_∞ synthesis with regional pole placement constraints
- Loop-shaping design
- Synthesis of robust gain-scheduled controllers for time-varying parameter-dependent systems

For users interested in developing their own applications, the LMI Lab provides a general-purpose and fully programmable environment to specify and solve virtually any LMI problem. Note that the scope of this facility is by no means restricted to control-oriented applications.

LMIs and LMI Problems

A linear matrix inequality (LMI) is any constraint of the form

$$A(x) := A_0 + x_1 A_1 + \ldots + x_N A_N < 0 \qquad \textbf{(1-1)}$$

where

- $x = (x_1, \ldots, x_N)$ is a vector of unknown scalars (the *decision* or *optimization* variables)
- $A_0, \ldots, A_N$ are given *symmetric* matrices
- < 0 stands for "negative definite," i.e., the largest eigenvalue of $A(x)$ is negative

Note that the constraints $A(x) > 0$ and $A(x) < B(x)$ are special cases of (1-1) since they can be rewritten as $-A(x) < 0$ and $A(x) - B(x) < 0$, respectively.

The LMI (1-1) is a convex constraint on x since $A(y) < 0$ and $A(z) < 0$ imply that $A\left(\frac{y+z}{2}\right) < 0$. As a result,

- Its solution set, called the *feasible set*, is a convex subset of $\mathbf{R}^N$
- Finding a solution x to (1-1), if any, is a convex optimization problem.

Convexity has an important consequence: even though (1-1) has no analytical solution in general, it can be solved numerically with guarantees of finding a solution when one exists. Note that a system of LMI constraints can be regarded as a single LMI since

$$\begin{cases} A_1(x) < 0 \\ \quad \vdots \quad \text{is equivalent to } A(x) := \operatorname{diag}(A_1(x), \ldots, A_K(x)) < 0 \\ A_K(x) < 0 \end{cases}$$

where diag $(A_1(x), \ldots, A_K(x))$ denotes the block-diagonal matrix with $A_1(x), \ldots, A_K(x)$ on its diagonal. Hence multiple LMI constraints can be imposed on the vector of decision variables x without destroying convexity.

In most control applications, LMIs do not naturally arise in the canonical form (1-1), but rather in the form

$$L(X_1, \ldots, X_n) < R(X_1, \ldots, X_n)$$

where $L(\cdot)$ and $R(\cdot)$ are affine functions of some structured *matrix* variables X_1, $\ldots$, X_n. A simple example is the Lyapunov inequality

$$A^T X + XA < 0 \tag{1-2}$$

where the unknown X is a symmetric matrix. Defining $x_1, \ldots, x_N$ as the independent scalar entries of X, this LMI could be rewritten in the form (1-1). Yet it is more convenient and efficient to describe it in its natural form (1-2), which is the approach taken in the LMI Lab.

The Three Generic LMI Problems

Finding a solution x to the LMI system

$$A(x) < 0 \tag{1-3}$$

is called the feasibility problem. Minimizing a convex objective under LMI constraints is also a convex problem. In particular, the *linear objective minimization problem*

$$\text{Minimize } c^T x \text{ subject to } A(x) < 0 \tag{1-4}$$

plays an important role in LMI-based design. Finally, the *generalized eigenvalue minimization problem*

$$\text{Minimize } \lambda \text{ subject to } \begin{cases} A(x) < \lambda B(x) \\ B(x) > 0 \\ C(x) < 0 \end{cases} \tag{1-5}$$

is quasi-convex and can be solved by similar techniques. It owes its name to the fact that is related to the largest generalized eigenvalue of the pencil $(A(x),B(x))$.

Many control problems and design specifications have LMI formulations [9]. This is especially true for Lyapunov-based analysis and design, but also for

optimal LQG control, H_∞ control, covariance control, etc. Further applications of LMIs arise in estimation, identification, optimal design, structural design [6, 7], matrix scaling problems, and so on. The main strength of LMI formulations is the ability to combine various design constraints or objectives in a numerically tractable manner.

A nonexhaustive list of problems addressed by LMI techniques includes the following:

- Robust stability of systems with LTI uncertainty (μ-analysis) [24, 21, 27]
- Robust stability in the face of sector-bounded nonlinearities (Popov criterion) [22, 28, 13, 16]
- Quadratic stability of differential inclusions [15, 8]
- Lyapunov stability of parameter-dependent systems [12]
- Input/state/output properties of LTI systems (invariant ellipsoids, decay rate, etc.) [9]
- Multi-model/multi-objective state feedback design [4, 17, 3, 9, 10]
- Robust pole placement
- Optimal LQG control [9]
- Robust H_∞ control [11, 14]
- Multi-objective H_∞ synthesis [17, 23, 10, 18]
- Design of robust gain-scheduled controllers [5, 2]
- Control of stochastic systems [9]
- Weighted interpolation problems [9]

To hint at the principles underlying LMI design, let's review the LMI formulations of a few typical design objectives.

Stability. the stability of the dynamical system

$$\dot{x} = Ax$$

is equivalent to the feasibility of

$$\text{Find } P = P^T \text{ such that } A^T P + P A < 0,\ P > I.$$

This can be generalized to linear differential inclusions (LDI)

$$\dot{x} = A(t)x$$

where $A(t)$ varies in the convex envelope of a set of LTI models:

$$A(t) \in \mathrm{Co}\{A_1,\ldots,A_n\} = \left\{ \sum_{i=1}^{n} a_i A_i : a_i \geq 0, \sum_{i=1}^{N} a_i = 1 \right\}$$

A sufficient condition for the asymptotic stability of this LDI is the feasibility of

$$\text{Find } P = P^T \text{ such that } A_i^T P + P A_i < 0, \;\; P > I.$$

RMS gain . the random-mean-squares (RMS) gain of a stable LTI system

$$\begin{cases} \dot{x} = Ax + Bu \\ y = Cx + Du \end{cases}$$

is the largest input/output gain over all bounded inputs $u(t)$. This gain is the global minimum of the following linear objective minimization problem [1, 26, 25].

Minimize γ over $X = X^T$ and γ such that

$$\begin{pmatrix} A^T X + XA & XB & C^T \\ B^T X & -\gamma I & D^T \\ C & D & -\gamma I \end{pmatrix} < 0$$

$$X > 0$$

LQG performance . for a stable LTI system

$$G \begin{cases} \dot{x} = Ax + Bw \\ y = Cx \end{cases}$$

where w is a white noise disturbance with unit covariance, the LQG or H_2 performance $\|G\|_2$ is defined by

$$\|G\|_2^2 := \lim_{T\to\infty} E\left\{\frac{1}{T}\int_0^T y^T(t)y(t)dt\right\}$$
$$= \frac{1}{2\pi}\int_{-\infty}^{\infty} G^H(j\omega)G(j\omega)d\omega$$

It can be shown that

$$\|G\|_2^2 = \inf\{\text{Trace}\,(CPC^T) : AP + PA^T + BB^T < 0\}$$

Hence $\|G\|_2^2$ is the global minimum of the LMI problem

Minimize Trace (Q) over the symmetric matrices P,Q such that

$$AP + PA^T + BB^T < 0$$
$$\begin{pmatrix} Q & CP \\ PC^T & P \end{pmatrix} > 0$$

Again this is a linear objective minimization problem since the objective Trace (Q) is linear in the decision variables (free entries of P,Q).

Further Mathematical Background

Efficient interior-point algorithms are now available to solve the three generic LMI problems (1-3)–(1-5) defined in "The Three Generic LMI Problems" on page 1-5. These algorithms have a polynomial-time complexity. That is, the number $N(\varepsilon)$ of flops needed to compute an ε-accurate solution is bounded by

$$N(\varepsilon) \leq M N^3 \log(V/\varepsilon)$$

where M is the total row size of the LMI system, N is the total number of scalar decision variables, and V is a data-dependent scaling factor. The LMI Control Toolbox implements the Projective Algorithm of Nesterov and Nemirovski [20, 19]. In addition to its polynomial-time complexity, this algorithm does not require an initial feasible point for the linear objective minimization problem (1-4) or the generalized eigenvalue minimization problem (1-5).

Some LMI problems are formulated in terms of inequalities rather than strict inequalities. For instance, a variant of (1-4) is

$$\text{Minimize } c^T x \text{ subject to } A(x) \leq 0.$$

While this distinction is immaterial in general, it matters when $A(x)$ can be made negative semi-definite but not negative definite. A simple example is

$$\text{Minimize } c^T x \text{ subject to } \begin{pmatrix} x & x \\ x & x \end{pmatrix} \geq 0. \qquad \textbf{(1-6)}$$

Such problems cannot be handled directly by interior-point methods which require strict feasibility of the LMI constraints. A well-posed reformulation of (1-6) would be

$$\text{Minimize } c^T x \text{ subject to } x \leq 0.$$

Keeping this subtlety in mind, we always use strict inequalities in this manual.

References

[1] Anderson, B.D.O, and S. Vongpanitlerd, Network Analysis, Prentice-Hall, Englewood Cliffs, 1973.

[2] Apkarian, P., P. Gahinet, and G. Becker, "Self-Scheduled H_∞ Control of Linear Parameter-Varying Systems," *Proc. Amer. Contr. Conf.*, 1994, pp. 856-860.

[3] Bambang, R., E. Shimemura, and K. Uchida, "Mixed H_2 /H_∞ Control with Pole Placement," State-Feedback Case," *Proc. Amer. Contr. Conf.*, 1993, pp. 2777-2779.

[4] Barmish, B.R., "Stabilization of Uncertain Systems via Linear Control," *IEEE Trans. Aut. Contr.*, AC–28 (1983), pp. 848-850.

[5] Becker, G., Packard, P., "Robust Performance of Linear-Parametrically Varying Systems Using Parametrically-Dependent Linear Feedback," *Systems and Control Letters*, 23 (1994), pp. 205-215.

[6] Bendsoe, M.P., A. Ben-Tal, and J. Zowe, "Optimization Methods for Truss Geometry and Topology Design," to appear in *Structural Optimization*.

[7] Ben-Tal, A., and A. Nemirovski, "Potential Reduction Polynomial-Time Method for Truss Topology Design," to appear in *SIAM J. Contr. Opt*.

[8] Boyd, S., and Q. Yang, "Structured and Simultaneous Lyapunov Functions for System Stability Problems," *Int. J. Contr.*, 49 (1989), pp. 2215-2240.

[9] Boyd, S., L. El Ghaoui, E. Feron, V. Balakrishnan, *Linear Matrix Inequalities in Systems and Control Theory*, SIAM books, Philadelphia, 1994.

[10] Chilali, M., and P. Gahinet, "H_∞ Design with Pole Placement Constraints: an LMI Approach," to appear in *IEEE Trans. Aut. Contr.* Also in *Proc. Conf. Dec. Contr.*, 1994, pp. 553-558.

[11] Gahinet, P., and P. Apkarian, "A Linear Matrix Inequality Approach to H_∞ Control," *Int. J. Robust and Nonlinear Contr.*, 4 (1994), pp. 421-448.

[12] Gahinet, P., P. Apkarian, and M. Chilali, "Affine Parameter-Dependent Lyapunov Functions for Real Parametric Uncertainty," *Proc. Conf. Dec. Contr.*, 1994, pp. 2026-2031.

[13] Haddad, W.M. and D.S. Berstein,"Parameter-Dependent Lyapunov Functions, Constant Real Parameter Uncertainty, and the Popov Criterion in Robust Analysis and Synthesis: Part 1 and 2," *Proc. Conf. Dec. Contr.*, 1991, pp. 2274-2279 and 2632-2633.

[14] Iwasaki, T., and R.E. Skelton, "All Controllers for the General H_∞ Control Problem: LMI Existence Conditions and State-Space Formulas," *Automatica*, 30 (1994), pp. 1307-1317.

[15] Horisberger, H.P., and P.R. Belanger, "Regulators for Linear Time-Varying Plants with Uncertain Parameters," *IEEE Trans. Aut. Contr.*, AC–21 (1976), pp. 705-708.

[16] How, J.P., and S.R. Hall, "Connection between the Popov Stability Criterion and Bounds for Real Parameter Uncertainty," *Proc. Amer. Contr. Conf.*, 1993, pp. 1084-1089.

[17] Khargonekar, P.P., and M.A. Rotea,"Mixed H_2 /H_∞ Control: a Convex Optimization Approach," *IEEE Trans. Aut. Contr.*, 39 (1991), pp. 824-837.

[18] Masubuchi, I., A. Ohara, and N. Suda, "LMI-Based Controller Synthesis: *A* Unified Formulation and Solution," submitted to *Int. J. Robust and Nonlinear Contr.*, 1994.

[19] Nemirovski, A., and P. Gahinet, "The Projective Method for Solving Linear Matrix Inequalities," *Proc. Amer. Contr. Conf.*, 1994, pp. 840-844.

[20] Nesterov, Yu, and A. Nemirovski, *Interior Point Polynomial Methods in Convex Programming: Theory and Applications*, SIAM Books, Philadelphia, 1994.

[21] Packard, A., and J.C. Doyle, "The Complex Structured Singular Value," *Automatica*, 29 (1994), pp. 71-109.

[22] Popov, V.M., "Absolute Stability of Nonlinear Systems of Automatic Control," *Automation and Remote Contro*l, 22 (1962), pp. 857-875.

[23] Scherer, C., "Mixed H_2 H_∞ Control," to appear in *Trends in Control: A European Perspective,* volume of the special contributions to the ECC 1995.

[24] Stein, G. and J.C. Doyle, "Beyond Singular Values and Loop Shapes," *J. Guidance*, 14 (1991), pp. 5-16.

[25] Vidyasagar, M., *Nonlinear System Analysis*, Prentice-Hall, Englewood Cliffs, 1992.

[26] Willems, J.C., “Least-Squares Stationary Optimal Control and the Algebraic Riccati Equation,” *IEEE Trans. Aut. Contr*., AC–16 (1971), pp. 621-634.

[27] Young, P. M., M. P. Newlin, and J. C. Doyle, “Let's Get Real,” in *Robust Control Theory*, Springer Verlag, 1994, pp. 143-174.

[28] Zames, G., “On the Input-Output Stability of Time-Varying Nonlinear Feedback Systems, Part I and II,” *IEEE Trans. Aut. Contr*., AC–11 (1966), pp. 228-238 and 465-476.

2

Uncertain Dynamical Systems

Linear Time-Invariant Systems 2-3
SYSTEM Matrix 2-3

Time and Frequency Response Plots 2-6

Interconnections of Linear Systems 2-9

Model Uncertainty 2-12

Uncertain State-Space Models 2-14
Polytopic Models 2-14
Affine Parameter-Dependent Models 2-15
Quantification of Parameter Uncertainty 2-17
Simulation of Parameter-Dependent Systems 2-19
From Affine to Polytopic Models 2-20
Example 2-21

Linear-Fractional Models of Uncertainty 2-23
How to Derive Such Models 2-23
Specification of the Uncertainty 2-26
From Affine to Linear-Fractional Models 2-32

References 2-35

The LMI Control Toolbox offers a variety of tools to facilitate the description and manipulation of uncertain dynamical systems. These include functions to:

- Manipulate the state-space realization of linear time-invariant (LTI) systems as a single `SYSTEM` matrix
- Form interconnections of linear systems
- Specify linear systems with uncertain state-space matrices (polytopic differential inclusions, systems with uncertain or time-varying physical parameters, etc.)
- Describe linear-fractional models of uncertainty

The next sections provide a tutorial introduction to uncertain dynamical systems and an overview of these facilities.

Linear Time-Invariant Systems

The LMI Control Toolbox provides streamlined tools to manipulate state-space representations of linear time-invariant (LTI) systems. These tools handle general LTI models of the form

$$E\frac{dx}{dt} = Ax(t) + Bu(t)$$
$$y(t) = Cx(t) + Du(t)$$

where A, B, C, D, E are real matrices and E is *invertible*, as well as their discrete-time counterpart

$$Ex_{k+1} = Ax_k + Bu_k$$
$$y_k = Cx_k + Du_k$$

Recall that the vectors $x(t)$, $u(t)$, $y(t)$ denote the state, input, and output trajectories. Similarly, x_k, u_k, y_k denote the values of the state, input, and output vectors at the sample time k.

The "descriptor" formulation ($E \neq I$) proves useful when specifying parameter-dependent systems (see "Affine Parameter-Dependent Models" on page 2-15) and also avoids inverting E when this inversion is poorly conditioned. Moreover, many dynamical systems are naturally written in descriptor form. For instance, the second-order system

$$m\ddot{x} + f\dot{x} + kx = u, \qquad y = x.$$

admits the state-space representation

$$\begin{pmatrix} 1 & 0 \\ 0 & m \end{pmatrix}\frac{dX}{dt} = \begin{pmatrix} 0 & 1 \\ -k & -f \end{pmatrix}X + \begin{pmatrix} 0 \\ 1 \end{pmatrix}u, \qquad y = (1, 0)X$$

where $X(t) := \begin{pmatrix} x(t) \\ \dot{x}(t) \end{pmatrix}$.

SYSTEM Matrix

For convenience, the state-space realization of LTI systems is stored as a single MATLAB matrix called a `SYSTEM` matrix. Specifically, a continuous- or

discrete-time LTI system with state-space matrices *A, B, C, D, E* is represented by the structured matrix

$$\left[\begin{array}{cc|c} A+j(E-I) & B & \begin{matrix} n \\ 0 \\ \vdots \\ 0 \end{matrix} \\ C & D & \\ \hline 0 & & -\mathrm{Inf} \end{array}\right]$$

where $j = \sqrt{-1}$. The upper right entry n corresponds to the number of states (i.e., $A \in \mathbf{R}^{n\times n}$) while the entry `Inf` is used to differentiate `SYSTEM` matrices from regular matrices. This data structure is similar to that used in the μ Analysis and Synthesis Toolbox.

The functions `ltisys` and `ltiss` create `SYSTEM` matrices and extract state-space data from them. For instance,

```
sys = ltisys( 1,1,1,0)
```

specifies the LTI system

$$\dot{x} = -x + u, \qquad y = x$$

To retrieve the values of *A, B, C, D* from the `SYSTEM` matrix `sys`, type

```
[a,b,c,d] = ltiss(sys)
```

For single-input/single-output (SISO) systems, the function `ltitf` returns the numerator/denominator representation of the transfer function

$$G(s) = D + C(sE - A)^{-1}B = \frac{n(s)}{d(s)}$$

Conversely, the command

```
sys = ltisys('tf',n,d)
```

returns a state-space realization of the SISO transfer function $n(s)/d(s)$ in `SYSTEM` format. Here `n` and `d` are the vector representation of $n(s)$ and $d(s)$ (type `help poly` for details).

The number of states, inputs, and outputs of a linear system are retrieved from the `SYSTEM` matrix with `sinfo`:

```
sinfo(sys)
System with 1 state(s), 1 input(s), and 1 output(s)
```

Similarly, the poles of the system are given by `spol`:

```
spol(sys)

ans =
    1
```

The function `ssub` selects particular inputs and outputs of a system and returns the corresponding subsystem. For instance, if G has two inputs and three outputs, the subsystem mapping the first input to the second and third outputs is given by

```
ssub(g,1,2:3)
```

The function `sinv` computes the inverse $H(s) = G(s)^{-1}$ of a system $G(s)$ with square invertible D matrix:

```
h = sinv(g)
```

Finally, the state-space realization of an LTI system can be "balanced" with `sbalanc`. This function seeks a diagonal scaling similarity that reduces the norms of A, B, C.

Time and Frequency Response Plots

Time and frequency responses of linear systems are plotted directly from the SYSTEM matrix with the function `splot`. For the sake of illustration, consider the second-order system

$$m\ddot{x} + f\dot{x} + kx = u, \qquad y = x$$

with $m = 2$, $f = 0.01$, and $k = 0.5$. This system is specified in descriptor form by

```
sys = ltisys([0 1, 0.5  0.01],[0,1],[1 0],0,[1 0,0 2])
```

the last input argument being the E matrix. To plot its Bode diagram, type

```
splot(sys,'bo')
```

The second argument is the string consisting of the first two letters of "bode." This command produces the plot of Figure 2-1. A third argument can be used to adjust the frequency range. For instance,

```
splot(sys,'bo',logspace( 1,1,50))
```

displays the Bode plot of Figure 2-2 instead.

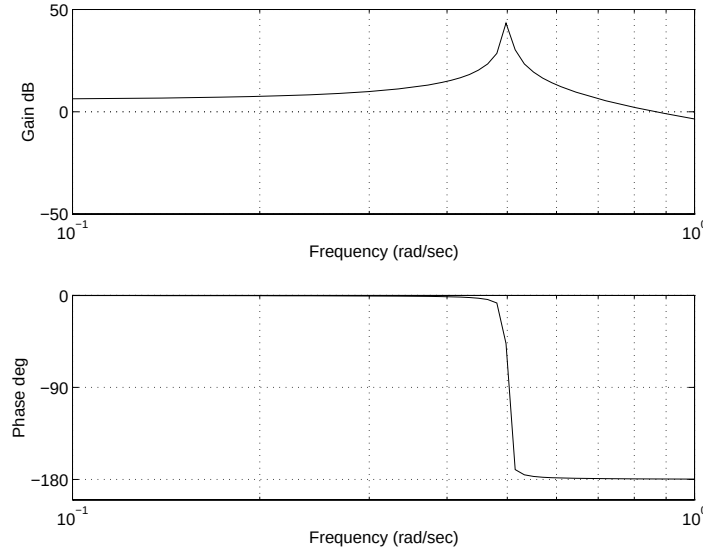


Figure 2-1: splot (sys,'bo')

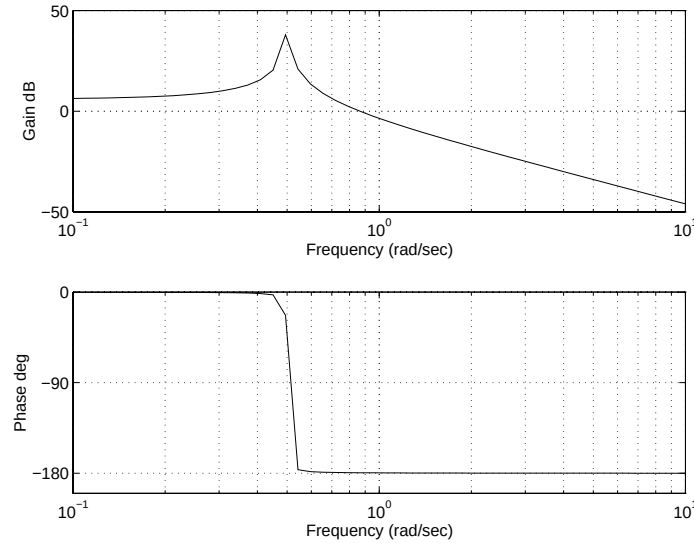


Figure 2-2: splot (sys,'bo',logspace (−1,1,50))

To draw the singular value plot of this second-order system, type

```
splot(sys,'sv').
```

For a system with transfer function $G(s) = D + C(sE - A)^{-1}B$, the singular value plot consists of the curves

$$(\omega, \sigma_i(G(j\omega)))$$

where σ_i denotes the i-th singular value of a matrix M in the order

$$\sigma_1(M) \geq \sigma_2(M) \geq \ldots \geq \sigma_p(M).$$

Similarly, the step and impulse responses of this system are plotted by `splot(sys,'st')` and `splot(sys,'im')`, respectively. See the "Command Reference" chapter for a complete list of available diagrams.

The function `splot` is also applicable to discrete-time systems. For such systems, the sampling period T must be specified to plot frequency responses. For instance, the Bode plot of the system

$$\begin{cases} x_{k+1} = 0.1x_k + 0.2u_k \\ y_k = -x_k \end{cases}$$

with sampling period $t = 0.01$ s is drawn by

```
splot(ltisys(0.1,0.2, 1),0.01,'bo')
```

Interconnections of Linear Systems

Tools are provided to form series, parallel, and simple feedback interconnections of LTI systems. More complex interconnections can be built either incrementally with these basic facilities or directly with `sconnect` (see"Polytopic Models" on page 2-14 for details). The interconnection functions work on the `SYSTEM` matrix representation of dynamical systems and their names start with an `s`. Note that most of these functions are also applicable to polytopic or parameter- dependent models (see "Polytopic Models" on page 2-14).

Series and parallel interconnections are performed by `sadd` and `smult`. In terms of transfer functions,

```
sadd(g1,g2)
```

returns the system with transfer function $G_1(s) + G_2(s)$ while

```
smult(g1,g2)
```

returns the system with transfer function $G_2(s)G_1(s)$. Both functions take up to ten input arguments, at most one of which can be a parameter-dependent system. Similarly, the command

```
sdiag(g1,g2)
```

appends (concatenates) the systems G_1 and G_2 and returns the system with transfer function

$$G(s) = \begin{pmatrix} G_1(s) & 0 \\ 0 & G_2(s) \end{pmatrix}$$

This corresponds to combining the input/output relations

$$y_1 = G_1(s)u_1, \qquad y_2 = G_2(s)u_2$$

into the single relation

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix} = G(s) \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$$

Finally, the functions `sloop` and `slft` form basic feedback interconnections. The function `sloop` computes the closed-loop mapping between r and y in the loop of Figure 2-3. The result is a state-space realization of the transfer function

$$(I - \varepsilon G_1 G_2)^{-1} G_1$$

where $\varepsilon = \pm 1$.

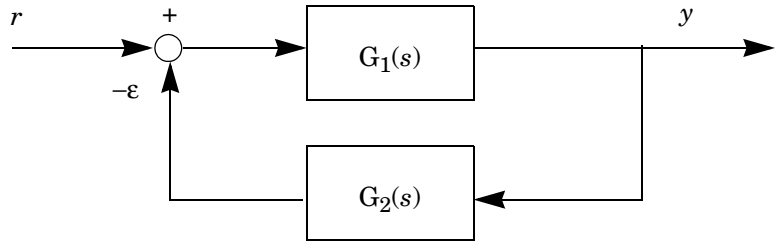
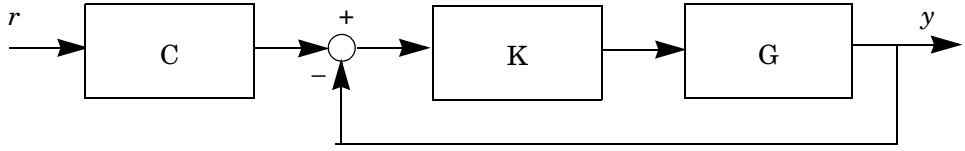


Figure 2-3: sloop

This function is useful to specify simple feedback loops. For instance, the closed-loop system `clsys` corresponding to the two-degree-of-freedom tracking loop



is derived by setting $G_1(s) = G(s)K(s)$ and $G_2(s) = 1$:

```
clsys = smult(c, sloop(smult(k,g),1))
```

The function `slft` forms the more general feedback interconnection of Figure 2-4 and returns the closed-loop mapping from $\begin{pmatrix}\omega_1\\ \omega_2\end{pmatrix}$ to $\begin{pmatrix}z_1\\ z_2\end{pmatrix}$. To form this interconnection when $u \in \mathbf{R}^2$ and $y \in \mathbf{R}^3$, the command is

```
slft(P1,P2,2,3)
```

The last two arguments dimension u and y. This function is useful to compute linear-fractional interconnections such as those arising in H_∞ theory and its extensions (see "How to Derive Such Models" on page 2-23).

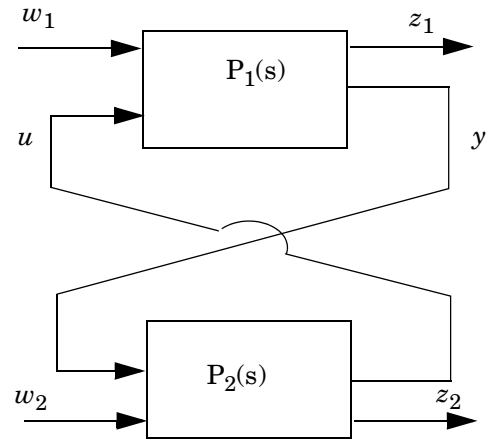


Figure 2-4: slft

Model Uncertainty

The notion of uncertain dynamical system is central to robust control theory. For control design purposes, the possibly complex behavior of dynamical systems must be approximated by models of relatively low complexity. The gap between such models and the true physical system is called the *model uncertainty*. Another cause of uncertainty is the imperfect knowledge of some components of the system, or the alteration of their behavior due to changes in operating conditions, aging, etc. Finally, uncertainty also stems from physical parameters whose value is only approximately known or varies in time. Note that model uncertainty should be distinguished from exogenous actions such as disturbances or measurement noise.

The LMI Control Toolbox focuses on the class of dynamical systems that can be approximated by linear models up to some possibly nonlinear and/or time-varying model uncertainty. When deriving the nominal model and estimating the uncertainty, two fundamental principles must be remembered:

- Uncertainty should be small where high performance is desired (tradeoff between performance and robustness). In other words, the linear model should be sufficiently accurate in the control bandwidth.
- The more information you have about the uncertainty (phase, structure, time invariance, etc.), the higher the achievable performance will be.

There are two major classes of uncertainty:

- Dynamical uncertainty, which consists of dynamical components neglected in the linear model as well as of variations in the dynamical behavior during operation. For instance, high-frequency flexible modes, nonlinearities for large inputs, slow time variations, etc.
- Parameter uncertainty, which stems from imperfect knowledge of the physical parameter values, or from variations of these parameters during operation. Examples of physical parameters include stiffness and damping coefficients in mechanical systems, aerodynamical coefficients in flying devices, capacitors and inductors in electric circuits, etc.

Other important characteristics of uncertainty include whether it is linear or nonlinear, and whether it is time invariant or time varying. Model uncertainty is generally a combination of dynamical and parametric uncertainty, and may arise at several different points in the control loop. For instance, there may be

dynamical uncertainty on the system actuators, and parametric uncertainty on some sensor coefficients. Two representations of model uncertainty are used in the LMI Control Toolbox:

- *Uncertain state-space models*. This representation is relevant for systems described by dynamical equations with uncertain and/or time-varying coefficients.
- *Linear-fractional representation* of uncertainty. Here the uncertain system is described as an interconnection of known LTI systems with uncertain components called "uncertainty blocks." Each uncertainty block $\Delta_i(\cdot)$ represents a family of systems of which only a few characteristics are known. For instance, the only available information about Δ_i may be that it is a time-invariant nonlinearity with gain less than 0.01.

Determinant factors in the choice of representation include the available model (state-space equations, frequency-domain model, etc.) and the analysis or synthesis tool to be used.

Uncertain State-Space Models

Physical models of a system often lead to a state-space description of its dynamical behavior. The resulting state-space equations typically involve physical parameters whose value is only approximately known, as well as approximations of complex and possibly nonlinear phenomena. In other words, the system is described by an *uncertain state-space model*

$$E\dot{x} = Ax + Bu, \qquad y = Cx + Du$$

where the state-space matrices *A, B, C, D, E* depend on uncertain and/or time-varying parameters or vary in some bounded sets of the space of matrices.

Of particular relevance to this toolbox are the classes of polytopic or parameter-dependent models discussed next. We collectively refer to such models as *P-systems*, "P–" standing for "polytopic or parameter-dependent." P-systems are specified with the function `psys` and manipulated like ordinary LTI systems except for a few specific restrictions.

Polytopic Models

We call *polytopic system* a linear time-varying system

$$E(t)\,\dot{x} = A(t)\,x + B(t)\,u$$

$$y = C(t)\,x + D(t)\,u$$

whose `SYSTEM` matrix $S(t) = \begin{bmatrix} A(t)+jE(t) & B(t) \\ C(t) & D(t) \end{bmatrix}$ varies within a fixed polytope of matrices, i.e.,

$$S(t) \in \mathrm{Co}\{S_1, \ldots, S_k\} := \left\{ \sum_{i=1}^{k} \alpha_i S_i \,:\, \alpha_i \geq 0,\ \sum_{i=1}^{k} \alpha_i = 1 \right\}$$

where $S_1, \ldots, S_k$ are given *vertex systems*:

$$S_1 = \begin{bmatrix} A_1+jE_1 & B_1 \\ C_1 & D_1 \end{bmatrix}, \ldots S_k = \begin{bmatrix} A_k+jE_k & B_k \\ C_k & D_k \end{bmatrix} \qquad \textbf{(2-1)}$$

In other words, $S(t)$ is a convex combination of the `SYSTEM` matrices $S_1, \ldots, S_k$. The nonnegative numbers $\alpha_1, \ldots, \alpha_k$ are called the polytopic coordinates of S.

Such models are also called polytopic linear differential inclusions in the literature [3] and arise in many practical situations, including:

- Multimodel representation of a system, each model being derived around particular operating conditions
- Nonlinear systems of the form

  $\dot{x} = A(x)\,x + B(x)\,u, \qquad y = C(x)\,x + D(x)\,u$

- State-space models depending affinely on time-varying parameters (see "From Affine to Polytopic Models" on page 2-20)

A simple example is the system

$$\dot{x} = (\sin x)\,x$$

whose state matrix $A = \sin x$ ranges in the polytope

$$A \in \mathrm{Co}\{-1, 1\} = [-1, 1].$$

Polytopic systems are specified by the list of their vertex systems, i.e., by the `SYSTEM` matrices $S_1, \ldots, S_k$ in (2-1). For instance, a polytopic model taking values in the convex envelope of the three LTI systems `s1, s2, s3` is declared by

```
polsys = psys([s1 s2 s3])
```

Affine Parameter-Dependent Models

The equations of physics often involve uncertain or time-varying coefficients. When the system is linear, this naturally gives rise to parameter-dependent models (PDS) of the form

$$E(p)\,\dot{x} = A(p)\,x + B(p)\,u$$

$$y = C(p)\,x + D(p)\,u$$

where $A(\cdot), \ldots, E(\cdot)$ are known functions of some parameter vector $p = (p_1, \ldots, p_n)$. Such models commonly arise from the equations of motion, aerodynamics, circuits, etc.

The LMI Control Toolbox offers various tools to analyze the stability and performance of parameter-dependent systems with an affine dependence on the parameter vector $p = (p_1, \ldots, p_n)$. That is, PDSs where

$$A(p) = A_0 + p_1A_1 + \ldots + p_nA_n,\ B(p) = B_0 + p_1B_1 + \ldots + p_nB_n,$$

and so on. Affine parameter-dependent models are well-suited for Lyapunov-based analysis and synthesis and are easily converted to linear-fractional uncertainty models for small-gain-based design (see the nonlinear spring example in "Sector-Bounded Uncertainty" on page 2-30).

With the notation

$$S(p) = \begin{pmatrix} A(p)+jE(p) & B(p) \\ C(p) & D(p) \end{pmatrix}, \qquad S_i = \begin{pmatrix} A_i+jE_i & B_i \\ C_i & D_i \end{pmatrix},$$

the affine dependence on p is written more compactly in `SYSTEM` matrix terms as

$$S(p) = S_0 + p_1 S_1 + \ldots + p_n S_n.$$

The system "coefficients" $S_0, \ldots, S_n$ fully characterize the dependence on the uncertain parameters $p_1, \ldots, p_n$. Note that $S_0, \ldots, S_n$ need not represent meaningful dynamical systems. Only their combination $S(p)$ is a relevant description of the problem.

Affine parameter-dependent systems are specified with `psys` by providing

- A description of the parameter vector p in terms of bounds on the parameter values and rates of variation
- The list of `SYSTEM` matrix coefficients $S_0, \ldots, S_n$

For instance, the system

$$S(p) = S_0 + p_1 S_1 + p_2 S_2$$

is defined by

```
s0 = ltisys(a0,b0,c0,d0,e0)
s1 = ltisys(a1,b1,c1,d1,e1)
s2 = ltisys(a2,b2,c2,d2,e2)
affsys = psys(pv, [s0 s1 s2])
```

where `pv` is the parameter description returned by `pvec` (see next subsection for details). The output `affsys` is a structured matrix storing all relevant data.

Important: By default, `ltisys` sets the E matrix to the identity. Omitting the arguments `e0, e1, e2` altogether results in setting $E(p) = I + (p_1 + p_2)I$.

To specify an affine PDS with a parameter-independent E matrix (e.g., $E(p) = I$), you must explicitly set $E_1 = E_2 = 0$ by typing

```
s0 = ltisys(a0,b0,c0,d0)
s1 = ltisys(a1,b1,c1,d1,0)
s2 = ltisys(a2,b2,c2,d2,0)
```

Quantification of Parameter Uncertainty

Parameter uncertainty is quantified by the range of parameter values and possibly the rates of parameter variation. This is done with the function `pvec`. The characteristics of parameter vectors defined with `pvec` are retrieved with `pvinfo`.

The parameter uncertainty range can be described as a box in the parameter space. This corresponds to cases where each uncertain or time-varying parameter p_i ranges between two empirically determined extremal values $\underline{p}_i$ and $\bar{p}_i$:

$$p_i \in [\underline{p}_i, \bar{p}_i]. \qquad \textbf{(2-1)}$$

If $p = (p_1, \ldots, p_n)$ is the vector of all uncertain parameters, (2-2) delimits a hyperrectangle of the parameter space $\mathbf{R}^n$ called the *parameter box*. Consider the example of an electrical circuit with uncertain resistor ρ and capacitor c ranging in

$$\rho \in [600, 1000] \qquad c \in [1, 5]$$

The corresponding parameter vector $p = (\rho, c)$ takes values in the box drawn in Figure 2-5. This uncertainty range is specified by the commands:

```
range = [600 1000,1 5]
p = pvec('box',range)
```

The i-th row of the matrix `range` lists the lower and upper bounds on p_i.

Similarly, bounds on the rate of variation $\dot{p}_i$ of $p_i(t)$ are specified by adding a third argument `rate` to the calling list of `pvec`. For instance, the constraints

$$0.1 \le \dot{\rho}(t) \le 1, \qquad |\dot{c}(t)| \le 0.001$$

are incorporated by

```
rate = [0.1 1, 0.001 0.001]
p = pvec('box',range,rate)
```

All parameters are assumed to be time-invariant when `rate` is omitted. Slowly varying parameters can be specified in this manner. In general, robustness against fast parameter variations is more stringent than robustness against constant but uncertain parameters.

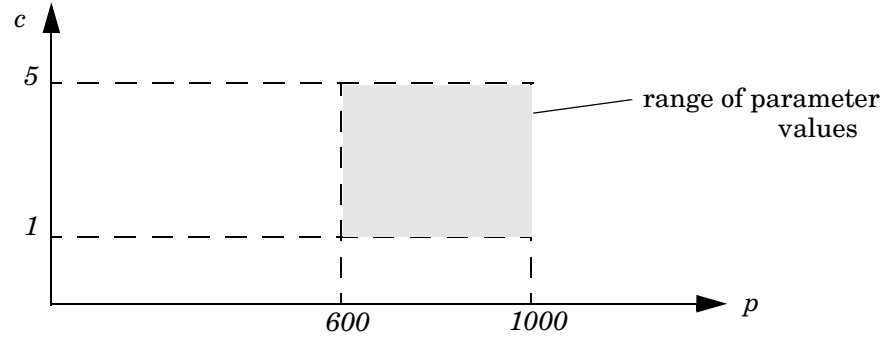


Figure 2-5: Parameter box

Alternatively, uncertain parameter vectors can be specified as ranging in a polytopic of the parameter space $\mathbf{R}^n$ like the one drawn in Figure 2-6 for $n = 2$. This polytope is characterized by the three vertices:

$$\Pi_1 = (1, 4), \qquad \Pi_2 = (3, 8), \qquad \Pi_3 = (10, 1).$$

An uncertain parameter vector p with this range of values is defined by

```
pi1=[1,4], pi2=[3,8], pi3=[10,1]
p = pvec('pol',[pi1,pi2,pi3])
```

The string `'pol'` indicates that the parameter range is defined as a polytope.

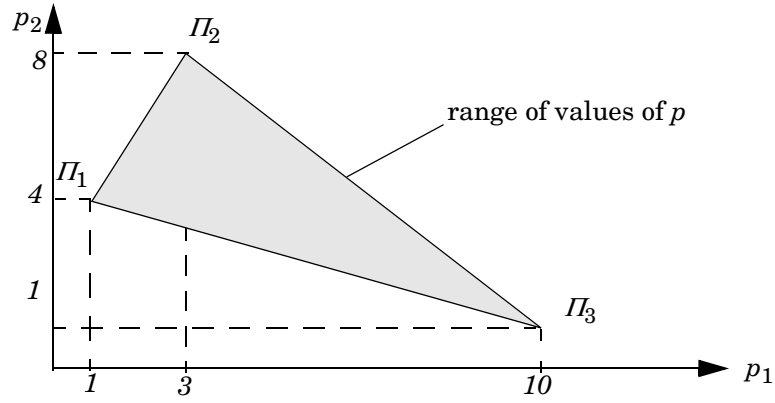


Figure 2-6: Polytopic parameter range

Simulation of Parameter-Dependent Systems

The function `pdsimul` simulates the time response of affine parameter-dependent systems along given parameter trajectories. The parameter vector $p(t)$ is required to range in a box (type `'box'` of `pvec`). The parameter trajectory is defined by a function `p = fun(t)` returning the value of p at time t. For instance,

```
pdsimul(pds,'traj')
```

plots the step response of a single-input parameter-dependent system `pds` along the parameter trajectory defined in `traj.m`. To obtain other time responses, specify an input signal as in

```
pdsimul(pds,'traj',1,'sin')
```

This command plots the response to a sine wave between $t = 0$ and $t = 1$ s. Multi-input multi-output (MIMO) systems are simulated similarly by specifying an appropriate input function.

From Affine to Polytopic Models

Affine parameter-dependent models

$$S(p) = \begin{pmatrix} A(p)+jE(p) & B(p) \\ C(p) & D(p) \end{pmatrix}$$

are readily converted to polytopic ones. Suppose for instance that each parameter p_i ranges in some interval $[\underline{p}_i, \bar{p}_i]$. The parameter vector $p = (p_1, \ldots, p_n)$ then takes values in a parameter box with 2^n corners $\Pi_1, \Pi_2, \ldots$. If the function $S(p)$ is affine in p, it maps this parameter box to some polytope of `SYSTEM` matrices. More precisely, this polytope is the convex envelope of the images $S(\Pi_1), S(\Pi_2), \ldots$ of the parameter box corners $\Pi_1, \Pi_2, \ldots$ as illustrated by the figure below.

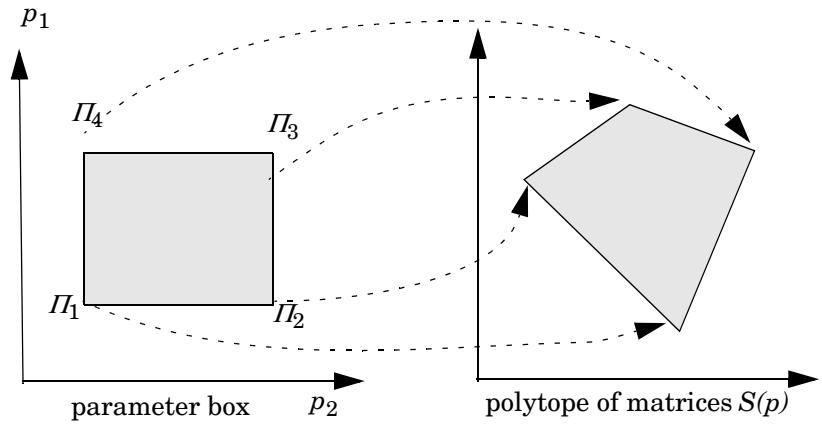


Figure 2-7: Mapping the parameter box to a polytope of systems

Given an affine parameter-dependent system, the function `aff2pol` performs this mapping and returns an equivalent polytopic model. The syntax is

```
polsys = aff2pol(affsys)
```

where `affsys` is the affine model. The resulting polytopic model `polsys` consists of the instances of `affsys` at the vertices of the parameter range.

Example

We conclude with an example illustrating the manipulation of polytopic and parameter-dependent models in the LMI Control Toolbox.

Example 2.1. Consider a simple electrical circuit with equation

$$L\frac{d^2i}{dt} + R\frac{di}{dt} + Ci = V$$

where the inductance L, the resistor R, and the capacitor C are uncertain parameters ranging in

$$L \in [10, 20], \qquad R \in [1, 2], \qquad C \in [100, 150].$$

A state-space representation of its undriven response is

$$E(L, R, C)\dot{x} = A(L, R, C)x$$

where $x^T = \left(i, \frac{di}{dt}\right)$ and

$$A(L,R,C) = \begin{pmatrix} 0 & 1 \\ -R & -C \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 0 & 0 \end{pmatrix} + L \times 0 + R\begin{pmatrix} 0 & 0 \\ -1 & 0 \end{pmatrix} + C\begin{pmatrix} 0 & 0 \\ 0 & -1 \end{pmatrix}$$

$$E(L,R,C) = \begin{pmatrix} 1 & 0 \\ 0 & L \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} + L\begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} + R \times 0 + C \times 0.$$

This affine system is specified with `psys` as follows:

```
a0 = [0 1;0 0]; e0 = [1 0;0 0]; s0 = ltisys(a0,e0)
aL = zeros(2); eL = [0 0;0 1]; sL = ltisys(aL,eL)
aR = [0 0; 1 0]; sR = ltisys(aR,0)
aC = [0 0;0  1]; sC = ltisys(aC,0)

pv=pvec('box',[10 20;1 2;100 150])
pds = psys(pv,[s0 sL sR sC])
```

The first `SYSTEM` matrix `s0` contains the state-space data for $L = R = C = 0$ while `sL`, `sR`, `sC` define the coefficient matrices of L, R, C. The range of parameter values is specified by the `pvec` command.

The results can be checked with `psinfo` and `pvinfo`:

```
psinfo(pds)

Affine parameter-dependent system with 3 parameters (4 systems)
   Each system has 2 state(s), 0 input(s), and 0 output(s)

pvinfo(pv)

Vector of 3 parameters ranging in a box
```

The system can also be evaluated for given values of L, R, C:

```
sys = psinfo(pds,'eval',[15 1.2 150])
[a,b,c,d,e] = ltiss(sys)
```

The matrices `a` and `e` now contain the values of $A(L, R, C)$ and $E(L, R, C)$ for $L = 15$, $R = 1.2$, and $C = 150$.

Finally, the polytopic counterpart of this affine model is given by

```
pols = aff2pol(pds)

psinfo(pols)

Polytopic model with 8 vertex systems
   Each system has 2 state(s), 0 input(s), and 0 output(s)
```

Linear-Fractional Models of Uncertainty

For systems with both dynamical and parametric uncertainty, a more general representation of uncertainty is the linear-fractional model of Figure 2-8 [1, 2]. In this generic model:

- The LTI system $P(s)$ gathers all known LTI components (controller, nominal models of the system, sensors, and actuators, . . .)
- The input vector u includes all external actions on the system (disturbance, noise, reference signal, . . .) and the vector y consists of all output signals generated by the system
- Δ is a structured description of the uncertainty. Specifically,

  $\Delta = \mathrm{diag}(\Delta_1, \ldots, \Delta_r)$

  where each uncertainty block Δ_i accounts for one particular source of uncertainty (neglected dynamics, nonlinearity, uncertain parameter, etc.).

  The diagonal structure of Δ reflects how each uncertainty component Δ_i enters the loop and affects the overall behavior of the true system.

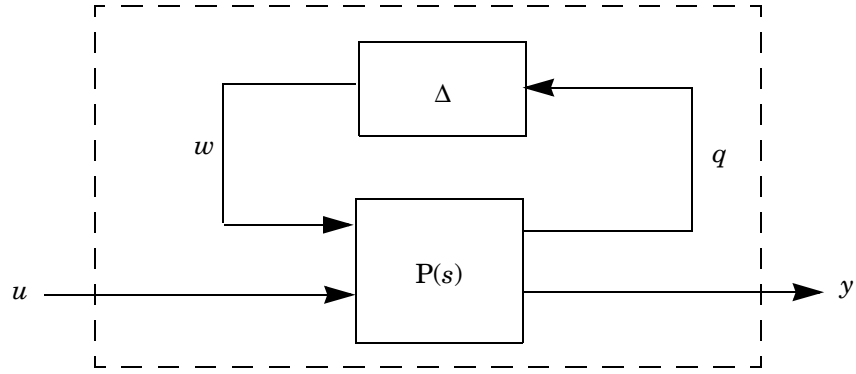
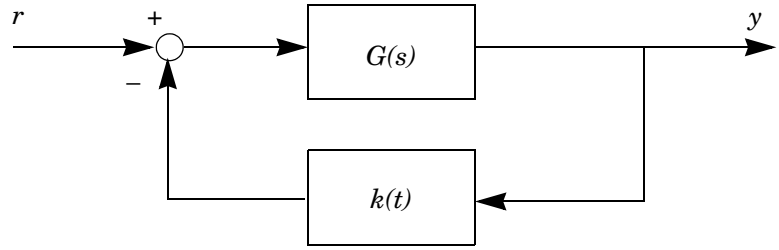


Figure 2-8: Linear-fractional uncertainty

How to Derive Such Models

Before proceeding with the specification of such uncertainty models, we illustrate how natural uncertainty descriptions can be recast into the standard linear-fractional model of Figure 2-8.

Example 2.2 . Consider the feedback loop



where

- $G(s)$ is an uncertain LTI system approximated within 5% accuracy by the second-order *nominal model*

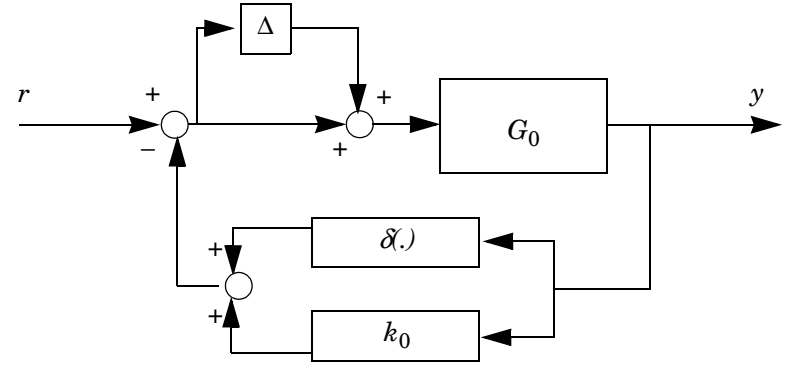
  $$G_0(s) = \frac{1}{s^2 + 0.01s + 1}$$

  In other words, $G(s) = G_0(s)(I + \Delta(s))$ where the multiplicative dynamical uncertainty $\Delta(s)$ is any stable LTI system with RMS gain less than 0.05
- $k(t)$ is a fluctuating gain modeled by

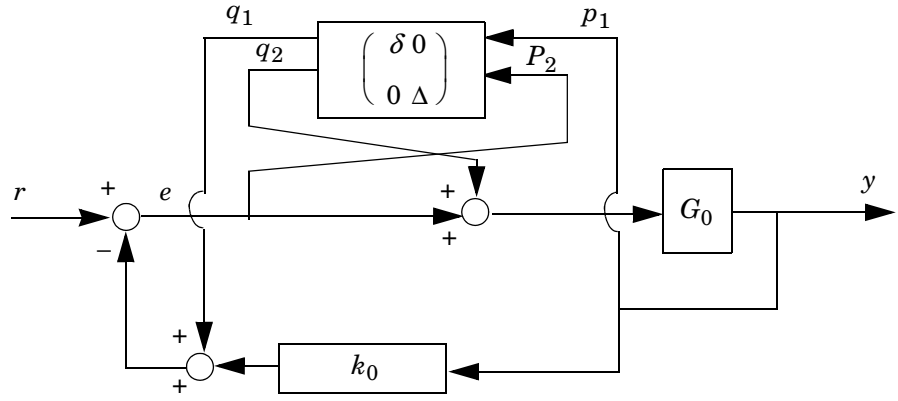
  $k(t) = k_0 + \delta(t)$

  where k_0 is the *nominal value* and $|\delta(t)| < 0.1\ k_0$.

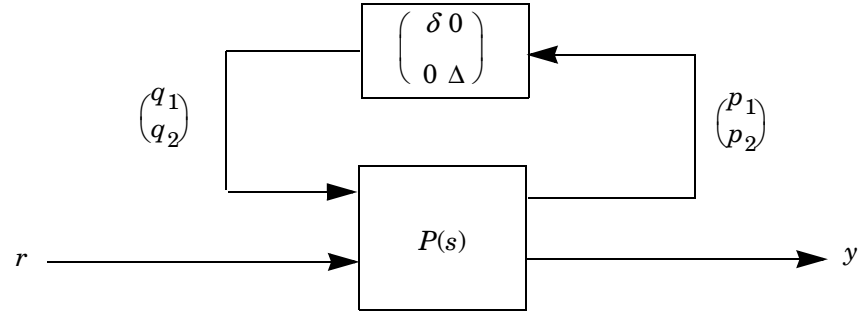
To derive a linear-fractional representation of this uncertain feedback system, first separate the uncertainty from the nominal models by rewriting the loop as



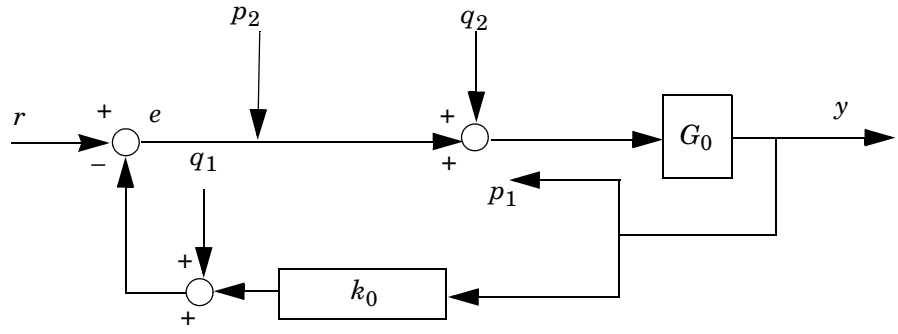
Then pull out all uncertain components and lump them together into a single block as follows:



If $\begin{pmatrix} p_1 \\ p_2 \end{pmatrix}$ and $\begin{pmatrix} q_1 \\ q_2 \end{pmatrix}$ denote the input and output vectors of the uncertainty block, the plant $P(s)$ is simply the transfer function from $\begin{pmatrix} q_1 \\ q_2 \\ r \end{pmatrix}$ to $\begin{pmatrix} p_1 \\ p_2 \\ y \end{pmatrix}$ in the diagram.



Hence the SYSTEM matrix of $P(s)$ can be computed with `sconnect` or by specifying the following interconnection:



Specification of the Uncertainty

In linear-fractional uncertainty models, each uncertainty block Δ_i is a dynamical system characterized by

- Its dynamical nature: linear time-invariant or time-varying, nonlinear memoryless, arbitrary nonlinear
- Its dimensions and structure (full block or scalar block $\Delta_i = \delta_i \times I$). Scalar blocks are used to represent uncertain parameters.
- Whether αii is real or complex in the case of scalar uncertainty $\Delta_i = \delta_i$ by I
- Quantitative information such as norm bounds or sector bounds

The properties of each individual uncertainty block Δ_i are specified with the function `ublock`. These individual descriptions are then appended with `udiag` to form a complete description of the structured uncertainty

$$\Delta = \text{diag}(\Delta_1, \ldots, \Delta_r)$$

Proper quantification of the uncertainty is important since this determines achievable levels of robust stability and performance. The basics about quantification issues are reviewed next.

Norm-Bounded Uncertainty

Norm bounds specify the amount of uncertainty in terms of RMS gain. Recall that the RMS gain or L_2 gain of a BIBO-stable system is defined as the worst-case ratio of the input and output energies:

$$\|\Delta\|_\infty = \sup_{\substack{w \in L_2 \\ w \neq 0}} \frac{\|\Delta w\|_{L_2}}{\|w\|_{L_2}}$$

where

$$\|w\|_{L_2}^2 = \int_0^\infty w^T(t)w(t)dt.$$

For additive uncertainty

$$G = G_0 + \Delta,$$

the bound on $\|\Delta\|_\infty$ is an estimate of the worst-case gap $G - G_0$ between the true system G and its linear model $G_0(s)$. For multiplicative uncertainty

$$G = G_0(I + \Delta),$$

this bound quantifies the relative gap between G and its model G_0.

Norm bounds are also useful to quantify parametric uncertainty. For instance, an uncertain parameter p can be represented as

$$p = p_0\,(1 + \delta), \qquad |\delta| < \delta_{\max}$$

where p_0 is the nominal value and $\delta_{\max}$ bounds the relative deviation from p_0. If $p \in [\underline{p}, \bar{p}\,]$, then p_0 and $\delta_{\max}$ are given by

$$p_0 = \frac{\underline{p} + \bar{p}}{2}, \qquad \delta_{\max} = \frac{\bar{p} - \underline{p}}{2p_0}$$

When Δ is LTI (i.e., when the physical system G itself is LTI), the uncertainty can be quantified more accurately by using frequency-dependent bounds of the form

$$\left\|W^{-1}(s)\Delta(s)\right\|_\infty < 1 \qquad \textbf{(2-2)}$$

where $W(s)$ is some SISO *shaping filter*. Such bounds specify different amounts of uncertainty across the frequency range and the uncertainty level at each frequency is adjusted through theshaping filter shaping filter $W(s)$. For instance, choosing

$$W(s) = \frac{2s+1}{s+100}$$

results in the frequency-weighted uncertainty bound

$$(|\Delta(j\omega)| < \beta(\omega)) := \sqrt{\frac{4\omega^2+1}{\omega^2+10^4}}$$

`fquan2.ps` drawn in Figure 2-9.

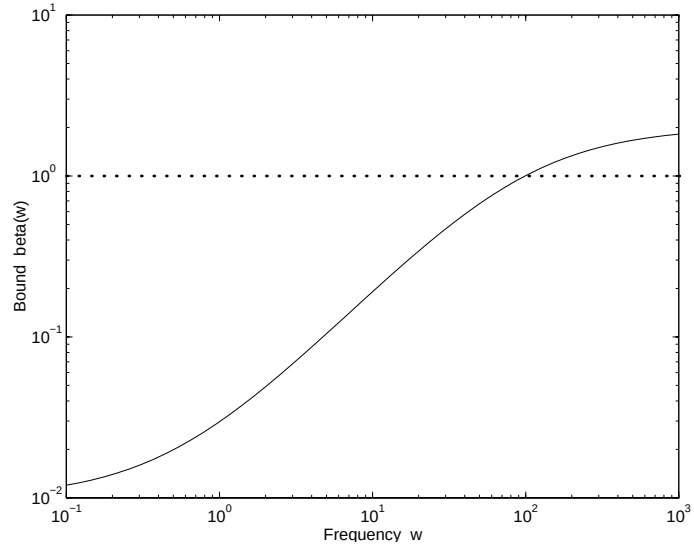


Figure 2-9: Frequency-dependent bound on |Δ(jw)|

These various norm bounds are easily specified with `ublock`. For instance, a 2-by- 3 LTI uncertainty block $\Delta(s)$ with RMS gain smaller than 10 is declared by

```
delta = ublock([2 3],10)
```

while a scalar nonlinearity $\Phi(\cdot)$ with unit RMS gain is defined by

```
phi = ublock(1,1,'nl')
```

In these commands, the first argument dimensions the block, the second argument quantifies the uncertainty by a norm bound or sector bounds, and the optional third argument is a string specifying the nature and structure of the uncertainty block. The default properties are full, complex-valued, and linear time invariant (LTI).

Scalar blocks and real parameter uncertainty are denoted by the letters `s` and `r`, respectively. For example, the command

```
delta = ublock(3,0.5,'sr')
```

declares a 3 –βψ–3 block $\Delta = \delta$ –βψ–I with repeated real parameter δ bounded by $|\delta| \leq 0.5$. Finally, setting the second argument of `ublock` to a SISO system $W(s)$ specifies a frequency-dependent norm bound on the LTI uncertainty block $\Delta(s)$. Thus,

```
delta = ublock(2, ltisys('tf',[100 0],[1 100]))
```

declares an LTI uncertainty $\Delta(s) \in \mathbf{C}^{2\times 2}$ satisfying

$$\sigma_{\max} < (\Delta(j\omega)) < |W(j\omega)|, \qquad W(s) = \frac{100s}{s+100}$$

at all frequencies ω.

After specifying each block Δ_i with `ublock`, call `udiag` to derive a complete description of the uncertainty structure

$$\Delta = \mathrm{diag}(\Delta_1, \ldots, \Delta_n).$$

For instance, the commands

```
delta1 = ublock([3 1],0.1)
delta2 = ublock(2,2,'s')
delta3 = ublock([1 2],100,'nl')
delta = udiag(delta1,delta2,delta3)
```

specify the uncertainty structure

$$\Delta = \mathrm{diag}(\Delta_1, \Delta_2, \Delta_3)$$

where

- $\Delta_1(s) \in \mathbf{C}^{3\times 1}$ and $\|\Delta_1\|_\infty < 0.1$

- $\Delta_2 = \delta$ by I_2 with $\delta \in \mathbf{C}$ and $|\delta| < 2$
- $\Delta_3(\cdot)$ is a nonlinear operator with two inputs and one output and satisfies $\|\Delta_3\|_\infty < 100$.

Finally, `uinfo` displays the characteristics of uncertainty structures declared with `ublock` and `udiag`.

```
uinfo(delta)
```

```
block    dims    type    real/cplx    full/scal    bounds
1        3x1     LTI     c            f            norm <= 0.1
2        2x2     LTI     c            s            norm <= 2
3        1x2     NL      r            f            norm <= 100
```

Sector-Bounded Uncertainty

The behavior of uncertain dynamical components can also be quantified in terms of sector bounds on their time-domain response [5, 4]. A (possibly nonlinear) BIBO-stable system $\phi(\cdot)$ is said to be in the sector $\{a, b\}$ if the mapping

$$\phi : u \in L_2 \alpha\, y \in L_2$$

satisfies a quadratic constraint of the form

$$\int_0^\infty (y(t) - au(t))^T (y(t) - bu(t)) dt \le 0 \quad \text{for all } u \in L_2$$

As a special case, passive systems satisfy

$$\int_0^\infty y(t)^T u(t) dt \ge 0$$

which corresponds to the sector $\{0, +\infty\}$. Note that sector bounds can always be converted to norm bounds since ϕ is in the sector $\{a, b\}$ if and only if

$$\left\| \phi - \frac{a+b}{2} \right\|_\infty < \frac{|a-b|}{2}$$

If ϕ is passive, a bilinear transformation maps ϕ to the operator $(I - \phi)(I + \phi)^{-1}$ of norm less than one.

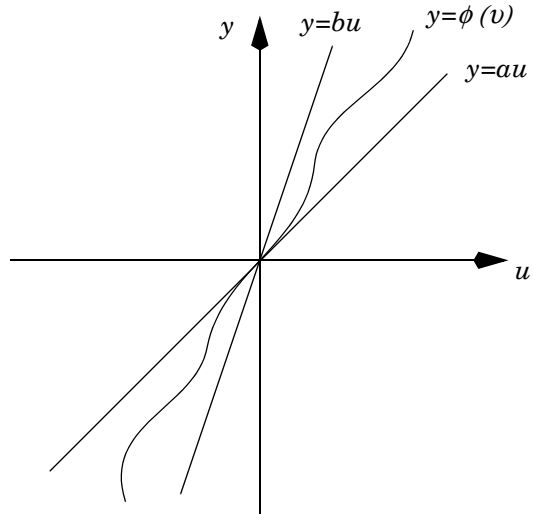
The system $\phi(\cdot)$ is called *memoryless* if $y(t)$ only depends on the input value $u(t)$ at time t, i.e.,

$$y(t) = \phi(u(t)).$$

For memoryless systems, the sector condition reduces to the "instantaneous" property

$$(y(t) - au(t))^T (y(t) - bu(t)) \eth 0 \qquad \text{at all time } t.$$

In the scalar case, this means that the response $y = \phi(u)$ lies in the sector delimited by the two lines $y = au$ and $y = bu$ as illustrated below.



A simple example is the nonlinear spring $y = k(u)$ with

$$k(u) = \begin{cases} 2u & \text{if } |u| \le 1 \\ u+1 & \text{if } u > 1 \\ u-1 & \text{if } u < -1 \end{cases}$$

The response of this spring lies in the sector {1, 2} as is seen from Figure 2-10. This scalar memoryless nonlinearity is specified with `ublock` by

```
k = ublock(1,[1 2],'nlm')
```

Robustness in the face of sector-bounded uncertainty is analyzed by the circle and Popov criteria.

From Affine to Linear-Fractional Models

Since some uncertain systems are more naturally specified as *affine* parameter-dependent systems, the function `aff2lft` is provided to derive a linear-fractional representation of their parametric uncertainty. All uncertain parameters are assumed real and are repeated in the Δ structure when necessary. For instance, the system

$$\frac{dx}{dt} = \begin{pmatrix} p_1 & 1 \\ -p_1 & p_1 + 2p_2 \end{pmatrix} x \qquad \textbf{(2-3)}$$

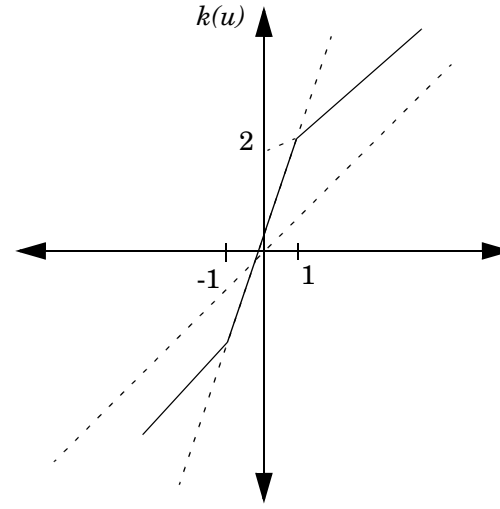


Figure 2-10: Nonlinear spring $y = k(u)$

with constant $p_1 \in [2, 10]$ and time-varying $p_2 \in [-1, 3]$ is specified by

```
s0 = ltisys([0 1,0 0])
s1 = ltisys([1 0, 1 1],0) % p1
```

```
s2 = ltisys([0 0,0 2],0) % p2
pv = pvec('box',[2 10, 1 3],[0 0,-Inf Inf])
pds = psys(pv,[s0 s1 s2])
```

To derive the linear-fractional representation of the uncertainty on p_1, p_2, type

```
[P,delta] = aff2lft(pds)
```

On output:

- The nominal plant `P` corresponds to the average values of p_1 and p_2 . Specifically, closing the interconnection of Figure 2-8 with $\Delta \equiv 0$ yields the system (2-4) for $p_1 = 6$ and $p_2 = 1$, as confirmed by

  ```
  cls = slft(zeros(3),P)
  a = ltiss(cls)

  a =
     6   1
      6  8
  ```

- The uncertainty structure `delta` consists of real scalar blocks as confirmed by

  ```
  uinfo(delta)

  block    dims       type     real/cplx    full/sc    bounds
                                            al
  1        2-by-2     LTI      r            s          norm <= 4
  2        1-by-1     LTV      r            s          norm <= 2
  ```

  Note that the parameter p_1 is repeated and that the norm bound corresponds to the maximal deviation from the average value of the parameter.

The corresponding block diagram appears in Figure 2-11.

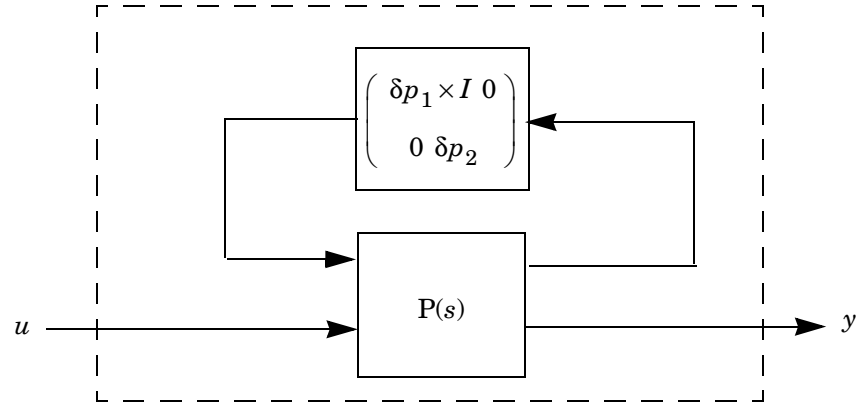


Figure 2-11: Linear-fractional representation of the system (2-4)

References

[1] Doyle, J.C. and G. Stein, "Multivariable Feedback Design: Concepts for a Classical/Modern Synthesis," *IEEE Trans. Aut. Contr*., AC-26 (1981), pp. 4-16.

[2] Doyle, J.C., "Analysis of Feedback Systems with Structured Uncertainties," *IEE Proc*., vol. 129, pt. D (1982), pp. 242–250.

[3] Boyd, S., L. El Ghaoui, E. Feron, V. Balakrishnan, *Linear Matrix Inequalities in Systems and Control Theory*, SIAM books, Philadelphia, 1994.

[4] Vidyasagar, M., *Nonlinear System Analysis*, Prentice-Hall, Englewood Cliffs, 1992.

[5] Zames, G., "On the Input-Output Stability of Time-Varying Nonlinear Feedback Systems, Part I and II," *IEEE Trans. Aut. Contr*., AC–11 (1966), pp. 228-238 and 465-476.

3

Robustness Analysis

Quadratic Lyapunov Functions 3-3
LMI Formulation 3-4
Quadratic Stability 3-6
Maximizing the Quadratic Stability Region 3-8
Decay Rate 3-9
Quadratic $H\infty$ Performance 3-10

Parameter-Dependent Lyapunov Functions 3-12
Stability Analysis 3-14

μ Analysis 3-17
Structured Singular Value 3-17
Robust Stability Analysis 3-19
Robust Performance 3-21

The Popov Criterion 3-24
Real Parameter Uncertainty 3-25

Example 3-28

References 3-32

Control systems are often designed for a simplified model of the physical plant that does not take into account all sources of uncertainty. *A-posteriori* robustness analysis is then necessary to validate the design and obtain guarantees of stability and performance in the face of plant uncertainty. The LMI Control Toolbox offers a variety of tools to assess robust stability and robust performance. These tools cover most available Lyapunov-based and frequency-domain analysis techniques.

- Quadratic stability/performance
- Tests involving parameter-dependent Lyapunov functions
- Mixed-μ analysis
- The Popov criterion

Each test is geared to a particular class of uncertainty, for example, real parameter uncertainty for parameter-dependent Lyapunov functions, or memoryless nonlinearities for the Popov criterion. Hence users should select the most appropriate tool for their problem. Since all of these tests are based on *sufficient* conditions, they are only productive when they succeed in establishing robust stability or performance.

Quadratic Lyapunov Functions

The notions of quadratic stability and quadratic performance are useful to analyze linear time-varying systems

$$E(t)\dot{x}(t) = A(t)x(t), \qquad x(0) = x_0.$$

Given such systems, a sufficient condition for asymptotic stability is the existence of a positive-definite quadratic Lyapunov function

$$V(x) = x^T P x$$

such that

$$\frac{dV(x(t))}{dt} < 0$$

along all state trajectories. In terms of $Q := P^{-1}$, this is equivalent to

$$A(t)QE(t)^T + E(t)QA(t)^T < 0 \qquad \textbf{(3-1)}$$

at all times t.

Assessing quadratic stability is not tractable in general since (3-1) places an infinite number of constraints on Q. However, (3-1) can be reduced to a finite set of LMI contraints in the following cases:

1 $A(t)$ and $E(t)$ are fixed affine functions of some time-varying parameters $p_1(t), \ldots, p_n(t)$.

$$A(t) = A_0 + p_1(t)A_1 + \ldots + p_n(t)A_n$$

$$E(t) = E_0 + p_1(t)E_1 + \ldots + p_n(t)E_n.$$

This is referred to as an *affine parameter-dependent* model.

2 $A(t) + jE(t)$ ranges in a fixed polytope of matrices, that is,

$$A(t) = \alpha_1(t)A_1 + \ldots + \alpha_n(t)A_n$$

$$E(t) = \alpha_1(t)E_1 + \ldots + \alpha_n(t)E_n$$

with $\alpha_i(t) \geq 0$ and $\sum_{i=1}^{n} \alpha_i(t) = 1$. This is referred to as a *polytopic* model.

The first case corresponds to systems whose state-space equations depend affinely on time-varying physical parameters, and the second case to

time-varying systems modeled by an envelope of LTI systems (see "Uncertain State-Space Models" on page 2-14 for details). Note that quadratic Lyapunov functions guarantee stability for *arbitrarily fast* time variations, which is their main source of conservatism.

Quadratic Lyapunov functions are also useful to assess robust RMS performance. Specifically, a time-varying system

$$\begin{cases} E(t)\dot{x} = A(t)x + B(t)u \\ \quad\quad y = C(t)x + D(t)u \end{cases} \tag{3-2}$$

with zero initial state satisfies

$$\|y\|_{L_2} < \gamma \|u\|_{L_2}$$

for all bounded input $u(t)$ if there exists a positive-definite Lyapunov function $V(x) = x^T P x$ such that

$$\frac{dV(x)}{dt} + y^T y - \gamma^2 u^T u < 0$$

In terms of $Q := P^{-1}$, this is equivalent to the family of "Bounded Real Lemma" inequalities

$$\begin{pmatrix} A(t)QE(t)^T + E(t)QA(t)^T & B(t) & E(t)QC(t)^T \\ B(t)^T & -\gamma I & D(t)^T \\ C(t)QE(t)^T & D(t) & -\gamma I \end{pmatrix} < 0$$

The smallest γ for which such a Lyapunov function exists is called the quadratic H_∞ performance. This is an upper bound on the worst-case RMS gain of the system (3-2).

LMI Formulation

Sufficient LMI conditions for quadratic stability are as follows [8, 1, 2]:

Affine models: consider the parameter-dependent model

$$E(p)\dot{x} = A(p)x,\ \ A(p) = A_0 + p_1A_1 + \ldots + p_nA_n,$$
$$E(t)\ = E_0 + p_1E_1 + \ldots + p_nE_n \tag{3-3}$$

where $p_i(t) \in [\underline{p}_i, \bar{p}_i]$, and let

$$V\ = \{(\omega_1, \ldots, \omega_n) : \omega_i \in \{\underline{p}_i, \bar{p}_i\}\}$$

denote the set of corners of the corresponding parameter box. The dynamical system (3-3) is quadratically stable if there exist symmetric matrices Q and $\{M_i\}_{i=1}^n$ such that

$$A(\omega)QE(\omega)^T + E(\omega)QA(\omega)^T + \sum_i \omega_i^2 M_i < 0 \text{ for all } \omega \in V \tag{3-4}$$

$$A_iPE_i^T + E_iPA_i^T + M_i \geq 0 \text{ for } i = 1,\ldots,n \tag{3-5}$$

$$M_i \geq 0 \tag{3-6}$$

$$Q > I \tag{3-7}$$

Polytopic models: the polytopic system

$$E(t)\dot{x} = A(t)x, \qquad A(t) + jE(t) \in \text{Co}\{A_1 + jE_1, \ldots, A_n + jE_n\},$$

is quadratically stable if there exist a symmetric matrix Q and scalars $t_{ij} = t_{ji}$ such that

$$A_iQE_j^T + E_jQA_i^T + A_jQE_i^T + E_iQA_j^T < 2t_{ij}I \text{ for } i,j \in \{1,\ldots,n\} \tag{3-8}$$

$$Q > I \tag{3-9}$$

$$\begin{bmatrix} t_{11} & \cdots & t_{1n} \\ \vdots & \ddots & \vdots \\ t_{1n} & \cdots & t_{nn} \end{bmatrix} < 0 \qquad \textbf{(3-10)}$$

Note that these LMI conditions are in fact necessary and sufficient for quadratic stability whenever

- In the affine case, no parameter p_i enters both $A(t)$ and $E(t)$, that is, $A_i = 0$ or $E_i = 0$ for all i. The conditions (3-5)–(3-6) can then be deleted and it suffices to check (3-1) at the corners ω of the parameter box.
- In the polytopic case, either $A(t)$ or $E(t)$ is constant. It then suffices to solve (3-8)–(3-9) for $t_{ij} = 0$.

Similar sufficient conditions are available for robust RMS performance. For polytopic systems, for instance, LMI conditions guaranteeing the RMS performance γ are as follows.

A symmetric matrix Q and scalars $t_{ij} = t_{ji}$ for $i,j \in \{1, \ldots, n\}$ exist such that

$$\begin{pmatrix} A_iQE_j^T + E_jQA_i^T + A_jQE_i^T + E_jQA_i & H & H \\ B_i^T + B_j^T & -2\gamma I & H \\ C_iQE_i^T + C_iQE_j^T & D_i + D_j & -2\gamma I \end{pmatrix} < 2t_{ij} \times 1$$

$$Q > 0$$

$$\begin{bmatrix} t_{11} & \cdots & t_{1n} \\ \vdots & & \vdots \\ t_{1n} & \cdots & t_{nn} \end{bmatrix} < 0$$

Quadratic Stability

The function `quadstab` tests the quadratic stability of polytopic or affine parameter-dependent systems. The corresponding LMI feasibility problem is solved with `feasp`.

Example 3.1. Consider the time-varying system

$$\dot{x} = A(t)x$$

where $A(t) \in \mathrm{Co}\{A_1, A_2, A_3\}$ with

$$A_1 = \begin{pmatrix} 0 & 1 \\ -2 & -0.2 \end{pmatrix},\ A_2 = \begin{pmatrix} 0 & 1 \\ -2.2 & -0.3 \end{pmatrix},\ A_3 = \begin{pmatrix} 0 & 1 \\ -1.9 & -0.1 \end{pmatrix}$$

This polytopic system is specified by

```
s1 = ltisys(a1)
s2 = ltisys(a2)
s3 = ltisys(a3)
polsys = psys([s1 s2 s3])
```

To test its quadratic stability, type

```
[tmin,P] = quadstab(polsys)

Solver for LMI feasibility problems L(x) < R(x)
   This solver minimizes t subject to L(x) < R(x) + t*I
   The best value of t should be negative for feasibility

Iteration : Best value of t so far

1          0.040521
2           0.032876

Result: best value of t:  0.032876
   f-radius saturation: 0.000% of R = 1.00e+08

This system is quadratically stable

tmin =

    3.2876e 02
```

Collectively denoting the LMI system (3-8)–(3-10) by $A(x) < 0$, `quadstab` assesses its feasibility by minimizing τ subject to $A(x) < \tau I$ and returns the global minimum `tmin` of this problem. Hence the system is quadratically stable if and only if `tmin` < 0. In this case, the second output `P` is the Lyapunov matrix proving stability.

Maximizing the Quadratic Stability Region

This option of `quadstab` is only available for affine parameter-dependent systems where the E matrix is constant and each parameter $p_i(t)$ ranges in an interval

$$p_i(t) \in [\underline{p}_i, \bar{p}_i]$$

Denoting the center and radius of each interval by $u_i = \frac{1}{2}(\underline{p}_i + \bar{p}_i)$ and $\delta_i = \frac{1}{2}(\underline{p}_i - \bar{p}_i)$, the maximal quadratic stability region is defined as the largest portion of the parameter box where quadratic stability can be established. In other words, the largest dilatation factor such that the system is quadratically stable whenever

$$p_i(t) \in [\mu i - \theta\delta_i, i + \theta\delta_i]$$

Computing this largest factor is a generalized eigenvalue minimization problem (see `gevp`). This optimization is performed by `quadstab` when the first entry of the options vector is set to 1.

Example 3.2. Consider the second-order system

$$\dot{x} = A(k, f)x$$

where

$$A(k,f) = \begin{pmatrix} 0 & 1 \\ -k & -f \end{pmatrix}, \quad k(t) \in [10, 12], \quad f(t) \in [0.5, 0.7]$$

This system is entered by

```
s0 = ltisys([0 1;0 0])
s1 = ltisys([0 0; 1 0],0) % k
s2 = ltisys([0 0;0  1],0) % f

pv = pvec('box',[10 12;0.5 0.7]) % parameter box
affsys = psys(pv,[s0 s1 s2])
```

To compute the largest dilatation of the parameter box where quadratic stability holds, type

```
[marg,P] = quadstab(affsys,[1 0 0])
```

The result is

```
marg =

   1.4908e+00
```

which means 149% of the specified box, that is,

$k(t) \in [9.51, 12.49], \qquad f(t) \in [0.451, 0.749]$

Note that `marg` Š 1 implies quadratic stability in the prescribed parameter box.

Decay Rate

For the time-varying system

$E(t)\dot{x} = A(t)x,$

the quadratic decay rate α^* is the smallest α such that

$A(t)QE(t)^T + E(t)QA(t)^T < \alpha E(t)QE(t)^T$

holds at all times for some fixed $Q > 0$. The system is quadratically stable if and only if $\alpha^* < 0$, in which case α^* is an upper bound on the rate of return to equilibrium. Specifically,

$x^T(t)Q^{-1}x(t) < e^{\alpha^* t}\,(x(0)^T Q^{-1} x(0)).$

Note that $-\frac{\alpha^*}{2}$ is also the largest β such that the shifted system

$E(t)\dot{x} = (A(t) + \beta E(t))x$

is quadratically stable.

For affine or polytopic models, the decay rate computation is attacked as a generalized eigenvalue minimization problem (see `gevp`). This task is performed by the function `decay`.

Example 3.3. For the time-varying system considered in Example 3.1, the decay rate is computed by

```
[drate,P] = decay(polsys)
```

This command returns

```
drate =

    5.6016e 02
```

```
P =
   6.0707e 01 1.9400e 02
   1.9400e 02 3.1098e 01
```

Quadratic H_∞ Performance

The function `quadperf` computes the quadratic H_∞ performance of an affine parameter-dependent system

$$E(p)\dot{x} = A(p)x + B(p)u$$

$$y = C(p)x + D(p)u$$

or of a polytopic system

$$E(t)\dot{x} = A(t)x + B(t)u$$

$$y = C(t)x + D(t)u$$

where

$$S(t) = \begin{pmatrix} A(t)+jE(t) & B(t) \\ C(t) & D(t) \end{pmatrix} \in \mathrm{Co}\{S_1,\ldots,S_n\}$$

Recall that the quadratic H_∞ performance γ_{qp} is an upper bound on the worst-case input/output RMS gain of the system. Specifically

$$\|y\|_{L_2} < \gamma_{qp}\,\|u\|_{L_2}$$

holds for all bounded input u when the initial state is zero. The function `quadperf` can also be used to test whether the worst-case RMS gain does not exceed a given value $\gamma > 0$, or to maximize the portion of the parameter box where this level γ is not exceeded.

Example 3.4. Consider the second-order system with time-varying mass and damping

$$\begin{pmatrix} 1 & 0 \\ 0 & m(t) \end{pmatrix}\begin{pmatrix} \dot{x} \\ \ddot{x} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -2 & -f(t) \end{pmatrix}\begin{pmatrix} x \\ \dot{x} \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \end{pmatrix}u$$

$$y = x, \qquad m(t) \in [0.8,\ 1.2], \qquad f(t) \in [0.4,\ 0.6]$$

This affine parameter-dependent model is specified as

```
s0 = ltisys([0 1; 2 0],[0;1],[1 0],0,[1 0;0 0])
s1 = ltisys(zeros(2),[0;0],[0 0],0,[0 0;0 1]) % m
s2 = ltisys([0 0;0  1],[0;0],[0 0],0,0) % f

pv = pvec('box',[0.8 1.2 ; 0.4 0.6])
affsys = psys(pv,[s0 s1 s2])
```

and its quadratic H_∞ performance from u to y is computed by

```
qperf = quadperf(affsys)

qperf =


   6.083e+00
```

This value is larger than the nominal LTI performance for $m = 1$ and $f = 0.5$ as confirmed by

```
nomsys = psinfo(affsys,'eval',[1 0.5])

norminf(nomsys)

   1.4311e+00
```

Parameter-Dependent Lyapunov Functions

The robustness test discussed next is applicable to affine parameter-dependent systems or time-invariant uncertain systems described by a polytope of models. To prove the robust stability of such systems, we seek a quadratic Lyapunov function that depends on the uncertain parameters or the polytopic coordinates in the case of a polytopic model. The resulting tests are less conservative than quadratic stability when the parameters are *constant* or slowly varying [6]. Moreover, available bounds on the rates of parameter variation can be explicitly taken into account.

Affine models: for an affine parameter-dependent system

$$E(p)\dot{x} = A(p)x$$

with parameter vector $p = (p_1, \ldots, p_n) \in \mathbf{R}^n$, we seek positive definite Lyapunov functions of the form

$$V(x, p) = x^T Q(p)^{-1} x$$

where

$$Q(p) = Q_0 + p_1 Q_1 + \ldots + p_n Q_n$$

For such Lyapunov functions, the stability condition $\frac{dV(x,p)}{dt} < 0$ is equivalent to

$$E(p)Q(p)A^T(p) + A(p)Q(p)E(p)^T - E(p)\frac{dQ}{dt}E(p)^T < 0. \qquad \textbf{(3-11)}$$

Given interval bounds

$$p_i \in [\underline{p}_i, \bar{p}_i], \qquad \in [\underline{v}_i, \bar{v}_i],$$

on each p_i and its time derivative $\frac{dp_i}{dt}$, the vectors p and $\frac{dp}{dt}$ range in n-dimensional "boxes." If V and T list the corners of these boxes, (3-11) holds for all parameter trajectories if the following LMI problem is feasible [6].

Find symmetric matrices $Q_0, Q_1, \ldots, Q_n$, and $\{M_i\}_{i=1}^n$ such that

- For all $(\omega, \tau) \in V \times T$

$$E(\omega)Q(\omega)A(\omega)^T + A(\omega)Q(\omega)E(\omega)^T - E(\omega)(Q(\tau) - Q_0)E(\omega)^T + \sum_i \omega_i^2 M_i < 0$$

- For $\omega \in V$ and $i = 1, \ldots, n$

$$A_i^T Q_i E(\omega) + E(\omega)^T Q_i A_i + A_i^T Q(\omega) E_i + E_i^T Q(\omega) A_i + A(\omega)^T Q_i E_i + E_i^T Q_i A(\omega) - E_i^T (Q(\tau) - Q_0) + M_i \geq 0$$

- $Q(\omega) > I$ for all $\omega \in V$
- $M_i \succeq 0$

Note that these conditions reduce to quadratic stability when the rates of variation $\frac{dp_i}{dt}$ are allowed to range in $(-\infty, +\infty)$. In such cases indeed, $Q_1, \ldots, Q_n$ must be set to zero for feasibility.

Polytopic models: A similar extension of the quadratic stability test is available for time-invariant polytopic systems

$$E\dot{x} = Ax$$

where one of the matrices A, E is constant and the other uncertain. Assuming that E is constant and A ranges in the polytope

$$A \in \{\alpha_1 A_1 + \ldots + \alpha_n A_n : \alpha_i \succeq 0, \alpha_1 + \ldots + \alpha_n = 1\},$$

we seek a Lyapunov function of the form $V(x, \alpha) = x^T Q(\alpha)^{-1} x$ where

$$Q(\alpha) = \alpha_1 Q_1 + \ldots + \alpha_n Q_n$$

Using such Lyapunov functions, sufficient conditions for stability over the entire polytope are as follows.

There exist symmetric matrices $Q_1, \ldots, Q_n$, and scalars $t_{ij} = t_{ji}$ such that

$$A_i Q_j E^T + E Q_j A_i^T + A_j Q_i E^T + E Q_i A_j^T < 2t_{ij}$$

$$Q_j > I$$

$$\begin{bmatrix} t_{11} & \cdots & t_{1n} \\ \vdots & \ddots & \vdots \\ t_{1n} & \cdots & t_{nn} \end{bmatrix} < 0$$

Stability Analysis

Given an affine or polytopic system, the function `pdlstab` seeks a parameter-dependent Lyapunov function establishing robust stability over a given parameter range or polytope of models. The syntax is similar to that of `quadstab`. For an affine parameter-dependent system with two uncertain parameters p_1 and p_2, for instance, `pdlstab` is invoked by

```
[tmin,Q0,Q1,Q2] = pdlstab(ps)
```

Here `ps` is the system description and includes available information on the range of values and rates of variation of each parameter (see `psys` and `pvec` for details). If no rates of variation are specified, the parameters are regarded as *time-invariant*. The function `pdlstab` determines whether the sufficient LMI conditions listed above are feasible. Feasibility is established when `tmin` < 0. In such cases, a parameter-dependent Lyapunov function proving stability is $V(x, p) = x^T Q(p)^{-1} x$ with

$$Q(p) = Q_0 + p_1 Q_1 + p_2 Q_2$$

Remark In the case of time-invariant uncertain parameters, `pdlstab` can be combined with a branch-and-bound scheme to reduce conservatism and enlarge the computed stability region. Specifically, if `pdlstab` fails to prove stability over the entire parameter range, you can split the range into smaller regions and try to establish stability on each subregion independently.

Example 3.5. Consider the second-order system

$$\begin{pmatrix} \dot{x} \\ \ddot{x} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -k(t) & -f(t) \end{pmatrix} \begin{pmatrix} x \\ \dot{x} \end{pmatrix}$$

where the stiffness $k(t)$ and damping $f(t)$ range in

$$k(t) \in [5, 10], \qquad f(t) \in [0.01, 0.1]$$

and their rates of variation are bounded by

$$\frac{dk}{dt} < 0.01, \qquad \frac{df}{dt} < 1 \tag{3-12}$$

This system and parameter data are specified by

```
s0 = ltisys([0 1;0 0])
s1 = ltisys([0 0; 1 0],0)
s2 = ltisys([0 0;0  1],0)
pv = pvec('box',[5 10 ; 0.01 0.1],[ 0.01 0.01 ;  1 1])
ps = psys(pv,[s0 s1 s2])
```

The second argument of `pvec` defines the range of parameter values while the third argument specifies the bounds on their rate of variation.

This system is not quadratically stable over the specified parameter box as confirmed by

```
tmin = quadstab(ps)
tmin =

   8.0118e 04
```

Nevertheless, it turns out to be robustly stable when the parameter variations do not exceed the maximum rates (3-12), as shown by

```
[tmin,Q0,Q1,Q2] = pdlstab(ps)

Solver for LMI feasibility problems L(x) < R(x)
   This solver minimizes t subject to L(x) < R(x) + t*I
   The best value of t should be negative for feasibility

Iteration : Best value of t so far

1  0.084914
2  0.011826
3  0.011826
4  1.878414e 03
```

```
5  1.878414e 03
6  4.846478e 04
7  1.528537e 04
8   7.832830e 04

Result: best value of t:  7.832830e 04
   f-radius saturation: 0.003% of R = 1.00e+07

This system is stable for the specified parameter trajectories
```

This makes `pdlstab` a useful tool to analyze systems with constant or slowly-varying parameters.

μ Analysis

μ analysis investigates the robust stability or performance of systems with linear time-invariant linear-fractional uncertainty. It is also applicable to time-invariant parameter-dependent systems by first deriving an equivalent linear-fractional representation with `aff2lft` (see "From Affine to Linear-Fractional Models" on page 2-32 for details). Nonlinear and/or time-varying uncertainty is addressed by the Popov criterion discussed in the next section.

Structured Singular Value

μ analysis is based on the concept of structured singular value (SSV or μ) [3, 12, 13, 10]. Consider the algebraic loop of Figure 3-1 where $M \in C^{n \times m}$ and $\Delta = \mathrm{diag}(\Delta_1, \ldots, \Delta_n)$ is a structured perturbation characterized by

- The dimensions of each block Δ_i
- Whether Δ_i is a complex or real-valued matrix
- Whether Δ_i is a full matrix or a scalar matrix of the form $\Delta_i = \delta_i \times I$

We denote by Δ the set of perturbations with this particular structure. The algebraic loop of Figure 3-1 is well posed if and only if $I - M\Delta$ is invertible. Measuring the smallest amount of perturbation $\Delta \in \Delta$ needed to destroy well-posedness is the purpose of the SSV.

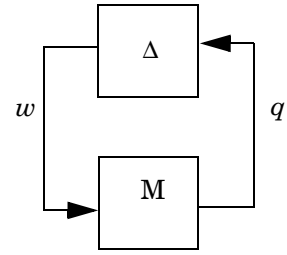


Figure 3-1: Static μ problem

Formally, the SSV of M with respect to the perturbation structure Δ is defined as [10]

$$\mu(M) := \frac{1}{\min\{\sigma_{\max}(\Delta) \ : \ \Delta \in \ \Delta \text{ and } I - M\Delta \text{ is singular}\}}$$

with the convention $\mu_\Delta(M) = 0$ if $I - M\Delta$ is invertible for all $\Delta \in \Delta$. From this definition, $I - M\Delta$ remains invertible as long as $\Delta \in \Delta$ satisfies

$$\sigma_{\max}(\Delta) < 1/\mu_\Delta(M)$$

That is, as long as the size of Δ does not exceed $K_\Delta := 1/\mu_\Delta(M)$. The critical size K is called thewell-posedness margin well-posedness margin. Note that $\mu_\Delta(M) = \sigma_{\max}(M)$ when $\Delta = C^{m\times n}$ (unstructured case). Thus $\mu_\Delta(M)$ extends the notion of largest singular value to the case of structured perturbations.

Computing $\mu_\Delta(M)$ is an NP-hard problem in general. However, a conservative estimate of the margin K_Δ (upper bound μ) can be computed by solving an LMI problem [4]. Assuming for simplicity that the blocks Δ_i are square, the variables in this LMI problem are two scaling matrices

$$D = \text{diag}(D_1, \ldots, D_n), \qquad G = \text{diag}(G_1, \ldots, G_n)$$

with the same block-diagonal structure as Δ and where

- $D_i = d_i \times I$ with $d_i > 0$ if Δ_i is a full block, and $D_i = D_i^H > 0$ otherwise.
- $G_i = 0$ if Δ_i is complex-valued, $G_i = g_i \times I$ with $g_i \in \mathbf{R}$ if Δ_i is a real-valued full block, and $G_i = G_i^H$ if $\Delta_i = \delta_i \times I$ with δ_i real.

Denoting by D and G the sets of D, G scalings with such properties, an upper bound on $\mu_\Delta(M)$ is given by

$$\nu_\Delta = \sqrt{\max(0, \alpha^*)}$$

where α^* is the global minimum of the following generalized eigenvalue minimization problem [4]:

Minimize α over $D \in D$ and $G \in G$ subject to

$$M^H D M + j(GM - M^H G) < \alpha D, \ D > I$$

The μ upper bound $\nu_\Delta(M)$ and the optimal D, G scalings are computed by the function `mubnd`. Its syntax is

```
[mu,D,G] = mubnd(M,delta)
```

where `delta` specifies the perturbation structure Δ (use `ublock` and `udiag` to define `delta`, see "Norm-Bounded Uncertainty" on page 2-27 and "Sector-Bounded Uncertainty" for details). When each block Δ_i is bounded by 1, the output `mu` is equal to $\nu_\Delta(M)$. If different bounds β_i are used for each Δ_i, the interpretation of `mu` is as follows:

The interconnection of Figure 3-1 is well posed for all $\Delta \in \Delta$ satisfying

$$\sigma_{\max}(\Delta_i) < \frac{\beta_i}{\mathrm{mu}}$$

For instance, `1/mu = 0.9` means that well-posedness is guaranteed for perturbation sizes that do not exceed 90% of the prescribed bounds.

Robust Stability Analysis

The structured singular value μ is useful to assess the robust stability of time-invariant linear-fractional interconnections (see Figure 3-2). Here $P(s)$ is a given LTI system and $\Delta(s) = \mathrm{diag}(\Delta_1(s), \ldots, \Delta_n(s))$ is a norm- or sector-bounded linear time-invariant uncertainty with some prescribed structure.

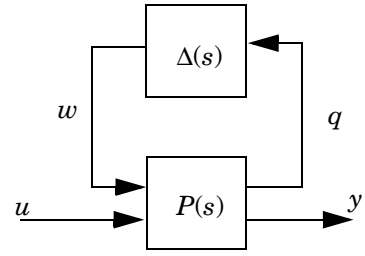


Figure 3-2: Robust stability analysis

This interconnection is stable for all structured $\Delta(s)$ satisfying $\|\Delta\|_\infty < 1$ if and only if [3, 12, 10, 16]

$$\mu_m := \sup_\omega \ \mu_\Delta(P(j\omega)) < 1.$$

The reciprocal $K_m = 1/\mu_m$ represents the robust stability margin, that is, the largest amount of uncertainty Δ that can be tolerated without losing stability.

As mentioned earlier, we can only compute a conservative estimate $\hat{K}_m$ of K_m based on the upper bound ν_Δ of μ_Δ. We refer to $\hat{K}_m$ as the *guaranteed stability margin*. A grid-based estimation of $\hat{K}_m$ is performed by `mustab`:

```
[margin,peakf] = mustab(P,delta,freqs)
```

On input, `delta` describes the uncertainty Δ (see `ublock` and `udiag` for details) and `freqs` is an optional vector of frequencies where to evaluate $\nu_\Delta(P(j\omega))$. On output, `margin` is the computed value of $\hat{K}_m$ and `peakf` is the frequency near which $\nu_\Delta(P(j\omega)$ peaks, i.e., where the margin is the smallest.

The uncertainty quantification may involve different bounds for each block Δ_i. In this case, `margin` represents the *relative* stability margin as a percentage of the specified bounds. More precisely, `margin` = θ means that stability is guaranteed as long as each block Δ_i satisfies

- For norm-bounded blocks

  $\sigma_{max}(\Delta_i(j\omega)) < \theta\ \beta_i(\omega)$

  where $\beta_i(\omega)$ is the (possibly frequency-dependent) norm bound specified with `ublock`
- For sector-bounded blocks

  Δ_i is in the sector $\left\{\frac{a+b}{2} - \theta\frac{b-a}{2}, \frac{a+b}{2} + \theta\frac{b-a}{2}\right\}$

  where $a < b$ are the sector bounds specified with `ublock`. In the special case $b = +\infty$, this condition reads

  Δ_i is in the sector $\left\{a + \frac{1-0}{1+0}, a + \frac{1+0}{1-0}\right\}$

  for θ ð 1 and

  Δ_i is *outside* the sector $\left\{a + \frac{1-0}{1+0}, a + \frac{1+0}{1-0}\right\}$

  for $\theta > 1$

For instance, `margin` = 0.7 for a single-block uncertainty $\Delta(s)$ bounded by 3 means that stability is guaranteed for $\|\Delta\|_\infty < 0.7\text{-by-}3 = 2.1$.

The syntax

```
[margin,peakf,fs,ds,gs] = mustab(P,delta)
```

also returns the *D, G* scaling matrices at the tested frequencies `fs`. To retrieve the values of *D* and *G* at the frequency `fs(i)`, type

```
Di = getdg(ds,i), Gi = getdg(gs,i)
```

Note that `Di` and `Gi` are not necessarily the optimal scalings at `fs(i)`. Indeed, the LMI optimization is often stopped before completion to speed up computations.

Caution: $\hat{K}_m$ is computed by frequency sweep, i.e., by evaluating $\nu_\Delta(P(j\omega))$ over a finite grid of frequencies. The coarser the gridding, the higher the risk of underestimating the peak value of $\nu_\Delta(P(j\omega))$. Worse, this function may be discontinuous at its peaks when the uncertainty contains real blocks. In such cases, any "blind" frequency sweep is likely to miss the peak value and yield an overoptimistic stability margin. The function `mustab` has built-in tests to detect such discontinuities and minimize the risks of missing the critical peaks.

Robust Performance

In robust control, it is customary to formulate the design specifications as abstract disturbance rejection objectives. The performance of a control system is then measured in terms of the closed-loop RMS gain from disturbances to outputs (see "H• Control" on page 5-3 for details). The presence of uncertainty typically deteriorates performance. For an LTI system with linear-fractional uncer- tainty (Figure 3-3), the robust performance γ_{rob} is defined as the worst-case RMS gain from u to y in the face of the uncertainty $\Delta(s)$.

This robust performance is generally worse (larger) than the nominal performance, i.e., the RMS gain from u to y for $(s) \equiv 0$.

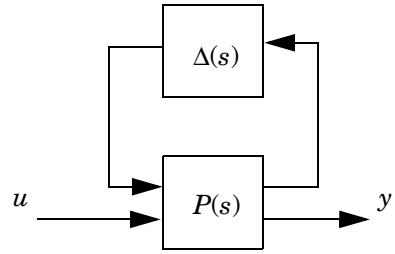


Figure 3-3: Robust performance problem

Assessing whether the performance γ can be robustly sustained, that is, whether $\gamma_{rob} < \gamma$, is equivalent to a robust stability problem. Indeed, $\|y\|_{L_2} < \gamma\|u\|_{L_2}$ holds for all $\Delta(s)$ in the prescribed uncertainty set if and only if the interconnection of Figure 3-4 is stable for all $\Delta(s)$ and for $\Delta_{perf}(s)$ satisfying

$$\|\Delta_{perf}(s)\|_\infty < \gamma^{-1}$$

Given a norm- or sector-bounded linear time-invariant uncertainty $\Delta(s)$, the function `muperf` assesses the robust performance γ_{rob} via this equivalent robust stability problem.

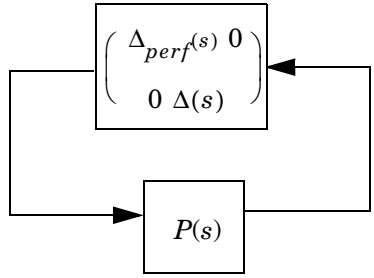


Figure 3-4: Equivalent robust stability problem

Two problems can be solved with `muperf`:

- Compute the robust performance γ_{rob} for the specified uncertainty $\Delta(s)$. This is done by

```
grob = muperf(P,delta)
```

The value `grob` is finite if and only if the interconnection of Figure 3-3 is robustly stable.

- Assess the robustness of a given performance $\gamma = g > 0$ in the face of the uncertainty $\Delta(s)$. The command

  ```
  margin = muperf(P,delta,g)
  ```

  computes a fraction `margin` of the specified uncertainty bounds for which the RMS gain from u to y is guaranteed not to exceed γ.

  Note that `g` should be larger than the nominal RMS gain for $\Delta = 0$. The performance `g` is robust if and only if `margin` Š 1.

The function `muperf` uses the following characterization of γ_{rob} . Assuming $\|\Delta\|_\infty < 1$ and denoting by D and G the related scaling sets,

$$\gamma_{rob} = \max_{\omega} \gamma_{rob}(\omega)$$

where $\gamma_{rob}(\omega)$ is the global minimum of the LMI problem

Minimize γ over $D \in D$, $G \in G$ such that $D > 0$ and

$$P(j\omega)^H \begin{pmatrix} D & 0 \\ 0 & I \end{pmatrix} P(j\omega) + j\left\{ \begin{pmatrix} G & 0 \\ 0 & 0 \end{pmatrix} P(j\omega) - P(j\omega)^H \begin{pmatrix} G & 0 \\ 0 & 0 \end{pmatrix} \right\} < \begin{pmatrix} D & 0 \\ 0 & \gamma^2 I \end{pmatrix}$$

The Popov Criterion

The Popov criterion gives a sufficient condition for the robust stability of the interconnection of Figure 3-5 where $G(s)$ is a given LTI system and $\phi = \text{diag}(\phi_1, \ldots, \phi_n)$ is a sector-bounded BIBO-stable uncertainty satisfying $\phi(0) = 0$. The operator ϕ can be nonlinear and/or time-varying which makes the Popov criterion more general than the μ stability test. For purely linear time-invariant uncertainty, however, the μ test is generally less conservative.

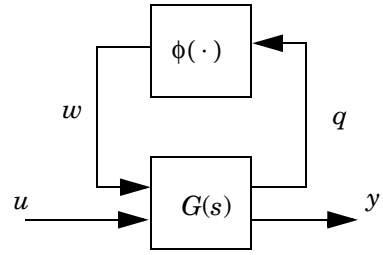


Figure 3-5: Popov criterion

A state-space representation of this interconnection is

$$\begin{cases} \dot{x} = Ax + Bw \\ q = Cx + Dw \\ w = \phi(q) \end{cases}$$

or equivalently in block-partitioned form:

$$\begin{cases} \dot{x} = Ax + \sum_j B_j w_j \\ q_j = C_j x + D_j w \quad (\in \mathbf{R}^{r_j}) \\ w = \phi(q_j) \end{cases}$$

Suppose that $\phi_j(\cdot)$ satisfies the sector bound

$$\forall T > 0, \quad \int_0^T (w_j - \alpha_j q_j)^T (w_j - \beta_j q_j) dt \le 0 \qquad \textbf{(3-13)}$$

(this includes norm bounds as the special case $\beta_j = -\alpha_j$). To establish robust stability, the Popov criterion seeks a Lyapunov function of the form

$$V(x,t) = \frac{1}{2}x^T Px + \sum_j \nu_j \int_0^{q_j} \phi_j(z)dz - \sum_j \sigma_j \int_0^t (w_j - \alpha_j q_j)^T (w_j - \beta_j q_j)d\tau \quad \textbf{(3-14)}$$

where $\sigma_j > 0$ and ν_j are scalars and $\nu_j = 0$ unless $D_j = 0$ and $\phi_j(\cdot)$ is a scalar memoryless nonlinearity (i.e., $w_j(t)$ depends only on $q_j(t)$; in such case, the sector constraint (3-13) reduces to $\alpha_j q_j^2 \le q_j \phi_j(q_j) \le \beta_j q_j^2$).

Letting

$$K\alpha := \mathrm{diag}(\alpha_j I_{r_j}), \qquad K_\beta := \mathrm{diag}(\beta_j I_{r_j})$$
$$S := \mathrm{diag}(\sigma_j I_{r_j}), \qquad N := \mathrm{diag}(\nu_j I_{r_j})$$

and assuming nominal stability, the conditions $V(x, t) > 0$ and $dV/dt < 0$ are equivalent to the following LMI feasibility problem (with the notation $\mathcal{H}(M) := M + M^T$):

Find $P = P^T$ and structured matrices S, N such that $S > 0$ and

$$\mathcal{H}\left\{ \begin{pmatrix} I \\ 0 \end{pmatrix} P(A,B) + \begin{pmatrix} 0 \\ I \end{pmatrix} N(CA,CB) - \begin{pmatrix} C^T K_\alpha \\ D^T K_\alpha - 1 \end{pmatrix} S \begin{pmatrix} C^T K_\beta \\ D^T K_\beta - 1 \end{pmatrix}^T \right\} < 0 \quad \textbf{(3-15)}$$

When $\phi(\cdot)$ is nonlinear time-varying, (3-15) reduces to the scaled small-gain criterion. The Popov stability test is performed by the command

```
[t,P,S,N] = popov(G,phi)
```

where `G` is the `SYSTEM` matrix representation of $G(s)$ and `phi` is the uncertainty description specified with `ublock` and `udiag`. The function `popov` returns the output `t` of the LMI solver `feasp`. Hence the interconnection is robust stable if `t` < 0. In this case, `P`, `S`, and `N` are solutions of the LMI problem (3-15).

Real Parameter Uncertainty

A sharper version of the Popov criterion can be used to analyze systems with uncertain real parameters. Assume that $D_j = 0$ and

$$w_j = \phi_j(q_j) := \delta_j q_j$$

where δ_j is an uncertain real parameter ranging in $[\alpha_j, \beta_j]$. Then a more general form for the corresponding term in the Lyapunov function (3-14) is

$$\int_0^{q_j} \phi(z)^T N_j dz - \int_0^t (w_j - \alpha_j q_j)^T S_j (w_j - \beta_j q_j) d\tau = \delta_j x^T C^T N_j C x - (\delta_j - \alpha_j)(\delta_j - \beta_j) \int_0^t q_j^T (S_j + S_j^T) q_j d\tau \tag{3-16}$$

where S_j and N_j are $r_j \times r_j$ matrices subject to $S_j + S_j^T > O, N = N_j^T$, and $N_j = 0$ when the parameter δ_j is time varying. As a result, the variables S and N in the Popov condition (3-15) assume a block-diagonal structure where $\sigma_j I_{r_j}$ and $\nu_j I_{r_j}$ are replaced by the full blocks S_j and N_j, respectively.

When all $\phi_j(\cdot)$ are real time-invariant parameters, the Lyapunov function (3-14) becomes

$$V(x,t) = \frac{1}{2} x^T \left(P + \sum_j \delta_j C^T N_j C \right) x - \sum_j (\delta_j - \alpha_j)(\delta_j - \beta_j) \int_0^t q_j^T (S_j + S_j^T) q_j d\tau$$

The first term can be interpreted as a parameter-dependent quadratic function of the state. This is analogous to the notion of parameter-dependent Lyapunov function discussed in “Parameter-Dependent Lyapunov Functions” on page 3-12, even though the Popov condition (3-15) is not equivalent to the robust stability conditions discussed in that section..

Finally, it is possible to obtain an even sharper test by applying the Popov criterion to the equivalent interconnection

$$\begin{cases} \dot{x} = (Ax + \sum_j B_j C_j \tilde{w}_j) \\ \tilde{q}_j = x \\ \tilde{w}_j = \delta_j \tilde{q}_j \end{cases} \tag{3-17}$$

The Lyapunov function now becomes

$$V(x,t) = \frac{1}{2} x^T \left(P + \sum_j \delta_j N_j \right) x - \sum_j (\delta_j - \alpha_j)(\delta_j - \beta_j) \int_0^t q_j^T (S_j + S_j^T) q_j d\tau$$

with S_j, N_j subject to $S_j + S_j^T > O$ and $N_j = N_j^T$. The resulting test is discussed in [5, 6] and often performs as well as the real stability test while eliminating the hazards of a frequency sweep (see “Caution” on page 3-21).

To use this refined test, invoke `popov` as follows.

```
[t,P,S,N] = popov(G,phi,1)
```

The third input `1` signals `popov` to perform the loop transformation (3-17) before solving the feasibility problem (3-15).

Example

The use of these various robustness tests is illustrated on the benchmark problem proposed in [14]. The physical system is sketched in Figure 3-6.

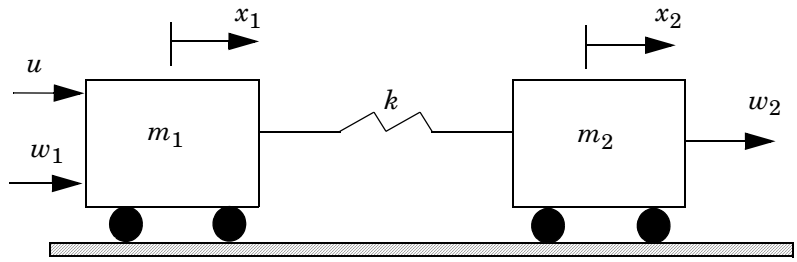


Figure 3-6: Benchmark problem

The goal is to design an output-feedback law $u = K(s)x_2$ that adequately rejects the disturbances w_1, w_2 and guarantees stability for values of the stiffness parameter k ranging in [0.5, 2.0]. For $m_1 = m_2 = 1$, a state-space description of this system is

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ \ddot{x}_1 \\ \ddot{x}_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -k & k & 0 & 0 \\ k & -k & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} (u + w_1) + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} w_2 \qquad \textbf{(3-18)}$$

A solution of this problem proposed in [15] is the fourth-order controller $u = C_K(sI - A_K)^{-1} B_K x_2$ where

$$A_K = \begin{bmatrix} 0 & -0.7195 & 1 & 0 \\ 0 & -2.9732 & 0 & 1 \\ -2.5133 & 4.8548 & -1.7287 & -0.9616 \\ 1.0063 & -5.4097 & -0.0081 & 0.0304 \end{bmatrix}, \quad B_K = \begin{bmatrix} 0.720 \\ 2.973 \\ -3.37 \\ 4.419 \end{bmatrix}$$

$$C_K = \begin{bmatrix} -1.506 & 0.494 & -1.738 & -0.932 \end{bmatrix}$$

The concern here is to assess, for this particular controller, the closed-loop stability margin with respect to the uncertain real parameter k. Since the plant equations (3-18) depend affinely on k, we can define the uncertain physical system as an affine parameter-dependent system `G` with `psys`.

```
A0=[0 0 1 0;0 0 0 1;0 0 0 0;0 0 0 0];
B0=[0;0;1;0]; C0=[0 1 0 0]; D0=0;
S0=ltisys(A0,B0,C0,D0); % system for k=0

A1=[0 0 0 0;0 0 0 0; 1 1 0 0;1  1 0 0];
B1=zeros(4,1); C1=zeros(1,4); D1=0;
S1=ltisys(A1,B1,C1,D1,0); % k component

pv=pvec('box',[0.5 2]) % range of values of k
G=psys(pv,[S0,S1])
```

After entering the controller data as a `SYSTEM` matrix `K`, close the loop with `slft`.

```
Cl = slft(G,K)
```

The result `Cl` is a parameter-dependent description of the closed-loop system.

To estimate the robust stability margin with respect to k, we can now apply to `Cl` the various tests described in this chapter:

- **Quadratic stability:** to determine the portion of the interval [0.5, 2] where quadratic stability holds, type

  ```
  marg = quadstab(Cl,[1 0 0])

  marg =

     4.1919e 01
  ```

  The value `marg` = 0.419 means 41% of [0.5, 2] (with respect to the center 1.25). That is, quadratic stability in the interval [0.943, 1.557]. Since quadratic stability assumes arbitrarily fast time variations of the parameter k, we can expect this answer to be conservative when k is time invariant.
- **Parameter-dependent Lyapunov functions:** when k does not vary in time, a less conservative estimate is provided by `pdlstab`. To test stability for $k \in [0.5, 2]$, type

```
t = pdlstab(Cl)

t =

    2.1721e 01
```

Since `t` < 0, the closed loop is robustly stable for this range of values of k.

Assume now that k slowly varies with a rate of variation bounded by 0.1. To test if the closed loop remains stable in the face of such slow variations, redefine the parameter vector and update the description of the closed-loop system by

```
pv1 = pvec('box',[0.5 2],[ 0.1 0.1])
G1 = psys(pv1,[S0,S1])
Cl1 = slft(G1,K)
```

Then call `pdlstab` as earlier.

```
t = pdlstab(Cl1)

t =

    2.0089e 02
```

Since t is again negative, this level of time variations does not destroy robust stability.

- **μ analysis**: to perform μ analysis, first convert the affine parameter-dependent model `Cl` to an equivalent linear-fractional uncertainty model.

```
[P0,deltak] = aff2lft(Cl)
uinfo(deltak)
```

block	dims	type	real/cplx	full/scal	bounds
1	1x1	LTI	r	s	norm <= 0.75

Here `P0` is the closed loop system for the nominal value $k_0 = 1.25$ of k and the uncertainty on k is represented as a real scalar block `deltak`. Note that k must be assumed *time invariant* in the μ framework.

To get the relative parameter margin, type

```
[pmarg,peakf] = mustab(P0,deltak)

pmarg =

   1.0670e+00

peakf =

   6.3959e 01
```

The value `pmarg` = 1.064 means stability as long as the deviation $|\delta k|$ from $k_0 = 1.25$ does not exceed $0.75 \times 1.067 \approx 0.80$. That is, as long as k remains in the interval [0.45, 2.05]. This estimate is sharp since the closed loop becomes unstable for $\delta k = -0.81$, that is, for $k = k_0 - 0.81 = 0.44$:

```
spol(slft(P0, 0.81))

ans =

 1.0160e+00 + 1.9208e+00i
 1.0160e+00   1.9208e+00i
 1.0512e+00 + 9.7820e 01i
 1.0512e+00   9.7820e 01i
 6.2324e 03 + 6.3884e 01i
 6.2324e 03   6.3884e 01i
 2.7484e 01 + 1.9756e 01i
 2.7484e 01   1.9756e 01i
```

- **Popov criterion:** finally, we can apply the Popov criterion to the linear-fractional model returned by `aff2lft`

```
[t,P,D,N] = popov(P0,deltak)

t =

    1.3767e 02
```

This test is also successful since `t` < 0. Note that the Popov criterion also proves stability for $k = k_0 + \delta k(\cdot)$ where $\delta k(\cdot)$ is any memoryless nonlinearity with gain less than 0.75.

References

[1] Barmish, B.R., "Stabilization of Uncertain Systems via Linear Control," *IEEE Trans. Aut. Contr.*, AC–28 (1983), pp. 848–850.

[2] Boyd, S., and Q. Yang, "Structured and Simultaneous Lyapunov Functions for System Stability Problems," *Int. J. Contr.*, 49 (1989), pp. 2215–2240.

[3] Doyle, J.C., "Analysis of Feedback Systems with Structured Uncertainties," *IEE Proc.*, vol. 129, pt. D (1982), pp. 242–250.

[4] Fan, M.K.H., A.L. Tits, and J.C. Doyle,"Robustness in the Presence of Mixed Parametric Uncertainty and Unmodeled Dynamics," *IEEE Trans. Aut. Contr.*, 36 (1991), pp. 25–38.

[5] Feron, E, P. Apkarian, and P. Gahinet, "S-Procedure for the Analysis of Control Systems with Parametric Uncertainties via Parameter-Dependent Lyapunov Functions," *Third SIAM Conf. on Contr. and its Applic.*, St. Louis, Missouri, 1995.

[6] Gahinet, P., P. Apkarian, and M. Chilali, "Affine Parameter-Dependent Lyapunov Functions for Real Parametric Uncertainty," *Proc. Conf. Dec. Contr.*, 1994, pp. 2026–2031.

[7] Haddad, W.M. and D.S. Berstein,"Parameter-Dependent Lyapunov Functions, Constant Real Parameter Uncertainty, and the Popov Criterion in Robust Analysis and Synthesis: Part 1 and 2," *Proc. Conf. Dec. Contr.*, 1991, pp. 2274–2279 and 2632–2633.

[8] Horisberger, H.P., and P.R. Belanger, "Regulators for Linear Time-Varying Plants with Uncertain Parameters," *IEEE Trans. Aut. Contr.*, AC–21 (1976), pp. 705–708.

[9] How, J.P., and S.R. Hall, "Connection between the Popov Stability Criterion and Bounds for Real Parameter Uncertainty," *Proc. Amer. Contr. Conf.*, 1993, pp. 1084–1089.

[10] Packard, A., and J.C. Doyle, "The Complex Structured Singular Value," *Automatica*, 29 (1994), pp. 71–109.

[11] Popov, V.M., "Absolute Stability of Nonlinear Systems of Automatic Control," *Automation and Remote Control*, 22 (1962), pp. 857–875.

[12] Safonov, M.G., "L1 Optimal Sensitivity vs. Stability Margin," *Proc. Conf. Dec. Contr.*, 1983.

[13] Stein, G. and J.C. Doyle, "Beyond Singular Values and Loop Shapes," *J. Guidance*, 14 (1991), pp. 5–16.

[14] Wie, B., and D.S. Berstein, "Benchmark Problem for Robust Control Design," *J. Guidance and Contr.*, 15 (1992), pp. 1057–1059.

[15] Wie, B., Q. Liu, and K.-W. Byun, "Robust H_∞ Control Synthesis Method and Its Application to Benchmark Problems," *J. Guidance and Contr.*, 15 (1992), pp 1140–1148.

[16] Young, P. *M.*, M. P. Newlin, and J. C. Doyle, "Let's Get Real," in *Robust Control Theory*, Springer Verlag, 1994, pp. 143–174.

[17] Zames, G., "On the Input-Output Stability of Time-Varying Nonlinear Feedback Systems, Part I and II," *IEEE Trans. Aut. Contr.*, AC–11 (1966), pp. 228–238 and 465–476.

4

State-Feedback Synthesis

Multi-Objective State-Feedback 4-3

Pole Placement in LMI Regions 4-5

LMI Formulation 4-7
Extension to the Multi-Model Case 4-9

The Function msfsyn 4-11

Design Example 4-13

References 4-18

In many control problems, the design specifications are a mix of performance and robustness objectives expressed both in the time and frequency domains. These various objectives are rarely encompassed by a single synthesis criterion. While some tracking and robustness are best captured by an H_∞ criterion, noise insensitivity is more naturally expressed in LQG terms, and transient behaviors are more easily tuned in terms of l_1 norm or closed-loop damping.

The LMI framework is particularly well suited to multi-objective state-feedback synthesis. As an illustration, the LMI Control Toolbox offers tools for state-feedback design with a combination of the following objectives:

- H_∞ performance (for tracking, disturbance rejection, or robustness aspects)
- H_2 performance (for LQG aspects)
- Robust pole placement specifications (to ensure fast and well-damped transient responses, reasonable feedback gain, etc.)

These tools apply to *multi-model problems*, i.e., when the objectives are to be robustly achieved over a polytopic set of plant models.

Multi-Objective State-Feedback

The function `msfsyn` performs multi-model H_2/H_∞ state-feedback synthesis with pole placement constraints. For simplicity, we describe the underlying problem in the case of a *single* LTI model. The control structure is depicted by Figure 4-1. The plant $P(s)$ is a given LTI system and we assume full measurement of its state vector x.

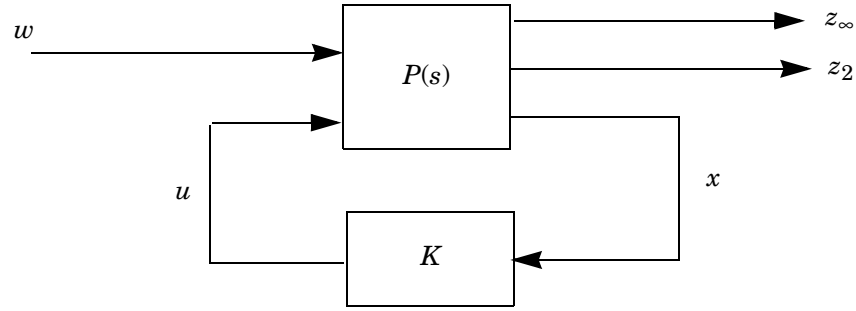


Figure 4-1: State-feedback control

Denoting by $T_\infty(s)$ and $T_2(s)$ the closed-loop transfer functions from w to z_∞ and z_2, respectively, our goal is to design a state-feedback law $u = Kx$ that

- Maintains the RMS gain (H_∞ norm) of T_∞ below some prescribed value $\gamma_0 > 0$
- Maintains the H_2 norm of T_2 (LQG cost) below some prescribed value $\nu_0 > 0$
- Minimizes an H_2/H_∞ trade-off criterion of the form

  $$\alpha\|T_\infty\|_\infty^2 + \beta\|T_2\|_2^2$$

- Places the closed-loop poles in a prescribed region D of the open left-half plane.

This abstract formulation encompasses many practical situations. For instance, consider a regulation problem with disturbance d and white measurement noise n, and let e denote the regulation error. Setting

$$w = \begin{pmatrix} d \\ n \end{pmatrix}, \quad z_\infty = e, \quad z_2 = \begin{pmatrix} x \\ u \end{pmatrix},$$

the mixed H_2/H_∞ criterion takes into account both the disturbance rejection aspects (RMS gain from d to e) and the LQG aspects (H_2 norm from n to z_2). In addition, the closed-loop poles can be forced into some sector of the stable half-plane to obtain well-damped transient responses.

Pole Placement in LMI Regions

The concept of LMI region [3] is useful to formulate pole placement objectives in LMI terms. LMI regions are convex subsets D of the complex plane characterized by

$$\mathcal{D} = \{z \in \mathbf{C} : L + Mz + M^T\bar{z} < 0\}$$

where M and $L = L^T$ are fixed real matrices. The matrix-valued function

$$f_{\mathcal{D}}(z) := L + Mz + M^T\bar{z}$$

is called the characteristic function of the region $\mathcal{D}$. The class of LMI regions is fairly general since its closure is the set of convex regions symmetric with respect to the real axis. More practically, LMI regions include relevant regions such as sectors, disks, conics, strips, etc., as well as any intersection of the above.

Another strength of LMI regions is the availability of a "Lyapunov theorem" for such regions. Specifically, if $\{\lambda_{ij}\}_{1 \le i,j \le m}$ and $\{\mu_{ij}\}_{1 \le i,j \le m}$ denote the entries of the matrices L and M, a matrix A has all its eigenvalues in $\mathcal{D}$ if and only if there exists a positive definite matrix P such that [3]

$$[\lambda_{ij}P + \mu_{ij}AP + \mu_{ji}PA^T]_{1 \le i,j \le m} < 0$$

with the notation

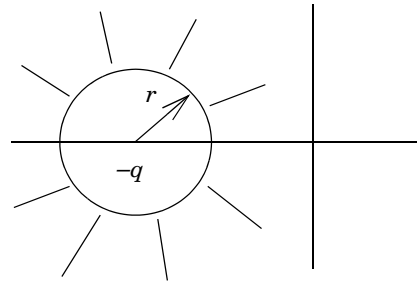
$$[S_{ij}]_{1 \le i,j \le m} := \begin{pmatrix} S_{11} & \cdots & S_{i1} \\ \vdots & \ddots & \vdots \\ S_{m1} & \cdots & S_{mm} \end{pmatrix}$$

Note that this condition is an LMI in P and that the classical Lyapunov theorem corresponds to the special case

$$f_D(z) = z + \bar{z}$$

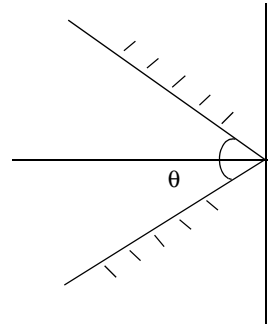
Next we list a few examples of useful LMI regions along with their characteristic function f_D:

- Disk with center at $(-q, 0)$ and radius r:



$$f_D(z) = \begin{pmatrix} -r & \bar{z}+q \\ z+q & -r \end{pmatrix}$$

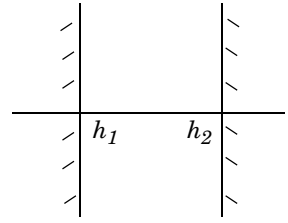
- Conic sector centered at the origin and with inner angle θ:



$$f_D(z) = \begin{pmatrix} \sin\frac{\theta}{2}(z+\bar{z}) & -\cos\frac{\theta}{2}(z-\bar{z}) \\ \cos\frac{\theta}{2}(z-\bar{z}) & \sin\frac{\theta}{2}(z-\bar{z}) \end{pmatrix}$$

The damping ratio of poles lying in this sector is at least $\cos\frac{\theta}{2}$.

- Vertical strip $h_1 < x < h_2$:



$$f_D(z) = \begin{pmatrix} 2h_1 - (z+\bar{z}) & 0 \\ 0 & (z+\bar{z}) - 2h_2 \end{pmatrix}$$

LMI Formulation

Given a state-space realization

$$\begin{cases} \dot{x} &= Ax + B_1 w + B_2 u \\ z_\infty &= C_1 x + D_{11} w + D_{12} u \\ z_2 &= C_2 x + D_{22} u \end{cases}$$

of the plant P, the closed-loop system is given in state-space form by

$$\begin{cases} \dot{x} &= (A + B_2 K)x + B_1 w \\ z_\infty &= (C_1 + D_{12} K)x + D_{11} w \\ z_2 &= (C_2 + D_{22} K)x \end{cases}$$

Taken separately, our three design objectives have the following LMI formulation:

- H_∞ **performance:** the closed-loop RMS gain from w to z_∞ does not exceed γ if and only if there exists a symmetric matrix X_∞ such that [5]

$$\begin{pmatrix} (A + B_2 K)X_\infty + X_\infty (A + B_2 K)^T & B_1 & X_\infty (C_1 + D_{12} K)^T \\ B_1^T & -I & D_{11}^T \\ (C_1 + D_{12} K)X_\infty & D_{11} & -\gamma^2 I \end{pmatrix} < 0$$

$$X_\infty > 0$$

- H_2 **performance:** the closed-loop H_2 norm of T_2 does not exceed ν if there exist two symmetric matrices X_2 and Q such that

$$\begin{pmatrix} (A+B_2K)X_2 + X_2(A+B_2K)^T & B_1 \\ B_1^T & -I \end{pmatrix} < 0$$

$$\begin{pmatrix} Q & (C_2+D_{22}K)X_2 \\ X_2(C_2+D_{22}K)^T & X_2 \end{pmatrix} > 0$$

$$\mathrm{Trace}(Q) < \nu^2$$

(see "LQG performance" on page 1-7 for details)

- **Pole placement:** the closed-loop poles lie in the LMI region

  $D = \{z \in \mathbf{C} : L + Mz + M^T\bar{z} < 0\}$

  where

  $L = L^T = \{\lambda_{ij}\}_{1 \le i,j \le m} \qquad M = \{\mu_{ij}\}_{1 \le i,j \le m}$

  if and only if there exists a symmetric matrix X_{pol} satisfying

  $$[\lambda_{ij}X_{pol} + \mu_{ij}(A+B_2K)X_{pol} + \mu_{ij}X_{pol} + \mu_{ji}X_{pol}(A+B_2K)^T]_{1 \le i,j \le m} < 0$$

  $$X_{pol} > 0$$

These three sets of conditions add up to a nonconvex optimization problem with variables Q, K, X_∞, X_2 and X_{pol}. For tractability in the LMI framework, we seek a single Lyapunov matrix

$$X := X_\infty = X_2 = X_{pol} \tag{4-1}$$

that enforces all three objectives. With the change of variable $Y := KX$, this leads to the following suboptimal LMI formulation of our multi-objective state-feedback synthesis problem [4, 3, 2]:

Minimize $\alpha\,\gamma^2 + \beta\,\mathrm{Trace}(Q)$ over Y, X, Q, and γ^2 satisfying

$$\begin{pmatrix} AX + XA^T + B_2Y + Y^TB_2^T & B_1 & XC_1^T + Y^TD_{12}^T \\ B_1^T & -I & D_{11}^T \\ C_1X + D_{12}Y & D_{11} & -\gamma^2 I \end{pmatrix} < 0 \tag{4-2}$$

$$\begin{pmatrix} Q & C_2X + D_{22}Y \\ XC_2^T + Y^TD_{22}^T & X \end{pmatrix} > 0 \tag{4-3}$$

$$[\lambda_{ij} + \mu_{ij}(AX + B_2Y)X_{pol} + \mu_{ji}(XA^T + Y^TB_2^T)]_{1 \le i,j \le m} < 0 \tag{4-4}$$

$$\mathrm{Trace}(Q) < \nu_0^2 \tag{4-5}$$

$$\gamma^2 < \gamma_0^2 \tag{4-6}$$

Denoting the optimal solution by (X^*, Y^*, Q^*, γ^*), the corresponding state-feedback gain is given by

$$K^* = Y^*(X^*)^{-1}$$

and this gain guarantees the worst-case performances:

$$\|T_\infty\|_\infty \le \gamma^*, \qquad \|T_2\|_2 \le \sqrt{\mathrm{Trace}(Q^*)}$$

Note that K^* does not yield the globally optimal trade-off in general due to the conservatism of Assumption (4-1).

Extension to the Multi-Model Case

The LMI approach outlined above extends to uncertain linear time-varying plants

$$P:\begin{cases}\dot{x} &= A(t)x + B_1(t)w + B_2(t)u \\ z_\infty &= C_1(t)x + D_{11}(t)w + D_{12}(t)u \\ z_2 &= C_2(t)x + D_{22}(t)u\end{cases}$$

with state-space matrices varying in a polytope:

$$\begin{pmatrix} A(t) & B_1(t) & B_2(t) \\ C_1(t) & D_{11}(t) & D_{12}(t) \\ C_2(t) & 0 & D_{22}(t) \end{pmatrix} \in \mathbf{Co}\left\{ \begin{pmatrix} A_k & B_{1k} & B_{2k} \\ C_{1k} & D_{11k} & D_{12k} \\ C_{2k} & 0 & D_{22k} \end{pmatrix} : k = 1, \ldots, K \right\}$$

Such polytopic models are useful to represent plants with uncertain and/or time-varying parameters (see "Polytopic Models" on page 2-14 and the design example below). Seeking a single quadratic Lyapunov function that enforces the design objectives for all plants in the polytope leads to the following multi-model counterpart of the LMI conditions (4-2)–(4-6):

Minimize $\alpha\,\gamma^2 + \beta\,\mathrm{Trace}(Q)$ over Y, X, Q, and γ^2 subject to

$$\begin{pmatrix} A_kX + XA_k^T + B_{2k}Y + Y^TB_{2k}^T & B_{1k} & XC_{1k}^T + Y^TD_{12k}^T \\ B_{1k}^T & -I & D_{11k}^T \\ C_{1k}X + D_{12k}Y & D_{11k} & -\gamma^2 I \end{pmatrix} < 0$$

$$\begin{pmatrix} Q & C_{2k}X + D_{22k}Y \\ XC_{2k}^T + Y^TD_{22k}^T & X \end{pmatrix} > 0$$

$$[\lambda_{ij} + \mu_{ij}(A_kX + B_{2k}Y) + \mu_{ji}(XA_k^T + Y^TB_{2k}^T)]_{1 \le i,j \le m} < 0$$

$$\mathrm{Trace}(Q) < \nu_0^2$$

$$\gamma^2 < \gamma_0^2$$

The Function msfsyn

The function `msfsyn` implements the LMI approach to multi-model H_2/H_∞ synthesis outlined above. The pole placement objectives are expressed in terms of LMI regions

$$\mathcal{D} = \{z \in \mathbf{C} : L + Mz + M\bar{z} < 0\}$$

characterized by the two matrices L and M. LMI regions are specified interactively with the function `lmireg`.

Denoting the closed-loop transfer functions from w to z_∞ and z_2 by T_∞ and T_2, `msfsyn` computes a suboptimal solution to the mixed problem:

$$\text{Minimize } \alpha\|T_\infty\|_0^2 + \beta\|T_2\|_2^2 \text{ subject to}$$

- $\|T_\infty\|_\infty < \gamma_0$
- $\|T_2\|_2 < \nu 0$
- The closed-loop poles lie in $\mathcal{D}$.

The syntax is

```
[gopt,h2opt,K,Pcl] = msfsyn(P,r,obj,region)
```

where

- `P` is the plant `SYSTEM` matrix in the single-model case, or the polytopic/parameter-dependent description of the plant in the multi-model case (see `psys` for details). The dimensions of the D_{22} matrix are specified by `r`
- `obj` = $[\gamma_0, \nu_0, \alpha, \beta]$ is a four-entry vector specifying the H_2/H_∞ criterion
- `region` specifies the LMI region to be used for pole placement, the default being the open left-half plane. Use `lmireg` to generate the matrix region, or set it to $[L, M]$ if the characteristic matrices L and M are readily available.

On output, `gopt` and `h2opt` are the guaranteed H_∞ and H_2 performances, `K` is the state-feedback gain, and `Pcl` is the closed-loop system in `SYSTEM` matrix or polytopic model format.

Several mixed or unmixed designs can be performed with `msfsyn`. The various possibilities are summarized in the table below.

`obj`	**Corresponding Design**
`[0 0 0 0]`	pole placement only
`[0 0 1 0]`	H_∞-optimal design
`[0 0 0 1]`	H_2-optimal design
`[g 0 0 1]`	minimize $\|T_2\|_2$ subject to $\|T_\infty\|_\infty <$ `g`
`[0 h 1 0]`	minimize $\|T_\infty\|_\infty$ subject to $\|T_2\|_2 <$ `h`
`[0 0 a b]`	`minimize` $a\left\|T_\infty\right\|_\infty^2 + b\left\|T_2\right\|_2^2$

Design Example

This example is adapted from [1] and covered by the demo `sateldem`. The system is a satellite consisting of two rigid bodies (main body and instrumentation module) joined by a flexible link (the "boom"). The boom is modeled as a spring with torque constant k and viscous damping f and finite-element analysis gives the following uncertainty ranges for k and f :

$$0.09 \eth \quad k \; 0.4$$
$$0.0038 \eth \quad f \; 0.04:$$

The dynamical equations are

$$\begin{cases} J_1\ddot{\theta}_1 + f(\dot{\theta}_1 - \dot{\theta}_2) + k(\theta_1 - \theta_2) = T + w \\ J_2\ddot{\theta}_2 + f(\dot{\theta}_2 - \dot{\theta}_1) + k(\theta_2 - \theta_1) = 0 \end{cases}$$

where θ_1 and θ_2 are the yaw angles for the main body and the sensor module, T is the control torque, and w is a torque disturbance on the main body.

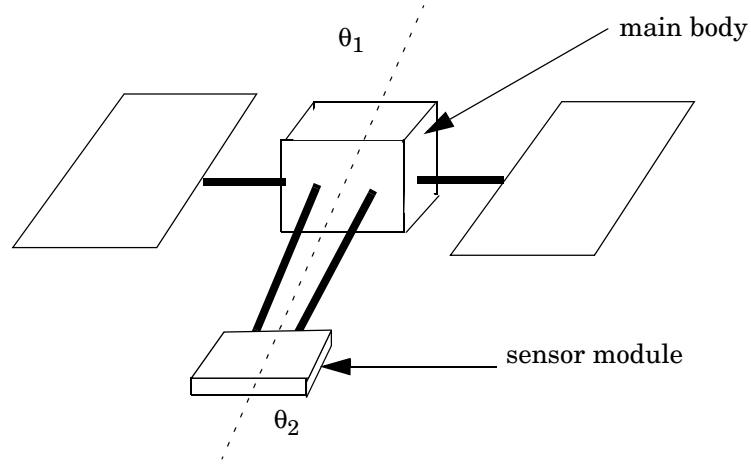


Figure 4-1: Satellite

The control purpose is to minimize the influence of the disturbance w on the angular position θ_2. This goal is expressed through the following objectives:

- Obtain a good trade-off between the RMS gain from w to θ_2 and the H_2 norm of the transfer function from w to

$$\begin{pmatrix} \theta_1 \\ \theta_2 \\ T \end{pmatrix}$$

(LQG cost of control)

- Place the closed-loop poles in the region shown in Figure 4-3 to guarantee some minimum decay rate and closed-loop damping

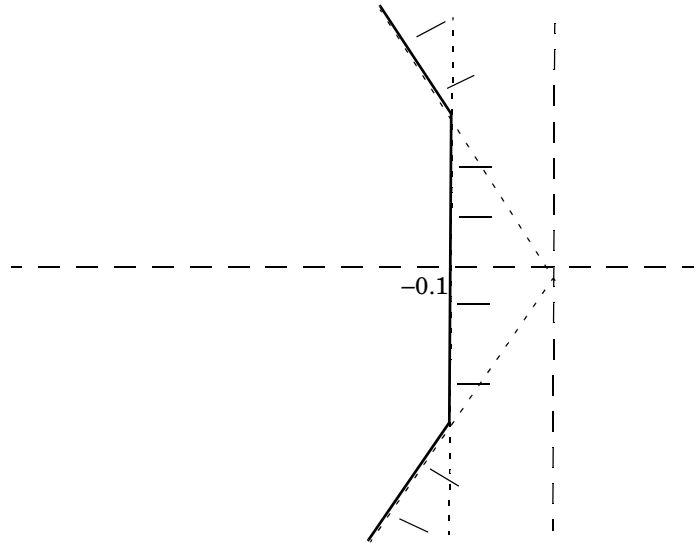


Figure 4-2: Pole placement region

- Achieve these objectives for all possible values of the varying parameters k and f. Since these parameters enter the plant state matrix in an affine manner, we can model the parameter uncertainty by a polytopic system with four vertices corresponding to the four combinations of extremal parameter values (see "From Affine to Polytopic Models" on page 2-20).

To solve this design problem with the LMI Control Toolbox, first specify the plant as a parameter-dependent system with affine dependence on k and f. A state-space description is readily derived from the dynamical equations as:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & J_1 & 0 \\ 0 & 0 & 0 & J_2 \end{bmatrix} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \ddot{\theta}_1 \\ \ddot{\theta}_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -k & k & -f & f \\ k & -k & f & -f \end{bmatrix} + \begin{bmatrix} \theta_1 \\ \theta_2 \\ \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} (w + T)$$

$$z_\infty = \theta_2, \qquad z_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \theta_1 \\ \theta_2 \\ \dot{\theta}_1 \\ \dot{\theta}_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} T$$

This parameter-dependent model is entered by the commands

```
a0 = [zeros(2) eye(2); zeros(2,4)]
ak = [zeros(2,4) ; [-1 1;1 -1] zeros(2)]
af = [zeros(2,4) ; zeros(2) [-1 1;1 -1]]
e0 = diag([1 1 J1 J2])

b = [0 0;0 0;1 1;0 0] % b = [b1 b2]
c = [0 1 0 0;1 0 0 0;0 1 0 0;0 0 0 0] % c = [c1;c2]
d = [0 0;0 0;0 0;0 1]

% range of parameter values
pv = pvec('box',[0.09 0.4 ; 0.0038 0.04])

% parameter-dependent plant
P = psys(pv,[ ltisys(a0,b,c,d,e0) , ...

   ltisys(ak,0*b,0*c,0*d,0) , ...
   ltisys(af,0*b,0*c,0*d,0) ])
```

Next, specify the LMI region for pole placement as the intersection of the half-plane $x < -0.1$ and of the sector centered at the origin and with inner angle $3\pi/4$. This is done interactively with the function `lmireg`:

```
region = lmireg
```

To assess the trade-off between the H_∞ and H_2 performances, first compute the optimal quadratic H_∞ performance subject to the pole placement constraint by

```
gopt = msfsyn(P,[1 1],[0 0 1 0],region)
```

This yields `gopt` ≈ 0. For a prescribed H_∞ performance `g` > 0, the best H_2 performance `h2opt` is computed by

```
[gopt,h2opt,K,Pcl] = msfsyn(P,[1 1],[g 0 0 1],region)
```

Here `obj = [g 0 0 1]` asks to optimize the H_2 performance subject to $\|T_\infty\|_\infty$ < `g` and the pole placement constraint. Repeating this operation for the values `g` ∈ {0.01, 0.1, 0.2, 0.5} yields the Pareto-like trade-off curve shown in Figure 4-4.

By inspection of this curve, the state-feedback gain `K` obtained for `g = 0.1` yields the best compromise between the H_∞ and H_2 objectives. For this choice of `K`, Figure 4-5 superimposes the impulse responses from w to θ_2 for the four combinations of extremal values of k and f.

Finally, the closed-loop poles for these four extremal combinations are displayed in Figure 4-6. Note that they are robustly placed in the prescribed LMI region.

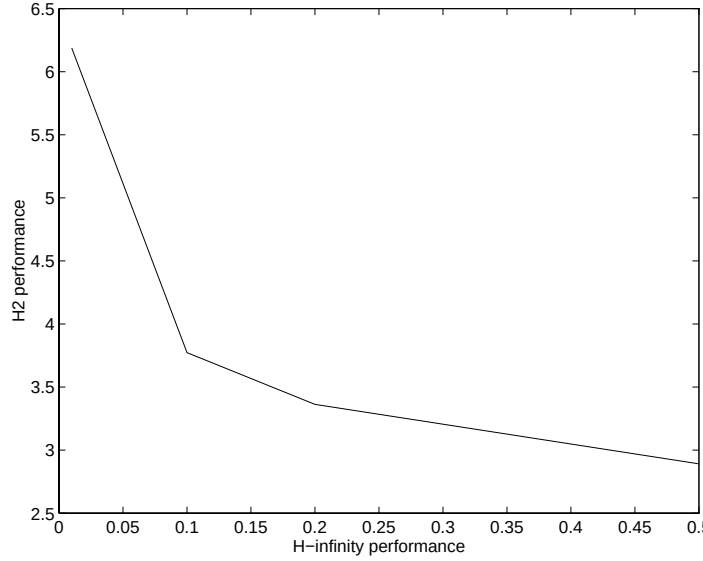


Figure 4-3: Trade-off between the H_∞ and H_2 performances

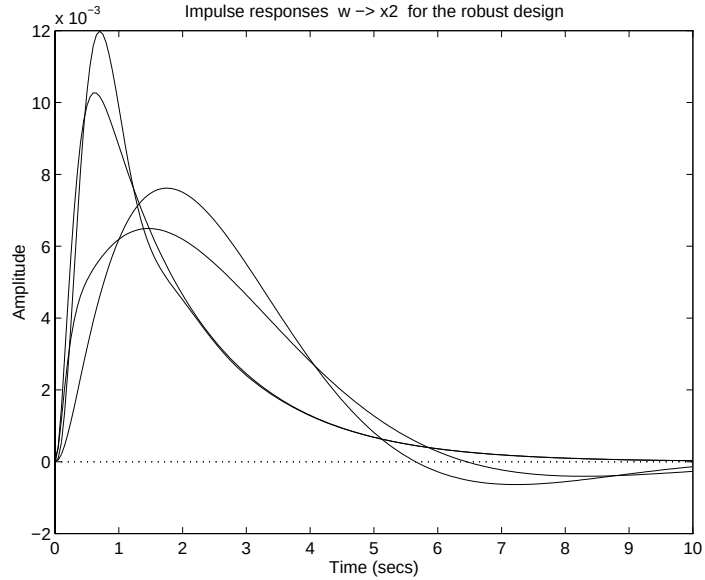


Figure 4-4: Impulse responses for the extremal values of k and f

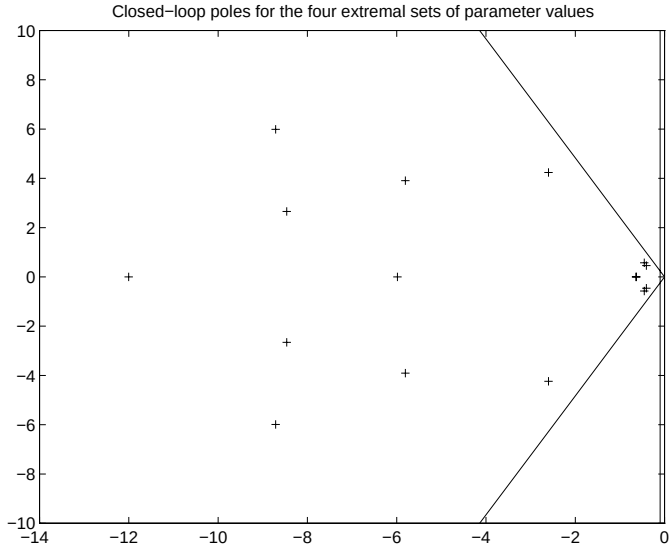


Figure 4-5: Corresponding closed-loop poles

References

[1] Biernacki, R.M., H. Hwang, and S.P. Battacharyya, "Robust Stability with Structured Real Parameter Perturbations," *IEEE Trans. Aut. Contr.*, AC–32 (1987), pp. 495–506.

[2] Boyd, S., L. El Ghaoui, E. Feron, V. Balakrishnan, *Linear Matrix Inequalities in Systems and Control Theory*, SIAM books, 1994.

[3] Chilali, M., and P. Gahinet, "H_∞ Design with Pole Placement Constraints: an LMI Approach," to appear in *IEEE Trans. Aut. Contr.* Also in *Proc. Conf. Dec. Contr.*, 1994, pp. 553–558.

[4] Khargonekar, P.P., and M.A. Rotea,"Mixed H_2/H_∞ Control: A Convex Optimization Approach," *IEEE Trans. Aut. Contr.*, 39 (1991), pp. 824-837.

[5] Scherer, C., "H_∞ Optimization without Assumptions on Finite or Infinite Zeros," *SIAM J. Contr. Opt.*, 30 (1992), pp. 143–166.

5

Synthesis of H_∞ Controllers

$H\infty$ Control 5-3
Riccati- and LMI-Based Approaches 5-7

$H\infty$ Synthesis 5-10
Validation of the Closed-Loop System 5-13

Multi-Objective $H\infty$ Synthesis 5-15
LMI Formulation 5-16
The Function hinfmix 5-20
Loop-Shaping Design with hinfmix 5-20

References 5-22

Disturbance rejection, input/output decoupling, and general robust stability or performance objectives can all be formulated as gain attenuation problems in the face of dynamical or parametric uncertainty. H_∞ control is the cornerstone of design techniques for this class of problems.

The LMI Control Toolbox addresses three variants of output-feedback H_∞ synthesis:

- Standard H_∞ control for continuous-time systems. Both the Riccati-based and LMI-based solutions of this problem are implemented
- Multi-objective H_∞ design (mixed H_2/H_∞ synthesis with closed-loop pole placement in LMI regions)
- Discrete-time H_∞ control via either Riccati- or LMI-based approaches

Additional facilities for loop-shaping design are described in Chapter 6, "Loop Shaping."

H_∞ Control

The H_∞ norm of a stable transfer function $G(s)$ is its largest input/output RMS gain, i.e.,

$$\|G\|_\infty = \sup_{\substack{u \in L_2 \\ u \neq 0}} \frac{\|y\|_{L_2}}{\|u\|_{L_2}}$$

where L_2 is the space of signals with finite energy and $y(t)$ is the output of the system G for a given input $u(t)$. This norm also corresponds to the peak gain of the frequency response $G(j\omega)$, that is,

$$\|G\|_\infty = \sup_{\omega} \sigma_{max}(G(j\omega))$$

In its abstract "standard" formulation, the H_∞ control problem is one of disturbance rejection. Specifically, it consists of minimizing the closed-loop RMS gain from w to z in the control loop of Figure 5-1. This can be interpreted as minimizing the effect of the worst-case disturbance w on the output z.

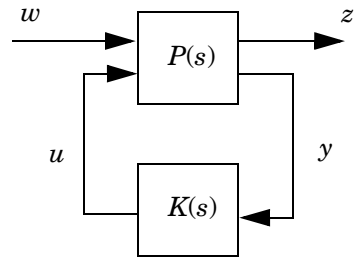


Figure 5-1: H_∞ control

Partitioning the plant $P(s)$ as

$$\begin{pmatrix} Z(s) \\ Y(s) \end{pmatrix} = \begin{pmatrix} P_{11}(s) & P_{12}(s) \\ P_{21}(s) & P_{22}(s) \end{pmatrix} \begin{pmatrix} W(s) \\ U(s) \end{pmatrix},$$

the closed-loop transfer function from w to z is given by the linear-fractional expression

$$F(P,K) := P_{11} + P_{12}K(I - P_{22}K)^{-1}P_{21} \tag{5-1}$$

Hence the optimal H_∞ control seeks to minimize $\|F(P, K)\|_\infty$ over all stabilizing LTI controllers $K(s)$. Alternatively, we can specify some maximum value γ for the closed-loop RMS gain and ask the following question:

Does there exist a stabilizing controller $K(s)$ that ensures $\|F(P, K)\|_\infty < \gamma$?

This is known as the suboptimal H_∞ control problem, and γ is called the prescribed H_∞ performance.

A number of control problems can be recast in this standard formulation.

Example 5.1. Consider the following disturbance rejection problem relative to the tracking loop of Figure 5-2:

For disturbances d with spectral density concentrated between 0 and 1 rad/s, maintain the closed-loop RMS gain from d to the output y below 1%.

Since the RMS gain is the largest gain over all square-integrable inputs, external signals with uneven spectral density must be specified by means of shaping filters. For instance, a disturbance d with all its energy in the

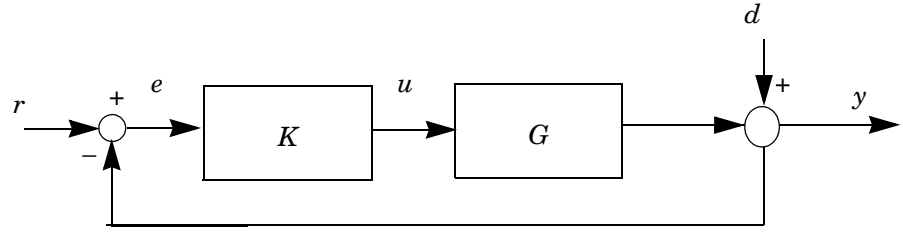


Figure 5-2: Tracking loop with external disturbance

frequency band [0,1] is described as

$$d(s) = W_{lp}(s)\tilde{d}(s)$$

where $\tilde{d}$ is an arbitrary square-integrable signal and $W_{lp}(s)$ is a low-pass filter with cutoff frequency at 1 rad/s. From the closed-loop equation

$y = Sd + Tr$ with $S := (I + GK)^{-1}$ and $T := GKS$,

the transfer function from the equalized disturbance $\tilde{d}$ to y is SW_{lp}. Hence our disturbance rejection objective is equivalent to finding a stabilizing controller $K(s)$ such that

$$\|S\ W_{lp}\|_\infty < 1.$$

This RMS gain constraint is readily turned into a standard H_∞ problem by observing that, in conformance with (5-1),

$$SW_{lp} = F(P, K) \quad \text{with} \quad Ps := \begin{pmatrix} W_{lp}(s) & -G(s) \\ W_{lp}(s) & -G(s) \end{pmatrix}$$

Example 5.2. Decoupling constraints are handled in a similar fashion. For instance, consider the loop of Figure 5-3 and the problem of decoupling the action of r_1 on y_1 from that of r_2 on y_2 in the control bandwidth ω ð 10 rad/s.

The closed-loop transfer function from $\begin{pmatrix} r_1 \\ r_2 \end{pmatrix}$ to $\begin{pmatrix} y_1 \\ y_2 \end{pmatrix}$

is $T = GK(I + GK)^{-1}$. Decoupling is achieved if $T(j\omega) \approx I$ in the bandwidth ω ð 10 rad/s, or equivalently if $S(j\omega) = I - T(j\omega)$ has a small gain in this bandwidth. This can be expressed as

$$\left\| \frac{10}{s + 10} S(s) \right\|_\infty < \varepsilon,$$

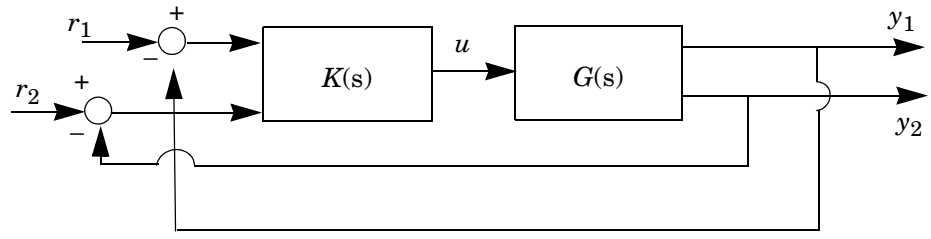
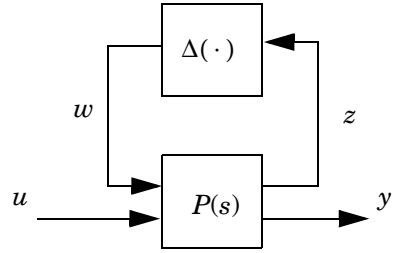


Figure 5-3: Decoupling problem

which is again of the standard form

$$\|F(P,K)\|_\infty < \varepsilon \quad \text{with} \quad P(s) := \begin{pmatrix} \frac{10}{s+10} I_2 & -\frac{10}{s+10} G(s) \\ I_2 & -G(s) \end{pmatrix}.$$

Example 5.3. H_∞ optimization is also useful for the design of robust controllers in the face of unstructured uncertainty. Consider the uncertain system



where the uncertainty $\Delta(\cdot)$ is an arbitrary BIBO-stable system satisfying the norm bound $\|\Delta\|_\infty < 1$. From the small gain theorem, the controller $u = K(s)y$ robustly stabilizes this uncertain system if the nominal closed-loop transfer function $\mathcal{F}(P, K)$ from w to z satisfies

$$\|\mathcal{F}(P, K)\|_\infty < 1.$$

Finally, a useful application of H_∞ synthesis is the loop shaping design procedure discussed in the next chapter.

Riccati- and LMI-Based Approaches

Since the continuous- and discrete-time cases are essentially similar, we review only continuous-time H_∞ synthesis. Both the Riccati-based and LMI-based approaches are implemented in the LMI Control Toolbox. While the LMI-based approach is computationally more involved for large problems, it has the merit of eliminating the regularity restrictions attached to the Riccati-based solution. Since both approaches are state-space methods, the plant $P(s)$ is given in state-space form by

$$\begin{aligned} \dot{x} &= Ax + B_1 w + B_2 u \\ z &= C_1 x + D_{11} w + D_{12} u \\ y &= C_2 x + D_{21} w + D_{22} u \end{aligned}$$

The Riccati-based approach is applicable to plants P satisfying

(A1) D_{12} and D_{21} have full rank,

(A2) $P_{12}(s)$ and $P_{21}(s)$ have no zeros on the $j\omega$-axis.

Given $\gamma > 0$, it gives necessary and sufficient conditions for the existence of internally stabilizing controllers $K(s)$ such that

$$\|\mathcal{F}(P, K)\|_\infty < \gamma$$

Specifically, the H_∞ performance γ is achievable if and only if [2]:

(i) $\gamma > \max\ (\sigma_{\max}(I - D_{12}D_{12}^{+})D_{11}),\ \sigma_{\max}(D_{11}(I - D_{21}^{+}D_{21})))$

(ii) the Riccati equations associated with the Hamiltonian pencils

$$H-\lambda\varepsilon=\left(\begin{array}{cc|ccc} A & 0 & 0 & \gamma^{-1}B_1 & B_2 \\ 0 & -A^T & -C_1^T & 0 & 0 \\ \hline C_1 & 0 & -I & \gamma^{-1}D_{11} & D_{12} \\ 0 & \gamma^{-1}B_1^T & \gamma^{-1}D_{11}^T & -I & 0 \\ 0 & B_2^T & D_{12}^T & 0 & 0 \end{array}\right)-\lambda\left(\begin{array}{cc|ccc} I & 0 & & & \\ 0 & 0 & & & \\ \hline & & & & \end{array}\right)$$

$$J-\lambda\varepsilon=\left(\begin{array}{cc|ccc} A^T & 0 & 0 & \gamma^{-1}C_1^T & C_2^T \\ 0 & -A & -B_1 & 0 & 0 \\ \hline B_1^T & 0 & -I & \gamma^{-1}D_{11}^T & D_{21}^T \\ 0 & \gamma^{-1}C_1 & \gamma^{-1}D_{11} & -I & 0 \\ 0 & C_2 & D_{21} & 0 & 0 \end{array}\right)-\lambda\left(\begin{array}{cc|ccc} I & 0 & & & \\ 0 & 0 & & & \\ \hline & & & & \end{array}\right)$$

have stabilizing solutions X_∞ and Y_∞, respectively

(iii) X_∞ and Y_∞ further satisfy

$$X_\infty \geq 0, \qquad Y_\infty \geq 0, \qquad \rho(X_\infty Y_\infty) < \gamma^2$$

Using this characterization, the optimal H_∞ performance γ_{opt} can be computed by bisection (the so-called γ-*iterations*). An H_∞ controller with performance $\gamma \geq \gamma_{opt}$ is then given by explicit formulas involving X_∞, Y_∞, and the plant state-space matrices. Singular H_∞ problems (those violating (A1)–(A2)) require regularization by small perturbation. A notable exception is the direct computation of the optimal H_∞ performance when D_{12} or D_{21} is singular [5].

In comparison, the LMI approach is applicable to any plant and does not involve γ-iterations. Rather, the H_∞ performance is directly optimized by solving the following LMI problem [4, 6]:

Minimize γ over $R = R^T$ and $S = S^T$ such that

$$\begin{pmatrix} N_{12} & 0 \\ 0 & I \end{pmatrix}^T \begin{pmatrix} AR + RA^T & RC_1^T & B_1 \\ C_1R & -\gamma I & D_{11} \\ B_1^T & D_{11}^T & -\gamma I \end{pmatrix} \begin{pmatrix} N_{12} & 0 \\ 0 & I \end{pmatrix} < 0 \tag{5-2}$$

$$\begin{pmatrix} N_{21} & 0 \\ 0 & I \end{pmatrix}^T \begin{pmatrix} A^TS + SA & SB_1 & C_1^T \\ B_1^TS & -\gamma I & D_{11}^T \\ C_1 & D_{11} & -\gamma I \end{pmatrix} \begin{pmatrix} N_{21} & 0 \\ 0 & I \end{pmatrix} < 0 \tag{5-3}$$

$$\begin{pmatrix} R & I \\ I & S \end{pmatrix} \geq 0 \tag{5-4}$$

where N_{12} and N_{21} denote bases of the null spaces of(B_2^T, D_{12}^T) and (C_2, D_{21}), respectively.

This problem falls within the scope of the LMI solver `mincx`. Note that the LMI constraints (5-2)–(5-3) amount to the inequality counterpart of the H_∞ Riccati equations, R^{-1} and S^{-1} being solutions of these inequalities. Again, explicit formulas are available to derive H_∞ controllers from any solution (R, S, γ) of (5-2)–(5-4) [3].

H_∞ Synthesis

The LMI Control Toolbox supports continuous- and discrete-time H_∞ synthesis using either Riccati- or LMI-based approaches. Transparency and numerical reliability were primary concerns in the development of the related tools. Original features of the Riccati-based functions include:

- The use of pencil-based Riccati solvers for highly accurate computation of Riccati solutions (see `care`, `ricpen`, and their discrete-time counterparts)
- Direct computation of the optimal H_∞ performance when D_{12} and D_{21} are rank-deficient (without using regularization) [5]
- Automatic regularization of singular problems for the controller computation.

In addition, reduced-order controllers are computed whenever the matrices $I - \gamma^{-2}X_\infty Y_\infty$ or $I - RS$ are nearly rank-deficient. The functions for standard H_∞ synthesis are listed in Table 5-1.

Table 5-1: Functions for H_∞ synthesis

H_∞ synthesis	Riccati-based	LMI-based
Continuous-time	`hinfric`	`hinflmi`
Discrete-time	`dhinfric`	`dhinflmi`

To illustrate the use of `hinfric` and `hinflmi`, consider the simple first-order plant $P(s)$ with state-space equations

$$\begin{cases} \dot{x} = w + 2u \\ z = x \\ y = -x + w \end{cases}$$

This plant is specified as a `SYSTEM` matrix by

```
a=0; b1=1; b2=2; c1=1; d11=0; d12=0; c2= 1; d21=1; d22=0;
P=ltisys(a,[b1 b2],[c1;c2],[d11 d12;d21 d22])
```

Note that the plant $P(s)$ is "singular" since $D_{12} = 0$.

To determine the optimal H_∞ performance over stabilizing control laws $u = K(s)y$, type

```
gopt = hinfric(P,[1 1])
```

where the second argument `[1 1]` specifies the numbers of measurements and controls (lengths of the vectors y and u). The γ-iterations performed by `hinfric` are displayed on the screen as follows:

```
Gamma-Iteration:

Gamma           Diagnosis
10.0000   :       feasible
1.0000  :       infeasible (Y is not positive semi-definite)
5.5000  :       feasible
3.2500  :       feasible
2.1250  :       feasible
1.5625  :       feasible
1.2812  :       feasible
1.1406  :       feasible
1.0703  :       feasible
1.0352  :       feasible
1.0176  :       feasible
1.0088  :       feasible

Best closed-loop gain (GAMMA_OPT): 1.008789
```

The optimal H_∞ performance for $P(s)$ is therefore approximately 1. For each tested value of γ, `hinfric` gives a feasible/infeasible diagnosis as well as the cause for infeasibility when applicable.

Alternatively, you can use the LMI-based function `hinflmi` with the same syntax:

```
gopt = hinflmi(P,[1 1])
```

This function optimizes the H_∞ performance using `mincx`:

```
  Minimization of gamma:

  Solver for linear objective minimization under LMI constraints
  Iterations :       Best objective value so far

          1               2.196542
***                  new lower bound:   0.571370
          2               1.381964
          3               1.381964
          4               1.094665
          5               1.052469
          6               1.008499
          7               1.008499
          8               1.008499
***                  new lower bound:  0.656673
          9               1.002517
***                  new lower bound:  0.878172
         10               1.002517
***                  new lower bound:  0.982223
         11               1.002517
***                  new lower bound:  0.993382

Result:feasible solution of required accuracy
       best objective value: 1.002517
       guaranteed relative accuracy: 9.11e 03
       f-radius saturation: 0.018% of R = 1.00e+08
```

The value displayed in the second column is the current best estimate of γ_{opt}. The optimal value 1.0025 approximately matches the value found by `hinfric`.

When a second output argument is provided, these two functions also return an optimal H_∞ controller $K(s)$:

```
[gopt,K] = hinfric(P,[1 1])
```

The output K is the SYSTEM matrix of $K(s)$. To compute a suboptimal H_∞ controller with guaranteed performance $\gamma < 10$, type

```
[g,K] = hinfric(P,[1 1],10,10)
```

In this case only the value $\gamma = 10$ is tested. In the LMI approach, the same problem is solved by the command

```
[g,K] = hinflmi(P,[1 1],10)
```

Finally, the syntax

```
[g,K,x1,x2,y1,y2] = hinfric(P,[1,1],10,10)
```

also returns the stabilizing solutions X_∞ = x2/x1 and Y_∞ = y2/y1 of the H_∞ Riccati equations for $\gamma = 10$. Similarly,

```
[g,K,x1,x2,y1,y2] = hinflmi(P,[1,1],10)
```

returns solutions X = x2/x1 and Y = y2/y1 of the H_∞ Riccati inequalities. Equivalently, R = x1 and S = y1 are solutions of the characteristic LMIs (5-2)-(5-4) since x2 and y2 are always set to g $\times I$. Further options and control parameters are discussed in the "Command Reference" chapter.

Finally, the counterpart of these functions for discrete-time H_∞ synthesis are called dhinfric, dhinflmi, and dnorminf and follow the exact same syntax.

Validation of the Closed-Loop System

Given the H_∞ controller K computed by hinfric or hinflmi, the closed-loop mapping represented in Figure 5-4 is formed with the function slft:

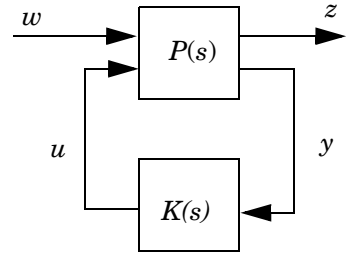


Figure 5-4: Closed-loop system

Closed-loop stability is then checked by inspecting the closed-loop poles with `spol`

```
spol(clsys)
```

while the closed-loop RMS gain from w to z is computed by

```
norminf(clsys)
```

You can also plot the time- and frequency-domain responses of the closed- loop `clsys` with `splot`.

Finally, `hinfpar` extracts the state-space matrices A, B_1, . . . from the plant `SYSTEM` matrix `P`:

```
[a,b1,b2,c1,c2,d11,d12,d21,d22] = hinfpar(P,r)
```

Set the second argument `r` to `[p2 m2]` if $D_{22} \in \mathbf{R}^{p_2 \times m_2}$.

Multi-Objective H_∞ Synthesis

In many real-world applications, standard H_∞ synthesis cannot adequately capture all design specifications. For instance, noise attenuation or regulation against random disturbances are more naturally expressed in LQG terms. Similarly, pure H_∞ synthesis only enforces closed-loop stability and does not allow for direct placement of the closed-loop poles in more specific regions of the left-half plane. Since the pole location is related to the time response and transient behavior of the feedback system, it is often desirable to impose additional damping and clustering constraints on the closed-loop dynamics. This makes multi-objective synthesis highly desirable in practice, and LMI theory offers powerful tools to attack such problems.

Mixed H_2/H_∞ synthesis with regional pole placement is one example of multi-objective design addressed by the LMI Control Toolbox. The control problem is sketched in Figure 5-5. The output channel z_∞ is associated with the H_∞ performance while the channel z_2 is associated with the LQG aspects (H_2 performance).

Denoting by $T_\infty(s)$ and $T_2(s)$ the closed-loop transfer functions from w to z_∞ and z_2, respectively, we consider the following multi-objective synthesis problem:

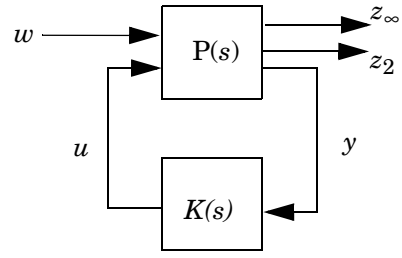


Figure 5-5: Multi-objective H_∞ synthesis

Design an output-feedback controller $u = K(s)y$ that

- Maintains the H_∞ norm of $T_\infty(s)$ (RMS gain) below some prescribed value $\gamma_0 > 0$
- Maintains the H_2 norm of $T_2(s)$ (LQG cost) below some prescribed value $\nu_0 > 0$

- Minimizes a trade-off criterion of the form

  $$\alpha\|T_\infty\|_\infty^2 + \beta\|T_2\|_2^2$$

  with $\alpha \geq 0$ and $\beta \geq 0$
- Places the closed-loop poles in some prescribed LMI region D

Recall that LMI regions are general convex subregions of the open left-half plane (see "Pole Placement in LMI Regions" on page 4-5 for details).

LMI Formulation

Let

$$\begin{cases} \dot{x} &= Ax + B_1 w + B_2 u \\ z_\infty &= C_\infty x + D_{\infty 1} w + D_{\infty 2} u \\ z_2 &= C_2 x + D_{21} w + D_{22} u \\ y &= C_y x + D_{y1} w \end{cases}$$

and

$$\begin{cases} \dot{\zeta} = A_{K\zeta} + B_{Ky} \\ \dot{u} = C_{K\zeta} + D_{Ky} \end{cases}$$

be state-space realizations of the plant $P(s)$ and controller $K(s)$, respectively, and let

$$\begin{cases} \dot{x}_{cl} &= A_{cl} x_{cl} + B_{cl}\ w \\ z_\infty &= C_{cl1} x_{cl} + D_{cl1}\ w \\ z_2 &= C_{cl2} x_{cl} + D_{cl2}\ w \end{cases}$$

be the corresponding closed-loop state-space equations.

Our three design objectives can be expressed as follows:

- H_∞ **performance:** the closed-loop RMS gain from w to z_∞ does not exceed γ if and only if there exists a symmetric matrix X_∞ such that

$$\begin{pmatrix} A_{cl}\chi_\infty + \chi_\infty A_{cl}^T & B_{cl} & X_\infty C_{cl1}^T \\ B_{cl}^T & -I & D_{cl1}^T \\ C_{cl1}\chi_\infty & D_{cl1} & -\gamma^2 I \end{pmatrix} < 0$$

$$\chi_\infty > 0$$

- H_2 **performance:** the H_2 norm of the closed-loop transfer function from w to z_2 does not exceed ν if and only if $D_{cl2} = 0$ and there exist two symmetric matrices χ_2 and Q such that

$$\begin{pmatrix} A_{cl}\chi_2 + \chi_2 A_{cl}^T & B_{cl} \\ B_{cl}^T & -I \end{pmatrix} < 0$$

$$\begin{pmatrix} Q & C_{cl2}\chi_2 \\ \chi_2 C_{cl2}^T & \chi_2 \end{pmatrix} > 0$$

$$\mathrm{Trace}(Q) < \nu^2$$

- **Pole placement:** the closed-loop poles lie in the LMI region

$$D = \{z \in \mathbf{C} : L + Mz + M^T\bar{z} < 0\}$$

with $L = L^T = \{\lambda_{ij}\}_{1 \le i,j \le m}$ and $M = [\mu_{ij}]_{1 \le i,j \le m}$ if and only if there exists a symmetric matrix χ_{pol} satisfying

$$[\lambda_{ij}\chi_{pol} + \mu_{ij}A_{cl}\chi_{pol} + \mu_{ji}X_{pol}A_{cl}^T]_{1 \le i,j \le m} < 0$$
$$\chi_{pol} > 0$$

For tractability in the LMI framework, we must seek a single Lyapunov matrix

$$\chi := \chi_\infty = \chi_2 = \chi_{\mathrm{pol}}$$

that enforces all three sets of constraints. Factorizing χ as

$$\chi = \chi_1\chi_2^{-1}, \quad \chi_1 := \begin{pmatrix} R & I \\ M^T & 0 \end{pmatrix}, \quad \chi_2 := \begin{pmatrix} 0 & S \\ I & N^T \end{pmatrix}$$

and introducing the change of controller variables [3]:

$$\begin{cases} B_K & := & NB_K + SB_2D_K \\ C_K & := & C_KM^T + D_KC_yR \\ A_K & := & NA_KM^T + NB_KC_yR + SB_2C_KM^T + S(A + B_2D_KC_y)R, \end{cases}$$

the inequality constraints on χ are readily turned into LMI constraints in the variables R, S, Q, A_K, B_K, C_K, and D_K [8, 1]. This leads to the following suboptimal LMI formulation of our multi-objective synthesis problem:

Minimize $\alpha\,\gamma^2 + \beta\,\mathrm{Trace}(Q)$ over R, S, Q, A_K, B_K, C_K, D_K, and γ^2 satisfying:

$$\begin{pmatrix} AR + RA^T + B_2C_K + C_K^TB_2^T & A_K^T + A + B_2D_KC_y & B_1 + B_2D_KD_{y1} & H \\ H & A^TS + SA + B_KC_y + C_y^TB_K^T & SB_1 + B_KD_{y1} & H \\ H & H & -I & H \\ C_\infty R + D_{\infty 2}C_K & C_\infty + D_{\infty 2}D_KC_y & D_{\infty 1} + D_{\infty 2}D_KD_{y1} & -\gamma^2 I \end{pmatrix} < 0$$

$$\begin{pmatrix} Q & C_2R + D_{22}C_K & C_2 + D_{22}D_KC_y \\ H & R & I \\ H & I & S \end{pmatrix} > 0$$

$$\left[\lambda_{ij}\begin{pmatrix} R & I \\ I & S \end{pmatrix} + \mu_{ij}\begin{pmatrix} AR + B_2C_K & A + B_2D_KC_y \\ A_K & SA + B_KC_y \end{pmatrix} + \mu_{ji}\begin{pmatrix} RA^T + C_K^TB_2^T & A_K^T \\ (A + B_2D_KC_y)^T & A^TS + C_y^TB_K^T \end{pmatrix} \right]_{1 \le i,j \le m} < 0$$

$$\mathrm{Trace}\,Q < \nu_0^2$$

$$\gamma^2 < \gamma_0^2$$

$$D_{21} + D_{22}D_KD_{y1} = 0$$

Given optimal solutions γ^*, Q^* of this LMI problem, the closed-loop H_∞ and H_2 performances are bounded by

$$\|T_\infty\|_\infty \le \gamma^*, \qquad \|T_2\|_2 \le \sqrt{\mathrm{Trace}(Q^*)}.$$

The Function hinfmix

The function `hinfmix` implements the LMI approach to mixed H_2/H_∞ synthesis with regional pole placement described above. Its syntax is

```
[gopt,h2opt,K,R,S] = hinfmix(P,r,obj,region)
```

where

- `P` is the `SYSTEM` matrix of the LTI plant $P(s)$. Note that z_2 or z_∞ can be empty, or even both when performing pure pole placement
- `r` is a three-entry vector listing the lengths of z_2, y, and u
- `obj` = $[\gamma_0, \nu_0, \alpha, \beta]$ is a four-entry vector specifying the H_2/H_∞ constraints and criterion
- `region` specifies the LMI region for pole placement, the default being the open left-half plane. Use `lmireg` to interactively generate the matrix `region`.

The outputs `gopt` and `h2opt` are the guaranteed H_∞ and H_2 performances, `K` is the controller `SYSTEM` matrix, and `R,S` are optimal values of the variables R, S (see "LMI Formulation" on page 5-16).

You can perform the following mixed and unmixed designs by setting `obj` appropriately:

`obj`	**Corresponding Design**
`[0 0 0 0]`	pole placement only
`[0 0 1 0]`	H_∞-optimal design
`[0 0 0 1]`	H_2-optimal design
`[g 0 0 1]`	minimize $\|T_2\|_2$ subject to $\|T_\infty\|_\infty <$ `g`
`[0 h 1 0]`	minimize $\|T_\infty\|_\infty$ subject to $\|T_2\|_2 <$ `h`
`[0 0 a b]`	minimize $a\|T_\infty\|_0^2 + b\|T_2\|_2^2$
`[g h a b]`	most general problem

Loop-Shaping Design with hinfmix

In the loop-shaping context (see next chapter), the closed-loop poles also include the poles of the shaping filters. Since these poles are not affected by the

controller, they impose hard limitations on achievable closed-loop clustering objectives. In particular, shaping filters with modes close to the imaginary axis will doom any attempt to push the closed-loop modes into a well-damped region.

When the purpose of such filters is to enforce integral control action, you can eliminate this difficulty by moving the integrator from the shaping filter to the control loop itself. This amounts to reparametrizing the controller as

$$K(s) = \frac{1}{s}\tilde{K}(s)$$

and performing the synthesis for $\tilde{K}$. An example can be found at the end of Chapter 6.

References

[1] Chilali, M., and P. Gahinet, "H_∞ Design with Pole Placement Constraints: an LMI Approach," to appear in *IEEE Trans. Aut. Contr.*, 1995.

[2] Doyle, J.C., Glover, K., Khargonekar, P., and Francis, B., "State-Space Solutions to Standard H_2 and H_∞ Control Problems," *IEEE Trans. Aut. Contr.*, AC–34 (1989), pp. 831–847.

[3] Gahinet, P., "Explicit Controller Formulas for LMI-based H_∞ Synthesis," submitted to *Automatica*. Also in *Proc. Amer. Contr. Conf.*, 1994, pp. 2396–2400.

[4] Gahinet, P., and P. Apkarian, "A Linear Matrix Inequality Approach to H_∞ Control," Int. *J. Robust and Nonlinear Contr.*, 4 (1994), pp. 421–448.

[5] Gahinet, P., and A.J. Laub, "Reliable Computation of γ_{opt} in Singular H_∞ Control," to appear in *SIAM J. Contr. Opt.*, 1995. Also in *Proc. Conf. Dec. Contr.*, Lake Buena Vista, Fl., 1994, pp. 1527–1532.

[6] Iwasaki, *T.*, and R.E. Skelton, "All Controllers for the General H_∞ Control Problem: LMI Existence Conditions and State-Space Formulas," *Automatica*, 30 (1994), pp. 1307–1317.

[7] Scherer, C., "H_∞ Optimization without Assumptions on Finite or Infinite Zeros," *SIAM J. Contr. Opt.*, 30 (1992), pp. 143–166.

[8] Scherer, C., "Mixed H_2H_∞ Control," to appear in *Trends in Control: A European Perspective*, volume of the special contributions to the ECC 1995.

6

Loop Shaping

The Loop-Shaping Methodology 6-2

The Loop-Shaping Methodology 6-3

Design Example 6-5

Specification of the Shaping Filters 6-10
Nonproper Filters and sderiv 6-12

Specification of the Control Structure 6-14

Controller Synthesis and Validation 6-16

Practical Considerations 6-18

Loop Shaping with Regional Pole Placement 6-19

References 6-24

The Loop-Shaping Methodology

Loop shaping is a popular technique for combining specifications such as tracking performance, bandwidth, disturbance rejection, roll-off, robustness to model uncertainty, gain limitation, etc. In this frequency domain method, the design specifications are reflected as gain constraints on the various closed-loop transfer functions.

Loop-shaping design involves four steps:

1. Express the design specifications as constraints on the gain response of the closed-loop transfer functions. This defines ideal gain profiles called *loop shapes*.
2. Specify the desired loop shapes graphically with the GUI `magshape`.
3. Specify the control structure, i.e., how the feedback loop is organized and which input/output transfer functions are relevant to the loop-shaping objectives.
4. Perform a standard H_∞ synthesis on the resulting control structure to compute an adequate controller.

This chapter contains a tutorial introduction to the loop-shaping methodology as well as the available tools for loop-shaping design.

The Loop-Shaping Methodology

Loop shaping is a design procedure to formulate frequency-domain specifications as H_∞ constraints and standard H_∞ synthesis problems. In loop-shaping design, the performance and robustness specifications are first expressed in terms of loop shapes, i.e., of desired gain responses for the various closed-loop transfer functions. These shaping objectives are then turned into uniform H_∞ bounds by means of s*haping filters*, and standard H_∞ synthesis algorithms are applied to compute an adequate controller.

To get a feeling for the loop-shaping methodology, consider the MIMO tracking loop of Figure 6-1 and suppose that we want to design a controller with integral behavior. This requirement is naturally expressed in terms of the open-loop response $GK(s)$, for instance by a gain constraint of the form

$$\sigma_{\min}(GK(j\omega)) > 1/\omega \quad \text{for} \quad \omega \ll 1 \tag{6-1}$$

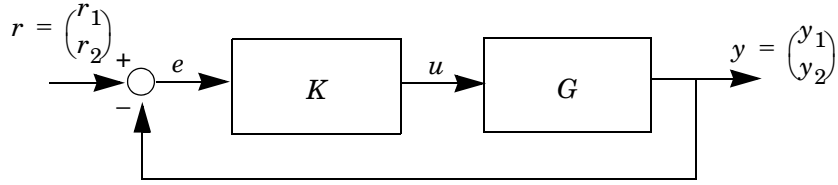


Figure 6-1: Simple tracking loop

Here the main closed-loop transfer functions are:

- The sensitivity function $S = (I + GK)^{-1}$ (the transfer function from r to e)
- The complementary sensitivity function $T = GK(I + GK)^{-1}$ (the transfer function from r to y).

To express the open-loop specification (6-1) in terms of these closed-loop transfer functions, observe that $\sigma_{\min}(GK(j\omega)) \approx 1/\sigma_{\max}(S(j\omega))$ whenever the open-loop gain is large, i.e., $\sigma_{\min}(GK(j\omega)) >> 1$. Thus, (6-1) is equivalent to requiring that

$$\sigma_{\max}(S(j\omega)) < \omega \quad \text{for} \quad \omega \ll 1 \tag{6-2}$$

To turn this frequency-dependent gain constraint into a normalized H_∞ bound, a useful tool is the notion of *shaping filter*. Shaping filters are rational filters whose magnitude plot reflects the desired loop shape. Here for instance, the shaping filter

$$W(s) = \frac{1}{s} I_2$$

can be used to normalize the constraint (6-2) to

$$\|W(s)S(s)\|_\infty < 1 \qquad \textbf{(6-3)}$$

Indeed, (6-3) is equivalent to

$$\frac{1}{\omega} < \sigma_{\min}(1 + GK(j\omega)) \text{ for all } \omega,$$

which in turn constrains GK to

$$\sigma_{\min}(GK(j\omega)) > \frac{1}{\omega} - 1 \approx \frac{1}{\omega} \text{ for } \omega << 1$$

Likewise, roll-off requirements on $GK(j\omega)$ are enforced by H_∞ bounds of the form

$$\|W(s)T(s)\|_\infty < 1$$

where $W(s)$ is some appropriate high-pass filter. This reformulation is rooted in the approximation $\sigma_{\max}(GK(j\omega)) \approx \sigma_{\max}(T(j\omega))$ valid whenever the open-loop gain is small, i.e., $\sigma_{\max}(GK(j\omega)) << 1$. Such constraints are also associated with robust stability in the face of plant uncertainty of the form $G(s) = (I + \Delta(s))G_0(s)$ where Δ satisfies the frequency-dependent bound

$$\sigma_{\max}(\Delta(j\omega)) < |W(j\omega)|$$

Finally, shaping filters are useful to incorporate information about the spectral density of exogenous signals such as disturbances or measurement noise (see “Example 5.1” on page 5-4).

Design Example

Loop shaping design with the LMI Control Toolbox involves four steps:

1. Express the design specifications in terms of loop shapes and shaping filters.
2. Specify the shaping filters by their magnitude profile. This is done interactively with the graphical user interface `magshape`.
3. Specify the control loop structure with the functions `sconnect` and `smult`, or alternatively with Simulink.
4. Solve the resulting H_∞ problem with one of the H_∞ synthesis functions.

A simple tracking problem is used to illustrate this design procedure. Type `radardem` to run the corresponding demo.

Example 6.1. Figure 6-2 shows a simplified mechanical model of a radar antenna. The stiffness K accounts for flexibilities in the coupling between the motor and the antenna.

The corresponding nominal transfer function from the motor torque $u = T$ to the angular velocity $y = \dot{\theta}_a$ of the antenna is

$$G_0(s) = \frac{30000}{(s + 0.02)(s^2 + 0.99s + 30030)}.$$

Our goal is to control $\dot{\theta}_a$ through the torque T. The control structure is the tracking loop of Figure 6-3 where the dynamical uncertainty $\Delta(s)$ accounts for neglected high-frequency dynamics and flexibilities.

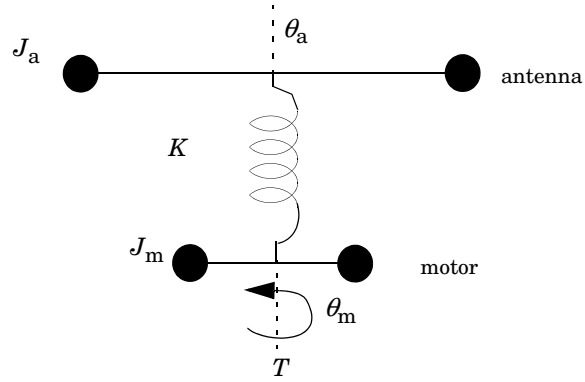


Figure 6-2: Second-order model of a radar antenna

The design specifications are as follows:

(*i*)integral control action, open-loop gain of at least 30 dB at 1 rad/*s*, and bandwidth of at least 10 rad/*s*,

(*ii*)99% rejection of output disturbances d with spectral density in [0, 1] rad/*s*,

(*iii*)robustness against the neglected high-frequency dynamics represented by the multiplicative model uncertainty $\Delta(s)$. A bound on $|\Delta(j\omega)|$ determined from experimental data is given in Figure 6-4.

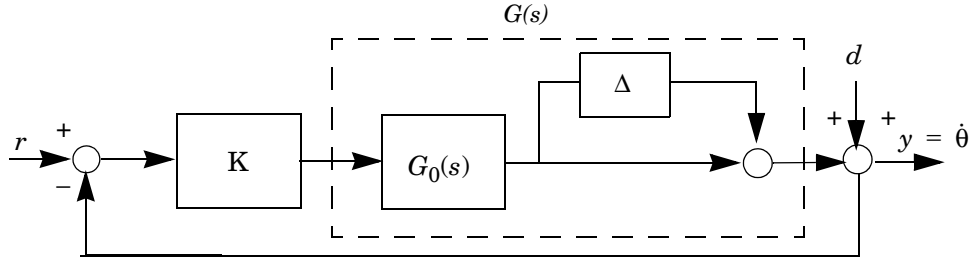


Figure 6-3: Velocity loop for the radar antenna

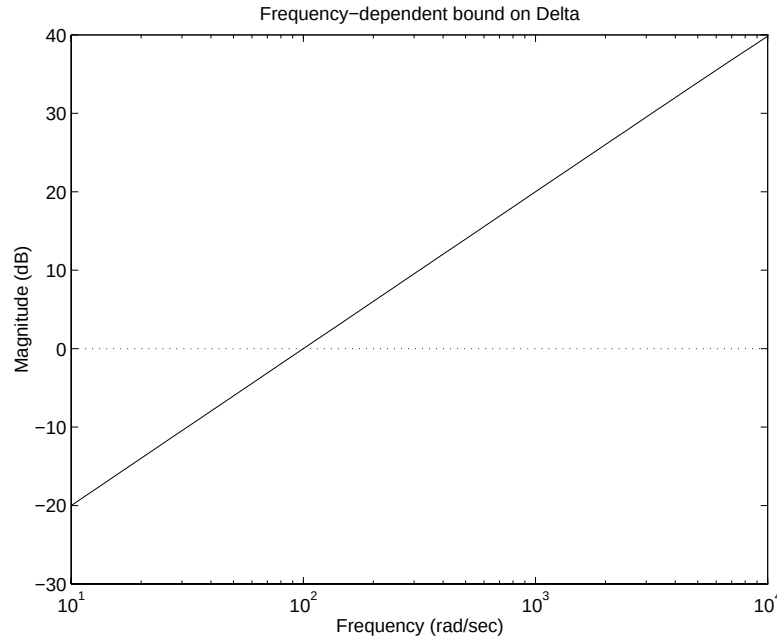


Figure 6-4: Empirical bound on the uncertainty $\Delta(s)$

We first convert these specifications into RMS gain constraints by means of shaping filters. While the performance requirement (*i*) amounts to a lower bound on the open-loop gain, the disturbance rejection target (*ii*) imposes $|S(j\omega)| < 0.01$ for $\omega < 1$ rad/*s*. Both requirements can be combined into a single constraint

$$\left\| W_1(s)S(s) \right\|_\infty < 1 \qquad \textbf{(6-4)}$$

where $S = 1 = (1 + G_0K)$ and $W_1(s)$ is a low-pass filter whose gain lies above the shaded region in Figure 6-5. From the small gain theorem [2], robust stability in the face of the uncertainty $\Delta(s)$ is equivalent to

$$\left\| W_2(s)T(s) \right\|_\infty < 1 \qquad \textbf{(6-5)}$$

where $T = G_0K = (1 + G_0K)$ and W_2 is a high pass filter whose gain follows the profile of Figure 6-4.

For tractability in the H_∞ framework, (6-4)–(6-5) are replaced by the single RMS gain constraint

$$\left\| \begin{matrix} W_1S \\ W_2T \end{matrix} \right\|_\infty < 1. \tag{6-6}$$

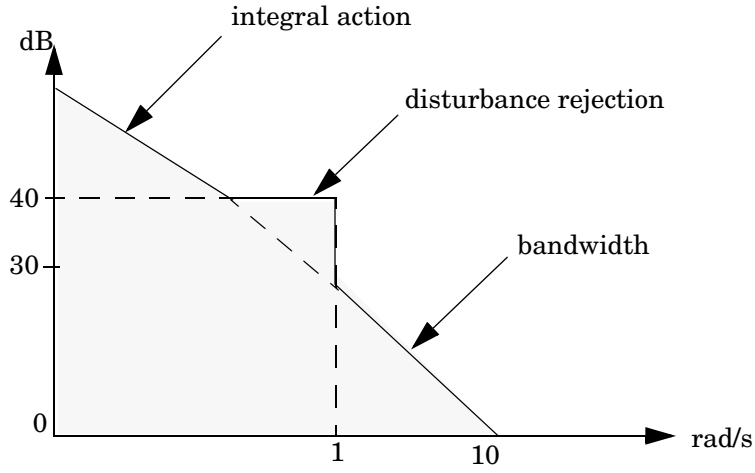


Figure 6-5: Magnitude profile for $W_1(s)$

The transfer function $\left(\begin{matrix} W_1S \\ W_2T \end{matrix} \right)$

maps r to the signals $\tilde{e}$, $\tilde{y}$ in the control loop of Figure 6-6. Hence our original design problem is equivalent to the following H_∞ synthesis problem:

> Find a stabilizing controller $K(s)$ that makes the closed-loop RMS gain from r to $\left(\begin{matrix} \tilde{e} \\ \tilde{y} \end{matrix} \right)$ less than one.

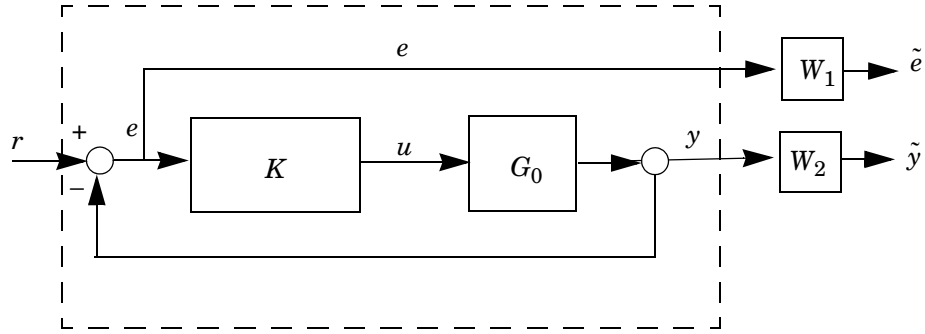


Figure 6-6: Control structure

To perform this H_∞ synthesis, we need to

1 Compute two shaping filters $W_1(s)$ and $W_2(s)$ whose magnitude responses match the profiles of Figures 6-5 and 6-4.

2 Specify the control structure of Figure 6-6 and derive the corresponding plant.

3 Call one of the H_∞-optimization functions to compute an adequate controller $K(s)$ for this plant.

The next three sections discuss these steps.

Specification of the Shaping Filters

The graphical user interface `magshape` allows you to specify shaping filters directly by their magnitude profile. Typing `magshape` brings up the worksheet shown in Figure 6-7. To construct the two shaping filters $W_1(s)$ and $W_2(s)$ involved in our problem:

1. Type `w1,w2` after the prompt `Filter names`. This tells `magshape` to write the filter `SYSTEM` matrices in MATLAB variables called `w1` and `w2`.

2. Indicate which filter you are about to specify by clicking on the corresponding button on the right-hand side of the window.

3. Click on the `add point` button: you can now specify the desired magnitude profile with the mouse. This profile is sketched with a few characteristic points marking the asymptotes and the slope changes. Clicking on the `delete point` or `move point` buttons allows you to delete or move particular points.

4. Once the desired profile is sketched, specify the filter order and click the `fit data` button to interpolate the data points by a rational filter. The gain of the resulting filter appears in solid on the plot, and its realization is written in the MATLAB variable of the same name. To make adjustments, you can move some of the points or change the filter order, and then redo the fitting.

5. If `w1` is used as a filter name, `magshape` checks if a `SYSTEM` matrix of the same name already exists in the MATLAB environment. When this is the case, its magnitude plot is drawn when the `w1` button is clicked. This is useful to load existing filters in the `magshape` window for modification or comparison.

Figure 6-7 is a snapshot of the `magshape` window after specifying the profiles of W_1 and W_2 and fitting the data points (marked by `o` in the plot). The solid lines show the rational fits obtained with an order 3 for W_1 and an order 1 for W_2.

The transfer functions of these filters (as given by `ltitf(w1)` and `ltitf(w2)`) are approximately

$$W_1(s) = \frac{0.64s^3 + 10.45s^2 + 149.28s + 25.87}{s^3 + 1.20s^2 + 0.83s + 8.31 \times 10^{-4}}$$

$$W_2(s) = 100\frac{3.41s + 217.75}{s + 3.54 \times 10^4}$$

Note that horizontal asymptotes have been added in both filters. Such asymptotes prevent humps in S and T near the crossover frequency, thus reducing the overshoot in the step response.

The function `magshape` always returns stable and proper filters. If you do not use `magshape` to generate your filters, make sure that they are stable with poles sufficiently far from the imaginary axis. To avoid difficulties in the subsequent H_∞ optimization, their poles should satisfy

$$\mathrm{Re}(s) \le -10^{-3}$$

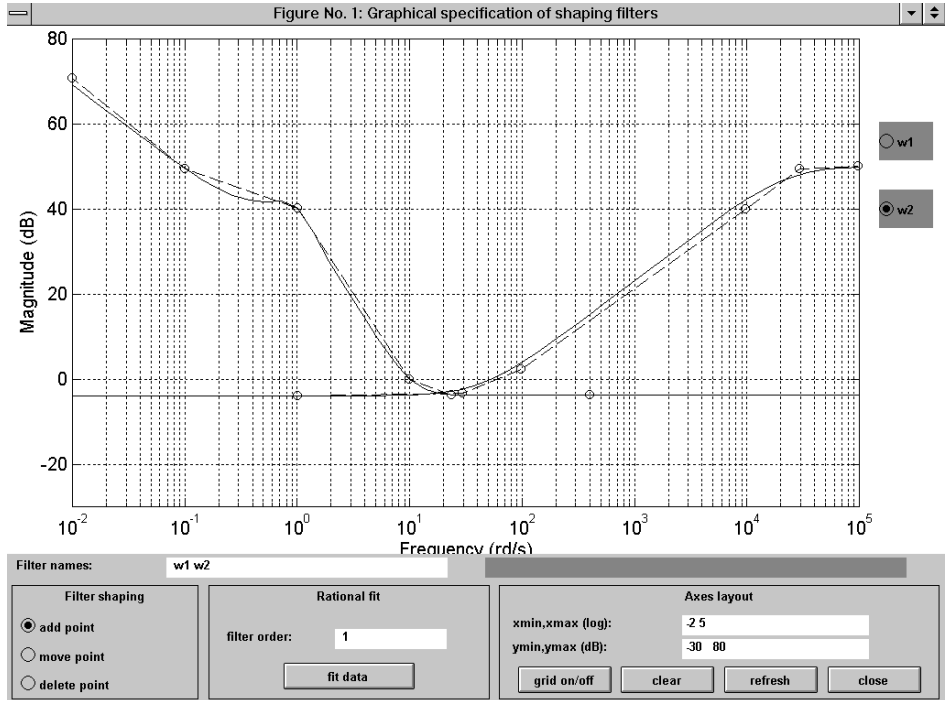


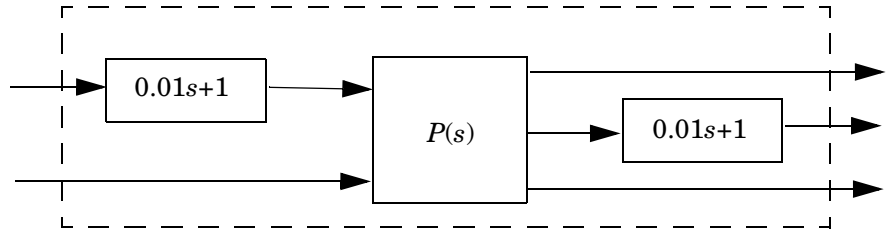
Figure 6-7: The magshape interface

Nonproper Filters and sderiv

To impose a given roll-off rate in the open-loop response, it is often desirable to use nonproper shaping filters, i.e., filters with a derivative action in the high-frequency range. Because such filters cannot be represented in SYSTEM matrix form, `magshape` approximates them by high-pass filters. A drawback of this approximation is the introduction of fast parasitic modes in the filter and augmented plant realizations, which in turn may cause numerical difficulties.

Alternatively, you can use `sderiv` to include nonproper shaping filters in the loop-shaping criterion. This function appends a SISO proportional-derivator

component $ns + d$ to selected inputs and/or outputs of a given LTI system. For instance, the interconnection



is specified by

```
Pd = sderiv(P,[ 1 2],[0.01 1])
```

In the calling list, `[ 1 2]` lists the input and output channels to be filtered by $ns + d$ (here `1` for "first input" and `2` for "second output") and the vector `[0.01 1]` lists the values of n and d. An error is generated if the resulting system `Pd` is not proper.

To specify more complex nonproper filters,

1. Specify the proper "low-frequency" part of the filters with `magshape`.
2. Augment the plant with these low-pass filters.
3. Add the derivative action of the nonproper filters by applying `sderiv` to the augmented plant.

This procedure is illustrated in the design example presented in "Loop Shaping with Regional Pole Placement" on page 6-19.

Specification of the Control Structure

To perform the H_∞ synthesis, the control structure of Figure 6-6 must be put in the standard linear-fractional form of Figure 6-8. In this equivalent representation, $P(s)$ is the nominal plant determined by $G_0(s)$ and the control structure, and $P_{aug}(s)$ is the *augmented plant* obtained by appending the shaping filters $W_1(s)$ and $W_2(s)$. While the H_∞ synthesis is performed on $P_{aug}(s)$, the physical closed-loop transfer functions from r to e and y are fully specified by $P(s)$ and $K(s)$ (recall that the shaping filters are not part of the physical control loop).

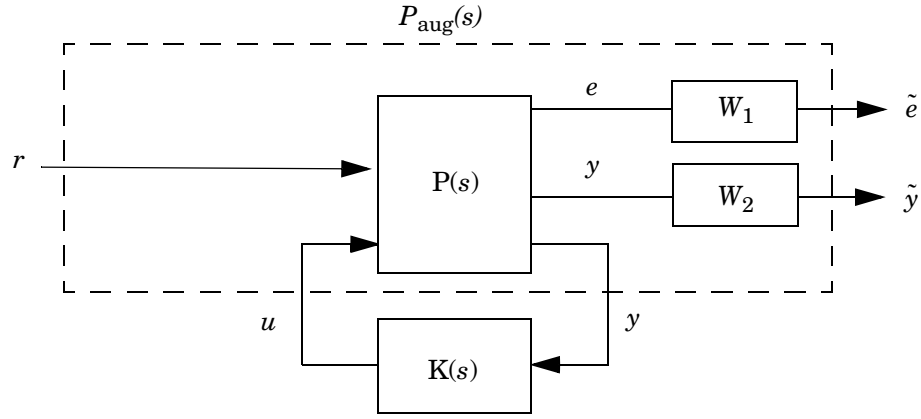
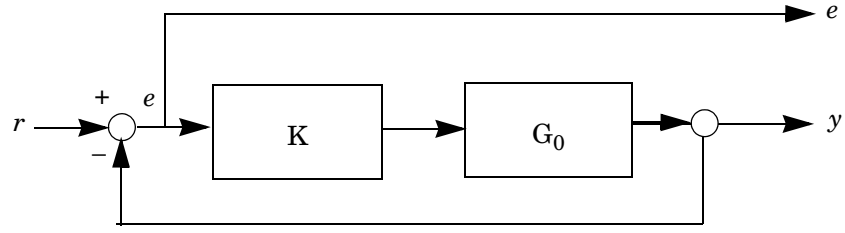


Figure 6-8: Equivalent standard H_∞ problem

The function `sconnect` computes the `SYSTEM` matrices of $P(s)$ or $P_{aug}(s)$ given G_0, W_1, W_2, and the qualitative description of the control structure (Figure 6-6). With this function, general control structures are described by listing the input and output signals and by specifying the input of each dynamical system. In our problem, the plant $P(s)$ corresponding to the control loop



is specified by

```
g0 = ltisys('tf',3e4,conv([1.02],[1.99 3.03e4]))

P = sconnect('r(1)','e=r-G0 ; G0','K:e','G0:K',g0)
```

The `ltisys` command returns a state-space realization of $G_0(s)$. In the `sconnect` command, the input arguments are strings or `SYSTEM` matrices specifying the control structure as follows:

- The first argument `'r(1)'` lists and dimensions the input signals. Here the only input is the scalar reference signal r.
- The second argument `'e=r-G0 ; G0'` lists the two output signals separated by a semicolon. Outputs are defined as combinations of the input signals and the outputs of the dynamical systems. For instance, the first output e is specified as r minus the output y of G_0. For systems with several outputs, the syntax `G0([1,3:4])` would select the first, third, and fourth outputs.
- The third argument `'K:e'` names the controller and specifies its inputs. A two-degrees-of-freedom controller with inputs e and r would be specified by `'K: e;r'`.
- The remaining arguments come in pairs and specify, for each known LTI system in the loop, its input list and its `SYSTEM` matrix. Here `'G0:K'` means that the input of G_0 is the output of K, and `g0` is the `SYSTEM` matrix of $G_0(s)$.

You can give arbitrary names to G_0 and K in the string definitions, provided that you use the same names throughout.

Similarly, the augmented plant $P_{\text{aug}}(s)$ corresponding to the loop of Figure 6-6 would be specified by

```
Paug = sconnect('r(1)','W1;W2','K:e = r-G0','G0:K',g0,...
'W1:e',w1,'W2:G0',w2)
```

where `w1` and `w2` are the shaping filter `SYSTEM` matrices returned by `magshape`. However, the same result is obtained more directly by appending the shaping filters to $P(s)$ according to Figure 6-8:

```
Paug = smult(P,sdiag(w1,w2,1))
```

Finally, note that `sconnect` is also useful to compute the `SYSTEM` matrix of ordinary system interconnections. In such cases, the third argument should be set to `[]` to signal the absence of a controller.

Controller Synthesis and Validation

After forming the augmented plant $P_{aug}(s)$ with `sconnect`, it suffices to call `hinfric` or `hinflmi` to test whether the specifications are feasible and compute an adequate controller $K(s)$. The loop-shaping design is feasible if and only if the H_∞ performance $\gamma = 1$ is achievable for P_{aug}. With our choice of shaping filters, we obtain

```
[gopt,k] = hinfric(Paug,[1 1],1,1);

Gamma-Iteration:

Gamma Diagnosis
1.0000 : feasible

Best closed-loop gain (GAMMA_OPT): 1.000000


** regularizing D12 or D21 to compute K(s):
..
```

Hence the specifications are met by the controller `k` returned by `hinfric`. The last two lines indicate that the H_∞ problem associated with P_{aug} is singular and has been regularized.

To validate the controller `k`, you can form the open-loop response GK by typing

```
gk = slft(k,g)
```

or the closed-loop transfer function from r to e and y by typing

```
Pcl = slft(P,k)
```

The Bode diagram of GK and the nominal step response are then plotted by

```
splot(gk,'bo')
splot(ssub(Pcl,1,2),'st')
```

In the second command, `ssub(Pcl,1,2)` selects the transfer function from r to y as a subsystem of `Pcl`. The step response with the controller computed above appears in Figure 6-9.

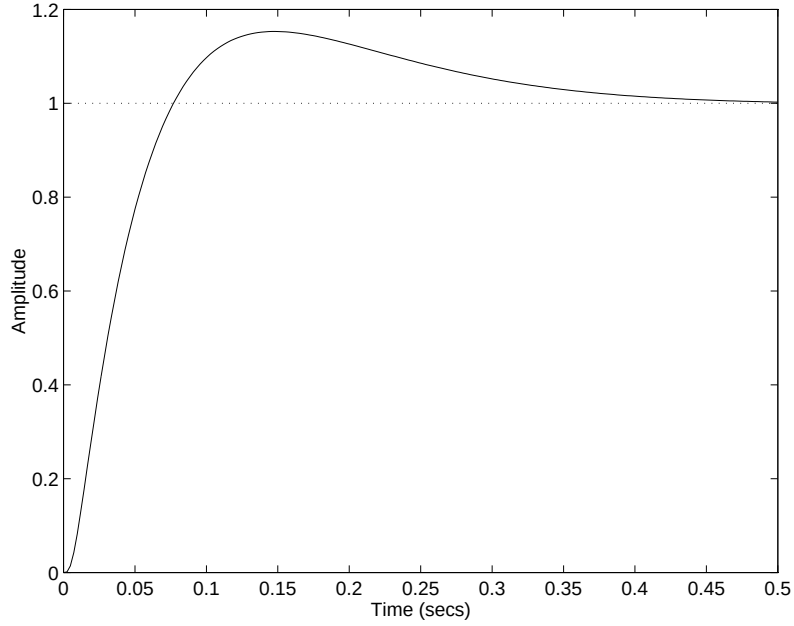


Figure 6-9: Step response of the closed-loop system

Practical Considerations

When the model $G_0(s)$ of the system has poles on the imaginary axis, loop-shaping design often leads to "singular" H_∞ problems with $j\omega$-axis zeros in the (1,2) or (2,1) blocks of P_{aug}. While immaterial in the LMI approach, jω-axis zeros bar the direct use of standard Riccati-based synthesis algorithms [1]. The function `hinfric` automatically regularizes such problems by perturbing the A matrix of P_{aug} to A + εI for some positive ε. The message

```
** jw-axis zeros of P12(s) regularized
```

or

```
** jw-axis zeros of P21(s) regularized
```

is then displayed to inform users of the presence of a singularity.

In doing so, however, `hinfric` shifts the shaping filter poles as part of the poles of P_{aug}. Since the shaping filters must remain stable for closed-loop stabilizability, the amount of regularization ε is thereby limited. When some shaping filters have modes close to the $j\omega$-axis, this is likely to result in poor closed-loop damping because the central controller computed by `hinfric` tends, in such problems, to place closed-loop poles within ε of the $j\omega$-axis.

If this difficulty arises, a simple remedy consists of perturbing the poles of the plant $P(s)$ *prior* to appending the shaping filters. Provided that all poles of $G_0(s)$ are controllable and observable, this typically tolerates much larger ε, which in turn results in better closed-loop damping. In our example, this would amount to forming P_{aug} as follows:

```
P = sconnect('r(1)','e=r-GO ; GO','K:e','GO:K',gO)

% pre-regularization
[a,b,c,d] = ltiss(P)
a = a + reg * eye(size(a))
Preg = ltisys(a,b,c,d)

Paug = smult(Preg, sdiag(w1,w2,1))
```

Note that the regularization level `reg` should always be chosen *positive*. Finally, a careful validation of the resulting controller on the *true* plant `P` is necessary since the H_∞ synthesis is performed on a perturbed system.

Loop Shaping with Regional Pole Placement

A weakness of the design outlined above is the plant inversion performed by the controller $K(s)$. Specifically, the resonant poles of $G(s)$ are cancelled by zeros of $K(s)$ as seen when comparing the magnitude plots of G and K (see Figure 6-10). Since such cancellations leave resonant modes in the closed-loop system, oscillatory responses may result when these modes are excited, e.g., by a disturbance at the plant input (see Figure 6-11).

There are several ways of circumventing this difficulty. One possible approach is to impose some minimum closed-loop damping via an additional regional pole placement constraint. In addition to avoiding cancellations, this is helpful to tune the transient responses and prevent fast controller dynamics. Loop-shaping design with pole placement is addressed by the function `hinfmix`. As mentioned in the previous chapter, using `hinfmix` requires extra care because of possible interferences between the root clustering objective and the shaping filter poles. Indeed, recall that the "closed-loop" dynamics of the loop-shaping criterion always comprise the shaping filter modes.

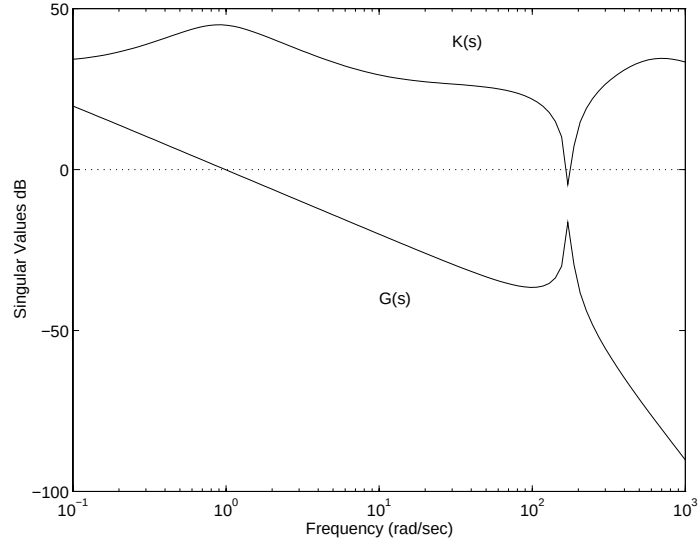


Figure 6-10: Pole-zero cancellation between *G* and *K*

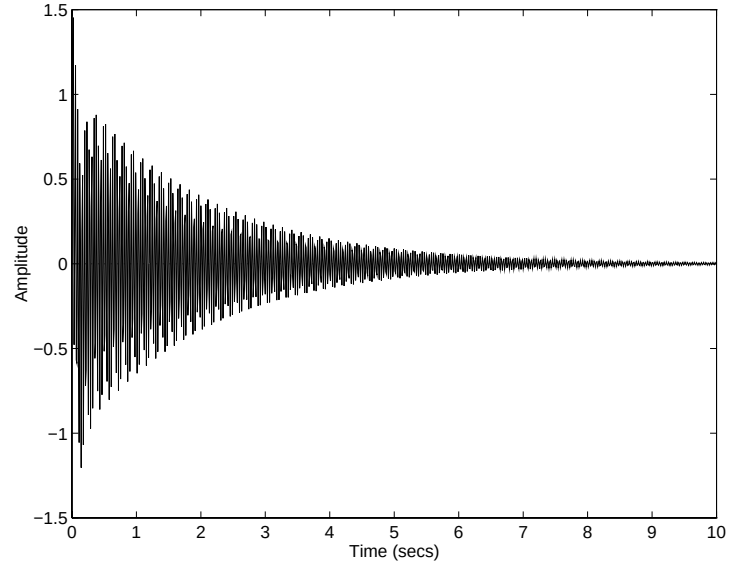


Figure 6-11: Response *y* for an impulse disturbance at the plant input

For "Example 6.1" on page 6-5, we chose as pole placement region the disk centered at (–1000, 0) and with radius 1000. By inspection of the shaping filters used in the previous design, this region clashes with the pseudo-integrator in $W_1(s)$ and the fast pole at $s = -3.54 \times 10^4$ in $W_2(s)$. To alleviate this difficulty,

1. Remove the pseudo-integrator from $W_1(s)$ and move it to the main loop as shown in Figure 6-12. This defines an equivalent loop-shaping problem where the controller is reparametrized as $K(s) = \tilde{K}(s)/s$ and the integrator is now assignable via output feedback.

2 Specify $W_2(s)$ as a nonproper filter with derivative action. This is done with `sderiv`.

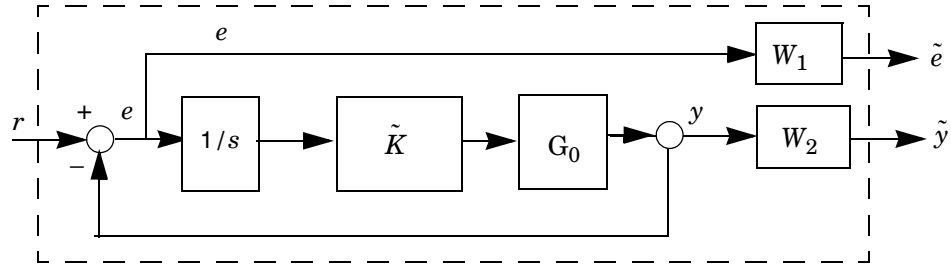


Figure 6-12: Modified control structure

A satisfactory design was obtained for the following values of W_1 and W_2:

$$W_1(s) = \frac{s^2 + 16s + 200}{s^2 + 1.2s + 0.8}, \qquad W_2(s) = 0.9 + \frac{s}{200}$$

A summary of the commands needed to specify the control structure and solve the constrained H_∞ problem is listed below. Run the demo `radardem` to perform this design interactively.

```
% control structure, sint = integrator
sint = ltisys('tf',1,[1 0])
[P,r] = sconnect('r','Int;G','Kt:Int','G:Kt',GO,'Int:e=r-G',
sint)

% add w1
w1=ltisys('tf',[1 16 200],[1 1.2 0.8])
Paug = smult(P,sdiag(w1,1,1))

% add w2
Paug = sderiv(Paug,2,[1/200 0.9])

% interactively specify the region for pole placement
region = lmireg

% perform the H-infinity synthesis with pole placement
r = [0 1 1]
```

```
[gopt,xx,Kt] = hinfmix(Paug,r,[0 0 1 0],region)
```

The full controller is then given by

```
K = smult(sint,Kt)
```

The resulting open-loop response and the time response to a step r and an impulse disturbance at the plant input appear in Figure 6-13 and 6-14. Note that the new controller no longer inverts the plant and that input disturbances are now rejected with satisfactory transient behavior.

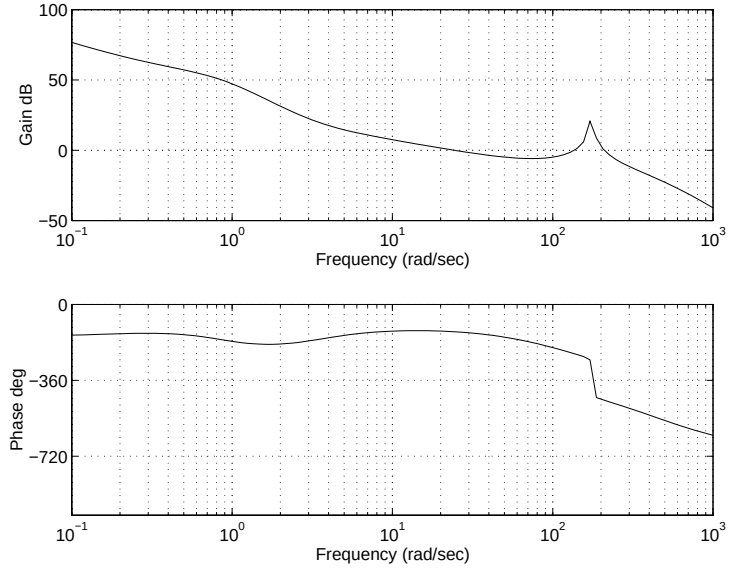


Figure 6-13: Open-loop response *GK(s)*

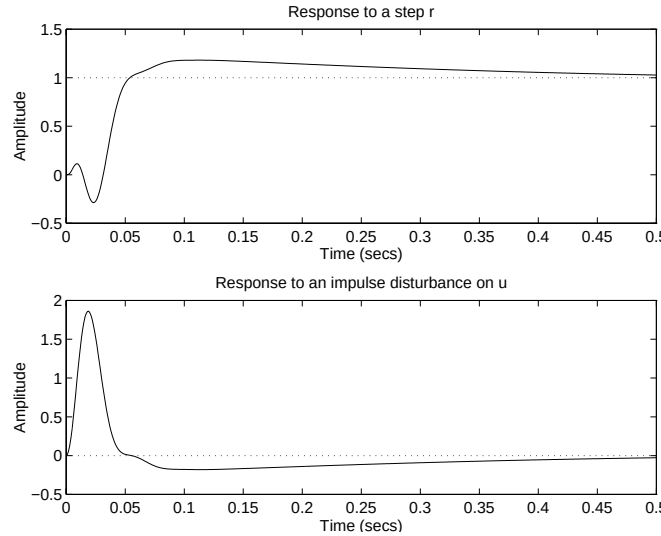


Figure 6-14: Transient behaviors with the new controller

References

[1] Doyle, J.C., Glover, K., Khargonekar, P., and Francis, B., "State-Space Solutions to Standard H_2 and H_∞ Control Problems," *IEEE Trans. Aut. Contr.*, AC–34 (1989), pp. 831–847.

[2] Zames, G., "On the Input-Output Stability of Time-Varying Nonlinear Feedback Systems, Part I and II," *IEEE Trans. Aut. Contr.*, AC–11 (1966), pp. 228–238 and 465–476.

7

Robust Gain-Scheduled Controllers

Gain-Scheduled Control 7-3

Synthesis of Gain-Scheduled H_∞ Controllers 7-7

Simulation of Gain-Scheduled Control Systems 7-9

Design Example 7-10

References . . . . . .

Gain scheduling is a widely used technique for controlling certain classes of nonlinear or linear time-varying systems. Rather than seeking a single robust LTI controller for the entire operating range, gain scheduling consists in designing an LTI controller for each operating point and in switching controller when the operating conditions change.

This section presents systematic tools to design gain-scheduled H_∞ controllers for linear parameter-dependent systems. These tools are applicable to time-varying and/or nonlinear systems whose linearized dynamics are reasonably well approximated by affine parameter-dependent models.

Gain-Scheduled Control

The synthesis technique discussed below is applicable to affine parameter-dependent plants with equations

$$P(.,\mathrm{p})\begin{cases} \dot{x} &= A(p)x + B_1(p)w + B_2u \\ z &= C_1(p)x + D_{11}(p)w + D_{12}u \\ y &= C_2x + D_{21}w + D_{22}u \end{cases}$$

where

$$p(t) = (p_1(t), \ldots, p_n(t)), \qquad \underline{p}i \le pi(t) \le \bar{p}i$$

is a time-varying vector of physical parameters (velocity, angle of attack, stiffness,. . .) and $A(\cdot)$, $B_1(\cdot)$, $C_1(\cdot)$, $D_{11}(\cdot)$ are affine functions of $p(t)$. This is a simple model of systems whose dynamical equations depend on physical coefficients that vary during operation. When these coefficients undergo large variations, it is often impossible to achieve high performance over the entire operating range with a single robust LTI controller. Provided that the parameter values are measured in real time, it is then desirable to use controllers that incorporate such measurements to adjust to the current operating conditions [4]. Such controllers are said to be scheduled by the parameter measurements. This control strategy typically achieves higher performance in the face of large variations in operating conditions. Note that $p(t)$ may include part of the state vector x itself provided that the corresponding states are accessible to measurement.

If the parameter vector $p(t)$ takes values in a box of $\mathbf{R}^n$ with corners $\{\Pi_i\}_{i=1}^{N}(N = 2^n)$, the plant `SYSTEM` matrix

$$S(p) := \left(\begin{array}{c|cc} A(p) & B_1(p) & B_2 \\ \hline C_1(p) & D_{11}(p) & D_{12} \\ C_2 & D_{21} & D_{22} \end{array}\right)$$

ranges in a matrix polytope with vertices $S(\Pi_i)$. Specifically, given any convex decomposition

$$p(t) = \alpha_1\Pi_1 + \ldots + \alpha_N\Pi_N, \qquad \alpha_i \geq 0, \qquad \sum_{i=1}^{N} \alpha_i = 1$$

of p over the corners of the parameter box, the `SYSTEM` matrix $S(p)$ is given by

$$S(p) = \alpha_1 S(\Pi_1) + \ldots + \alpha_N S(\Pi_N)$$

This suggests seeking parameter-dependent controllers with equations

$$K(.,\mathrm{p})\begin{cases} \zeta = A_K(p)\zeta + B_K(p)y \\ u = C_K(p)\zeta + D_K(p)y \end{cases}$$

and having the following *vertex property*:

> Given the convex decomposition $p(t) = \sum_{i=1}^{N} \alpha_i\Pi_i$ of the current parameter value $p(t)$, the values of $A_K(p)$, $B_K(p)$, . . . are derived from the values $A_K(\Pi_i)$, $B_K(\Pi_i)$, . . . at the corners of the parameter box by
>
> $$\begin{pmatrix} A_K(p) & B_K(p) \\ C_K(p) & D_K(p) \end{pmatrix} = \sum_{i=1}^{N} \alpha_i \begin{pmatrix} A_K(\Pi_i) & B_K(\Pi_i) \\ C_K(\Pi_i) & D_K(\Pi_i) \end{pmatrix}$$

In other words, the controller state-space matrices at the operating point $p(t)$ are obtained by convex interpolation of the LTI *vertex controllers*

$$K_i := \begin{pmatrix} A_K(\Pi_i) & B_K(\Pi_i) \\ C_K(\Pi_i) & D_K(\Pi_i) \end{pmatrix}$$

This yields a smooth scheduling of the controller matrices by the parameter measurements $p(t)$.

For this class of controllers, consider the following H_∞-like synthesis problem relative to the interconnection of Figure 7-1:

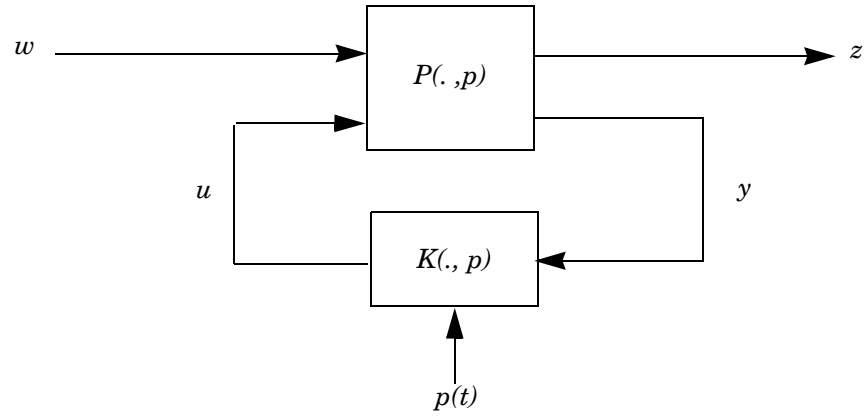


Figure 7-1: Gain-scheduled H_∞ problem

Design a gain-scheduled controller $K(., p)$ satisfying the vertex property and such that

- The closed-loop system is stable for all admissible parameter trajectories $p(t)$
- The worst-case closed-loop RMS gain from w to z does not exceed some level $\gamma > 0$.

Using the notion of quadratic H_∞ performance to enforce the RMS gain constraint, and observing that the closed-loop system is a polytopic system, this synthesis problem can be reduced to the following LMI problem [3, 2]

Find two symmetric matrices R and S such that

$$\begin{pmatrix} N_{12} & 0 \\ 0 & I \end{pmatrix}^T \begin{pmatrix} A_i R + R A_i^T & R C_{1i}^T & B_{1i} \\ C_{1i} R & -\gamma I & D_{11i} \\ B_1^T & D_{11i}^T & -\gamma I \end{pmatrix} \begin{pmatrix} N_{12} & 0 \\ 0 & I \end{pmatrix} < 0, \quad i = 1, \ldots, N \tag{7-1}$$

$$\begin{pmatrix} N_{21} & 0 \\ 0 & I \end{pmatrix}^T \begin{pmatrix} A_i^T S + SA & SB_{1i} & C_{1i}^T \\ B_{1i}^T S & -\gamma I & D_{11i}^T \\ C_{1i} & D_{11i} & -\gamma I \end{pmatrix} \begin{pmatrix} N_{21} & 0 \\ 0 & I \end{pmatrix} < 0, \quad i = 1, \ldots, N \tag{7-2}$$

$$\begin{pmatrix} R & I \\ I & S \end{pmatrix} \geq 0 \tag{7-3}$$

where

$$\begin{pmatrix} A_i & B_{1i} \\ C_{1i} & D_{11i} \end{pmatrix} := \begin{pmatrix} A(\Pi_i) & B_1(\Pi_i) \\ C_1(\Pi_i) & D_{11}(\Pi_i) \end{pmatrix}$$

and N_{12} and N_{21} are bases of the null spaces of $((B_2^T, D_{12}^T))$ and (C_2, D_{21}), respectively.

Remark: Requiring the plant matrices B_2, C_2, D_{12}, D_{21} to be independent of $p(t)$ may be restrictive in practice. When B_2 and D_{12} depend on $p(t)$, a simple trick to enforce these requirements consists of filtering u through a low-pass filter with sufficiently large bandwidth. Appending this filter to the plant rejects the parameter dependence of B_2 and D_{12} into the A matrix of the augmented plant. Similarly, parameter dependences in C_2 and D_{21} can be eliminated by filtering the output (see [1] for details).

Synthesis of Gain-Scheduled H_∞ Controllers

The synthesis of gain-scheduled H_∞ controllers is performed by the function `hinfgs`. Since such controllers are characterized by their vertex values, both the plant and the controller are treated as polytopic models by `hinfgs`. The calling sequence is

```
[gopt,pdK] = hinfgs(pdP,r)
```

Here `pdP` is an affine or polytopic model of the plant (see `psys`) and the two-entry vector `r` specifies the dimensions of the D_{22} matrix (set `r=[p2 m2]` if $y \in \mathbf{R}^{p_2}$ and $u \in \mathbf{R}^{m_2}$).

On output, `gopt` is the best achievable RMS gain and `pdK` is the polytopic system consisting of the vertex controllers.

$$K_i = \begin{pmatrix} A_K(\Pi_i) & B_K(\Pi_i) \\ C_K(\Pi_i) & D_K(\Pi_i) \end{pmatrix} \qquad K_i = \begin{pmatrix} A_K(\Pi_i) & B_K(\Pi_i) \\ C_K(\Pi_i) & D_K(\Pi_i) \end{pmatrix}$$

Given any value `p` of the parameter vector $p(t)$, the corresponding state-space parameters

$$K(p) = \begin{pmatrix} A_K(p) & B_K(p) \\ C_K(p) & D_K(p) \end{pmatrix}$$

of the gain-scheduled controller are returned in `SYSTEM` matrix format by

```
c = polydec(pv,p)
Kp = psinfo(pdK,'eval',c)
```

The function `polydec` computes the convex decomposition $p(t) = \sum_{i=1}^{N} \alpha_i \Pi_i$ and returns `c` $= (\alpha_1, \ldots, \alpha_N)$. The controller state-space data $K(p) = \sum_{i=1}^{N} \alpha_i K_i$ at time t is then computed by `psinfo`. Note that `polydec` ensures the continuity of the polytopic coordinates α_i as a function of p, thereby guaranteeing the continuity of the controller state-space matrices along parameter trajectories.

Given the polytopic controller representation `pdK`, a polytopic model of the closed-loop system is formed by

```
pcl = slft(pdP,pdK)
```

and evaluated for a given parameter value `p` of $p(t)$ by

```
pclt = psinfo(pcl,'eval',polydec(pv,p))
```

Simulation of Gain-Scheduled Control Systems

Being time-varying systems, gain-scheduled control systems cannot be simulated with `splot`. The function `pdsimul` is provided to simulate the time response of such systems along a given parameter trajectory.

With `pdsimul`, you must first define the parameter trajectory and the input signal as two functions of time with syntax `p=trajfun(t)` and `u=inputfun(t)`. You can then plot the corresponding time response by

```
pdsimul(pcl,'trajfun',tf,'inputfun')
```

where `pcl` is the polytopic representation of the closed-loop system and `tf` is the desired duration of the simulation. If `'inputfun'` is omitted, the step response is plotted by default. See the next paragraph for an example.

Design Example

This example is drawn from [2] and deals with the design of a gain-scheduled autopilot for the pitch axis of a missile. Type `misldem` to run the corresponding demo.

The missile dynamics are highly dependent on the angle of attack α, the speed V, and the altitude H. These three parameters undergo large variations during operation. A simple model of the linearized dynamics of the missile is the parameter-dependent model

$$G\begin{cases}\begin{pmatrix}\dot{\alpha}\\ \dot{q}\end{pmatrix} = \begin{pmatrix}-Z\alpha & 1\\ -M\alpha & 0\end{pmatrix}\begin{pmatrix}\alpha\\ q\end{pmatrix} + \begin{pmatrix}0\\ 1\end{pmatrix}\delta_m\\ \begin{pmatrix}a_{zv}\\ q\end{pmatrix} = \begin{pmatrix}-1 & 0\\ 0 & 1\end{pmatrix}\begin{pmatrix}\alpha\\ q\end{pmatrix}\end{cases} \qquad \textbf{(7-4)}$$

where a_{zv} is the normalized vertical acceleration, q the pitch rate, δm the fin deflection, and Z_α, M_α are aerodynamical coefficients depending on α, V, and H. These two coefficients are "measured" in real time.

As V, H, and α vary in

$$V \in [0.5, 4] \text{ Mach}, \qquad H \in [0, 18000] \text{ m}, \qquad \alpha \in [0, 40] \text{ degrees}$$

during operation, the coefficients Z_α and M_α range in

$$Z_\alpha \in [0.5, 4], \qquad M_\alpha \in [0, 106].$$

Our goal is to control the vertical acceleration a_{zv} over this operating range. The stringent performance specifications (settling time < 0.5 s) and the bandwidth limitation imposed by unmodeled high frequency dynamics make gain scheduling desirable in this context.

The control structure is sketched in Figure 7-2. To enforce the performance and robustness requirements, we can use the loop-shaping criterion

$$\left\| \begin{matrix} W_1 S \\ W_2 K S \end{matrix} \right\|_\infty < 1 \qquad \textbf{(7-5)}$$

Note that $S = (I + GK)^{-1}$ is now a time-varying system and that the H_∞ norm must be understood in terms of input/output RMS gain. Adequate shaping filters are derived with `magshape` as

$$W_1(s) = \frac{2.01}{s + 0.201}$$

$$W_2(s) = \frac{9.678s^3 + 0.029s^2}{s^3 + 1.206 \times 10^4 s^2 + 1.136 \times 10^7 s + 1.066 \times 10^{10}}$$

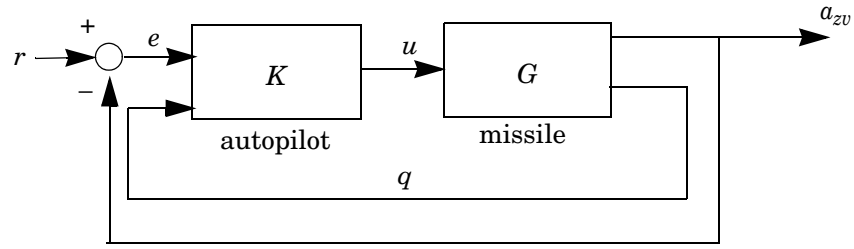


Figure 7-2: Missile autopilot

The gain-scheduled controller synthesis parallels the standard loop-shaping procedure described in Chapter 6. First enter the parameter-dependent model (7-4) of the missile pitch axis by:

```
% parameter range:

Zmin=.5; Zmax=4; Mmin=0; Mmax=106;
pv = pvec('box',[Zmin Zmax ; Mmin Mmax])

% affine model:

s0 = ltisys([0 1;0 0],[0;1],[-1 0;0 1],[0;0])
s1 = ltisys([-1 0;0 0],[0;0],zeros(2),[0;0],0) % Z_al
s2 = ltisys([0 0;-1 0],[0;0],zeros(2),[0;0],0) % M_al
pdG = psys(pv,[s0 s1 s2])
```

For loop-shaping purposes, we must form the augmented plant associated with the criterion (7-5). This is done with `sconnect` and `smult` as follows:

```
% form the plant interconnection

[pdP,r] = sconnect('r','e=r-GP1;K','K:e;G(2)','G:K',pdG);

% append the shaping filters

Paug = smult(pdP,sdiag(w1,w2,eye(2)))
```

Note that `pdP` and `Paug` are polytopic models at this stage.

We are now ready to perform the gain-scheduled controller synthesis with `hinfgs`:

```
[gopt,pdK] = hinfgs(Paug,r)

gopt =

   0.205
```

Our specifications are achievable since the best performance $\gamma_{opt} = 0.205$ is smaller than 1, and the polytopic description of a controller with such performance is returned in `pdK`.

To validate the design, form the closed-loop system with

```
pCL = slft(pdP,pdK)
```

You can now simulate the step response of the gain-scheduled system along particular parameter trajectories. For the sake of illustration, consider the spiral trajectory

$$Z_\alpha(t) = 2.25 + 1.70\, e^{-4t} \cos(100\, t)$$

$$M_\alpha(t) = 50 + 49\, e^{-4t} \sin(100\, t)$$

shown in Figure 7-3. After defining this trajectory is a function `spiralt.m`:

```
function p = spiral(t)

p = [2.25 + 1.70*exp(-4*t).*cos(100*t) ; ...
50 + 49*exp(-4*t).*sin(100*t)];
```

you can plot the closed-loop step response along $p(t)$ by

```
[t,x,y]=pdsimul(pCL,'spiralt',0.5)

plot(t,1-y(:,1))
```

The function `pdsimul` returns the output trajectories $y(t) = (\mathrm{e}(t), u(t))^T$ and the response $a_{zv}(t)$ is deduced by

$$a_{zv}(t) = 1 - \mathrm{e}(t)$$

The second command plots $a_{zv}(t)$ versus time as shown in Figure 7-4.

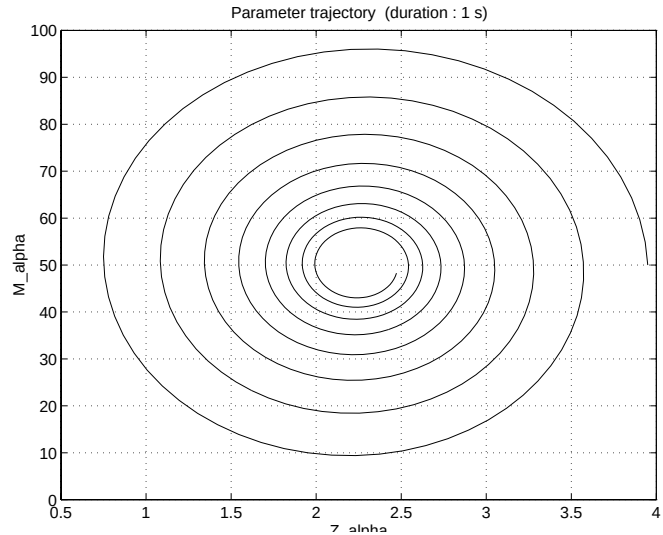


Figure 7-3: Parameter trajectory

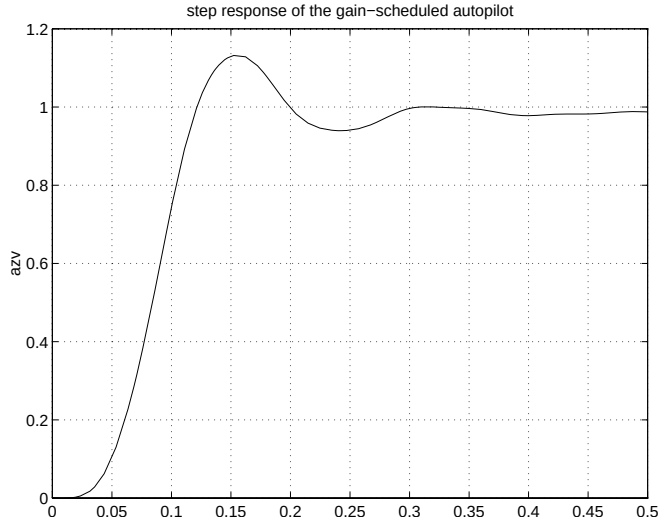


Figure 7-4: Corresponding step response

References

[1] Apkarian, P., and P. Gahinet,"A Convex Characterization of Gain-Scheduled H_∞ Controllers," to appear in *IEEE Trans. Aut. Contr*., 1995.

[2] Apkarian, P., P. Gahinet, and G. Becker, "Self-Scheduled H_∞ Control of Linear Parameter-Varying Systems," to appear in *Automatica*, 1995.

[3] Becker, G., Packard, P., "Robust Performance of Linear Parametrically Varying Systems Using Parametrically Dependent Linear Feedback," *Systems and Control Letters,* 23 (1994), pp. 205–215.

[4] Packard, A., "Gain Scheduling via Linear Fractional Transformations," *Syst. Contr. Letters*, 22 (1994), pp. 79–92.

8

The LMI Lab

Background and Terminology 8-3

Overview of the LMI Lab 8-6

Specifying a System of LMIs 8-8

Retrieving Information 8-21

LMI Solvers 8-22

From Decision to Matrix Variables and Vice Versa . . 8-28

Validating Results 8-29

Modifying a System of LMIs 8-30

Advanced Topics 8-33

References 8-44

The LMI Lab is a high-performance package for solving general LMI problems. It blends simple tools for the specification and manipulation of LMIs with powerful LMI solvers for three generic LMI problems. Thanks to a structure-oriented representation of LMIs, the various LMI constraints can be described in their natural block-matrix form. Similarly, the optimization variables are specified directly as *matrix variables* with some given structure. Once an LMI problem is specified, it can be solved numerically by calling the appropriate LMI solver. The three solvers `feasp`, `mincx`, and `gevp` constitute the computational engine of the LMI Control Toolbox. Their high performance is achieved through C-MEX implementation and by taking advantage of the particular structure of each LMI.

The LMI Lab offers tools to

- Specify LMI systems either symbolically with the LMI Editor or incrementally with the `lmivar` and `lmiterm` commands
- Retrieve information about existing systems of LMIs
- Modify existing systems of LMIs
- Solve the three generic LMI problems (feasibility problem, linear objective minimization, and generalized eigenvalue minimization)
- Validate results

This chapter gives a tutorial introduction to the LMI Lab as well as more advanced tips for making the most out of its potential. The tutorial material is also covered by the demo `lmidem`.

Background and Terminology

Any linear matrix inequality can be expressed in the canonical form

$$L(x) = L_0 + x_1 L_1 + \ldots + x_N L_N < 0$$

where

- $L_0, L_1, \ldots, L_N$ are given symmetric matrices
- $x = (x_1, \ldots, x_N)^T \in \mathbf{R}^N$ is the vector of scalar variables to be determined. We refer to $x_1, \ldots, x_N$ as the *decision variables*. The names "design variables" and "optimization variables" are also found in the literature.

Even though this canonical expression is generic, LMIs rarely arise in this form in control applications. Consider for instance the Lyapunov inequality

$$A^T X + XA < 0 \tag{8-1}$$

where $A = \begin{pmatrix} -1 & 2 \\ 0 & -2 \end{pmatrix}$ and the variable $X = \begin{pmatrix} x_1 & x_2 \\ x_2 & x_3 \end{pmatrix}$ is a symmetric matrix.

Here the decision variables are the free entries x_1, x_2, x_3 of X and the canonical form of this LMI reads

$$x_1 \begin{pmatrix} -2 & 2 \\ 2 & 0 \end{pmatrix} + x_2 \begin{pmatrix} 0 & -3 \\ -3 & 4 \end{pmatrix} + x_3 \begin{pmatrix} 0 & 0 \\ 0 & -4 \end{pmatrix} < 0 \tag{8-2}$$

Clearly this expression is less intuitive and transparent than (8-1). Moreover, the number of matrices involved in (8-2) grows roughly as $n^2/2$ if n is the size of the A matrix. Hence, the canonical form is very inefficient from a storage viewpoint since it requires storing $o(n^2/2)$ matrices of size n when the single n-by-n matrix A would be sufficient. Finally, working with the canonical form is also detrimental to the efficiency of the LMI solvers. For these various reasons, the LMI Lab uses a *structured representation* of LMIs. For instance, the expression $A^TX + XA$ in the Lyapunov inequality (8-1) is explicitly described as a function of the matrix variable X, and only the A matrix is stored.

In general, LMIs assume a block matrix form where each block is an affine combination of the matrix variables. As a fairly typical illustration, consider the following LMI drawn from H_∞ theory

$$N^T \begin{pmatrix} A^T X + XA & XC^T & B \\ CX & -\gamma I & D \\ B^T & D^T & -\gamma I \end{pmatrix} N < 0 \qquad \textbf{(8-3)}$$

where A, B, C, D, N are given matrices and the problem variables are $X = X^T \in \mathbf{R}^{n \times n}$ and $\gamma \in \mathbf{R}$. We use the following terminology to describe such LMIs:

- N is called the *outer factor*, and the block matrix

$$L(X, \gamma) = \begin{pmatrix} A^T X + XA & XC^T & B \\ CX & -\gamma I & D \\ B^T & D^T & -\gamma I \end{pmatrix}$$

  is called the *inner factor*. The outer factor *needs not be square* and is *often absent*.
- X and γ are the *matrix variables* of the problem. Note that scalars are considered as 1-by-1 matrices.
- The inner factor $L(X, \gamma)$ is a symmetric *block matrix*, its block structure being characterized by the sizes of its diagonal blocks. By symmetry, $L(X, \gamma)$ is entirely specified by the blocks on or above the diagonal.
- Each block of $L(X, \gamma)$ is an affine expression in the matrix variables X and γ. This expression can be broken down into a sum of elementaryterm *terms*. For instance, the block (1,1) contains two elementary terms: $A^T X$ and XA.
- Terms are either *constant* or *variable*. Constant terms are fixed matrices like B and D above. Variable terms involve one of the matrix variables, like XA, XC^T, and $-\gamma I$ above.

The LMI (8-3) is specified by the list of terms in each block, as is any LMI regardless of its complexity.

As for the matrix variables X and γ, they are characterized by their dimensions and structure. Common structures include rectangular unstructured, symmetric, skew-symmetric, and scalar. More sophisticated structures are sometimes encountered in control problems. For instance, the matrix variable X could be constrained to the block-diagonal structure

$$X = \left(\begin{array}{c|cc} x_1 & 0 & 0 \\ \hline 0 & x_2 & x_3 \\ 0 & x_3 & x_4 \end{array}\right)$$

Another possibility is the symmetric Toeplitz structure

$$X = \begin{pmatrix} x_1 & x_2 & x_3 \\ x_2 & x_1 & x_2 \\ x_3 & x_2 & x_1 \end{pmatrix}$$

Summing up, structured LMI problems are specified by declaring the matrix variables and describing the term content of each LMI. This term-oriented description is systematic and accurately reflects the specific structure of the LMI constraints. There is no built-in limitation on the number of LMIs that can be specified or on the number of blocks and terms in any given LMI. LMI systems of arbitrary complexity can therefore, be defined in the LMI Lab.

Overview of the LMI Lab

The LMI Lab offers tools to specify, manipulate, and numerically solve LMIs. Its main purpose is to

- Allow for straightforward description of LMIs in their natural block-matrix form
- Provide easy access to the LMI solvers (optimization codes)
- Facilitate result validation and problem modification

The structure-oriented description of a given LMI system is stored as a single vector called the *internal representation* and generically denoted by `LMISYS` in the sequel. This vector encodes the structure and dimensions of the LMIs and matrix variables, a description of all LMI terms, and the related numerical data. It must be stressed that users need not attempt to read or understand the content of `LMISYS` since all manipulations involving this internal representation can be performed in a transparent manner with LMI-Lab tools.

The LMI Lab supports the following functionalities:

Specification of a System of LMIs

LMI systems can be either specified as symbolic matrix expressions with the interactive graphical user interface `lmiedit`, or assembled incrementally with the two commands `lmivar` and `lmiterm`. The first option is more intuitive and transparent while the second option is more powerful and flexible.

Information Retrieval

The interactive function `lmiinfo` answers qualitative queries about LMI systems created with `lmiedit` or `lmivar` and `lmiterm`. You can also use `lmiedit` to visualize the LMI system produced by a particular sequence of `lmivar`/`lmiterm` commands.

Solvers for LMI Optimization Problems

General-purpose LMI solvers are provided for the three generic LMI problems defined on page 1-5. These solvers can handle very general LMI systems and matrix variable structures. They return a feasible or optimal vector of decision variables x^*. The corresponding values $X_1^*, \ldots, X_K^*$ of the matrix variables are given by the function `dec2mat`.

Result Validation

The solution x^* produced by the LMI solvers is easily validated with the functions `evallmi` and `showlmi`. This allows a fast check and/or analysis of the results. With `evallmi`, all variable terms in the LMI system are evaluated for the value x^* of the decision variables. The left- and right-hand sides of each LMI then become constant matrices that can be displayed with `showlmi`.

Modification of a System of LMIs

An existing system of LMIs can be modified in two ways:

- An LMI can be removed from the system with `dellmi`.
- A matrix variable X can be deleted using `delmvar`. It can also be instantiated, that is, set to some given matrix value. This operation is performed by `setmvar` and allows, for example, to fix some variables and solve the LMI problem with respect to the remaining ones.

Specifying a System of LMIs

The LMI Lab can handle any system of LMIs of the form

$$N^T L(X_1, \ldots, X_K)\, N < MT\, R(X_1, \ldots, X_K)\, M$$

where

- $X_1, \ldots, X_K$ are matrix variables with some prescribed structure
- The left and right outer factors N and M are given matrices with *identical* dimensions
- The left and right inner factors $L(\cdot)$ and $R(\cdot)$ are symmetric block matrices with identical block structures, each block being an affine combination of X_1, $\ldots$, X_K and their transposes.

Important: Throughout this chapter, "left-hand side" refers to what is on the "smaller" side of the inequality, and "right-hand side" to what is on the "larger" side. Accordingly, X is called the right-hand side and 0 the left-hand side of the LMI $0 < X$ even when this LMI is written as $X > 0$.

The specification of an LMI system involves two steps:

1 Declare the dimensions and structure of each matrix variable $X_1, \ldots, X_K$.

2 Describe the term content of each LMI.

This process creates the so-called *internal representation* of the LMI system. This computer description of the problem is used by the LMI solvers and in all subsequent manipulations of the LMI system. It is stored as a single vector which we consistently denote by `LMISYS` in the sequel (for the sake of clarity).

There are two ways of generating the internal description of a given LMI system: (1) by a sequence of `lmivar`/`lmiterm` commands that build it incrementally, or (2) via the LMI Editor `lmiedit` where LMIs can be specified directly as symbolic matrix expressions. Though somewhat less flexible and powerful than the command-based description, the LMI Editor is more straightforward to use, hence particularly well-suited for beginners. Thanks to its coding and decoding capabilities, it also constitutes a good tutorial introduction to `lmivar` and `lmiterm`. Accordingly, beginners may elect to skip

the subsections on `lmivar` and `lmiterm` and to concentrate on the GUI-based specification of LMIs with `lmiedit`.

A Simple Example

The following tutorial example is used to illustrate the specification of LMI systems with the LMI Lab tools. Run the demo `lmidem` to see a complete treatment of this example.

Example 8.1. Consider a stable transfer function

$$G(s) = C(sI - A)^{-1}B \tag{8-4}$$

with four inputs, four outputs, and six states, and consider the set of input/output scaling matrices D with block-diagonal structure

$$D = \begin{pmatrix} d_1 & 0 & 0 & 0 \\ 0 & d_1 & 0 & 0 \\ 0 & 0 & d_2 & d_3 \\ 0 & 0 & d_4 & d_5 \end{pmatrix} \tag{8-5}$$

The following problem arises in the robust stability analysis of systems with time-varying uncertainty [4]:

> Find, if any, a scaling D of structure (8-5) such that the largest gain across frequency of $D\,G(s)\,D^{-1}$ is less than one.

This problem has a simple LMI formulation: there exists an adequate scaling D if the following feasibility problem has solutions:

Find two symmetric matrices $X \in \mathbf{R}^{6\times 6}$ and $S = \mathrm{D}^{\mathrm{T}} D \in \mathbf{R}^{4\times 4}$ such that

$$\begin{pmatrix} A^T X + XA + C^T SC & XB \\ B^T X & -S \end{pmatrix} < 0 \tag{8-6}$$

$$X > 0 \tag{8-7}$$

$$S > 1 \tag{8-8}$$

The LMI system (8-6)–(8-8) can be described with the LMI Editor as outlined below. Alternatively, its internal description can be generated with `lmivar` and `lmiterm` commands as follows:

```
setlmis([])
X=lmivar(1,[6 1])
S=lmivar(1,[2 0;2 1])

% 1st LMI
lmiterm([1 1 1 X],1,A,'s')
lmiterm([1 1 1 S],C',C)
lmiterm([1 1 2 X],1,B)
lmiterm([1 2 2 S], 1,1)

% 2nd LMI
lmiterm([ 2 1 1 X],1,1)

% 3rd LMI
lmiterm([ 3 1 1 S],1,1)
lmiterm([3 1 1 0],1)

LMISYS = getlmis
```

Here the `lmivar` commands define the two matrix variables X and S while the `lmiterm` commands describe the various terms in each LMI. Upon completion,

`getlmis` returns the internal representation `LMISYS` of this LMI system. The next three subsections give more details on the syntax and usage of these various commands. More information on how the internal representation is updated by `lmivar`/`lmiterm` can also be found in "How It All Works" on page 8-18.

setlmis and getlmis

The description of an LMI system should begin with `setlmis` and end with `getlmis`. The function `setlmis` initializes the LMI system description. When specifying a new system, type

```
setlmis([])
```

To add on to an existing LMI system with internal representation `LMISO`, type

```
setlmis(LMISO)
```

When the LMI system is completely specified, type

```
LMISYS = getlmis
```

This returns the internal representation `LMISYS` of this LMI system. This MATLAB description of the problem can be forwarded to other LMI-Lab functions for subsequent processing. The command `getlmis` must be used *only once* and after declaring *all* matrix variables and LMI terms.

lmivar

The matrix variables are declared one at a time with `lmivar` and are characterized by their structure. To facilitate the specification of this structure, the LMI Lab offers two predefined structure types along with the means to describe more general structures:

Type 1: **Symmetric block diagonal** structure. This corresponds to matrix variables of the form

$$X = \begin{pmatrix} D_1 & 0 & \dots & 0 \\ 0 & D_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & D_r \end{pmatrix}$$

where each diagonal block D_j is square and is either zero, a *full symmetric* matrix, or a *scalar* matrix

$$D_j = d \times I, \qquad d \in \mathbf{R}$$

This type encompasses ordinary symmetric matrices (single block) and scalar variables (one block of size one).

Type 2: **Rectangular** structure. This corresponds to arbitrary rectangular matrices without any particular structure.

Type 3: **General** structures. This third type is used to describe more sophisticated structures and/or correlations between the matrix variables. The principle is as follows: each entry of X is specified independently as either 0, x_n, or $-x_n$ where x_n denotes the n-th decision variable in the problem. For details on how to use Type 3, see "Structured Matrix Variables" on page 8-33 below as well as the `lmivar` entry of the "Command Reference" chapter.

In "Example 8.1" on page 8-9, the matrix variables X and S are of Type 1. Indeed, both are symmetric and S inherits the block-diagonal structure (8-5) of D. Specifically, S is of the form

$$S = \begin{pmatrix} s_1 & 0 & 0 & 0 \\ 0 & s_1 & 0 & 0 \\ 0 & 0 & s_2 & s_3 \\ 0 & 0 & s_3 & s_4 \end{pmatrix}$$

After initializing the description with the command `setlmis([])`, these two matrix variables are declared by

```
lmivar(1,[6 1]) % X
lmivar(1,[2 0;2 1]) % S
```

In both commands, the first input specifies the structure type and the second input contains additional information about the structure of the variable:

- For a matrix variable X of Type 1, this second input is a matrix with two columns and as many rows as diagonal blocks in X. The first column lists the

sizes of the diagonal blocks and the second column specifies their nature with the following convention:

1 → full symmetric block

0 → scalar block

–1 → zero block

In the second command, for instance, `[2 0;2 1]` means that S has two diagonal blocks, the first one being a 2-by-2 scalar block and the second one a 2–βψ–2 full block.

- For matrix variables of Type 2, the second input of `lmivar` is a two-entry vector listing the row and column dimensions of the variable. For instance, a 3-by-5 rectangular matrix variable would be defined by

```
lmivar(2,[3 5])
```

For convenience, `lmivar` also returns a "tag" that identifies the matrix variable for subsequent reference. For instance, X and S in "Example 8.1" could be defined by

```
X = lmivar(1,[6 1])
S = lmivar(1,[2 0;2 1])
```

The identifiers X and S are integers corresponding to the ranking of X and S in the list of matrix variables (in the order of declaration). Here their values would be `X=1` and `S=2`. Note that these identifiers still point to X and S after deletion or instantiation of some of the matrix variables. Finally, `lmivar` can also return the total number of decision variables allocated so far as well as the entry-wise dependence of the matrix variable on these decision variables (see the `lmivar` entry of the "Command Reference" chapter for more details).

lmiterm

After declaring the matrix variables with `lmivar`, we are left with specifying the term content of each LMI. Recall that LMI terms fall into three categories:

- The *constant terms*, i.e., fixed matrices like I in the left-hand side of the LMI $S > I$
- The *variable terms*, i.e., terms involving a matrix variable. For instance, A^TX and C^TSC in (8-6). Variable terms are of the form PXQ where X is a variable and P, Q are given matrices called the left and right *coefficients*, respectively.

- The *outer factors*

The following rule should be kept in mind when describing the term content of an LMI:

Important: Specify only the terms in the blocks on or above the diagonal. The inner factors being symmetric, this is sufficient to specify the entire LMI. *Specifying all blocks results in the duplication of off-diagonal terms, hence in the creation of a different LMI.* Alternatively, you can describe the blocks on or below the diagonal.

LMI terms are specified one at a time with `lmiterm`. For instance, the LMI

$$\begin{pmatrix} A^TX + XA + C^TSC & XB \\ B^TX & -S \end{pmatrix} < 0$$

is described by

```
lmiterm([1 1 1 1],1,A,'s')
lmiterm([1 1 1 2],C',C)
lmiterm([1 1 2 1],1,B)
lmiterm([1 2 2 2], 1,1)
```

These commands successively declare the terms $A^TX + XA$, C^TSC, XB, and $-S$. In each command, the first argument is a four-entry vector listing the term characteristics as follows:

- The first entry indicates to which LMI the term belongs. The value `m` means "left-hand side of the m-th LMI," and `-m` means "right-hand side of the m-th LMI"
- The second and third entries identify the block to which the term belongs. For instance, the vector `[1 1 2 1]` indicates that the term is attached to the (1, 2) block
- The last entry indicates which matrix variable is involved in the term. This entry is `0` for constant terms, `k` for terms involving the k-th matrix variable

X_k , and `k` for terms involving X_k^T (here X and S are first and second variables in the order of declaration).

Finally, the second and third arguments of `lmiterm` contain the numerical data (values of the constant term, outer factor, or matrix coefficients P and Q for variable terms PXQ or PX^TQ). These arguments must refer to existing MATLAB variables and be *real-valued*. See"Complex-Valued LMIs" on page 8-35 for the specification of LMIs with complex-valued coefficients.

Some shorthand is provided to simplify term specification. First, blocks are zero by default. Second, in *diagonal blocks* the extra argument `'s'` allows you to specify the conjugated expression $AXB + B^TX^TA^T$ with a *single* `lmiterm` command. For instance, the first command specifies $A^TX + XA$ as the "symmetrization" of XA. Finally, scalar values are allowed as shorthand for s*calar matrices*, i.e., matrices of the form αI with α scalar. Thus, a constant term of the form αI can be specified as the "scalar" α. This also applies to the coefficients P and Q of variable terms. The dimensions of scalar matrices are inferred from the context and set to 1 by default. For instance, the third LMI $S > I$ in "Example 8.3" on page 8-33 is described by

```
lmiterm([–3 1 1 2],1,1)          % 1*S*1 = S
lmiterm([3 1 1 0],1)             % 1*I = I
```

Recall that by convention `S` is considered as the right-hand side of the inequality, which justifies the –3 in the first command.

Finally, to improve readability it is often convenient to attach an identifier (tag) to each LMI and matrix variable. The variable identifiers are returned by `lmivar` and the LMI identifiers are set by the function `newlmi`. These identifiers can be used in `lmiterm` commands to refer to a given LMI or matrix variable. For the LMI system of "Example 8.1", this would look like:

```
setlmis([])
X = lmivar(1,[6 1])
S = lmivar(1,[2 0;2 1])

BRL = newlmi
lmiterm([BRL 1 1 X],1,A,'s')
lmiterm([BRL 1 1 S],C',C)
lmiterm([BRL 1 2 X],1,B)
lmiterm([BRL 2 2 S], 1,1)
```

```
Xpos = newlmi
lmiterm([-Xpos 1 1 X],1,1)

Slmi = newlmi
lmiterm([-Slmi 1 1 S],1,1)
lmiterm([Slmi 1 1 0],1)

LMISYS = getlmis
```

Here the identifiers `X` and `S` point to the variables X and S while the tags `BRL`, `Xpos`, and `Slmi` point to the first, second, and third LMI, respectively. Note that `Xpos` refers to the right-hand side of the second LMI. Similarly, `X` would indicate transposition of the variable X.

The LMI Editor lmiedit

The LMI Editor `lmiedit` is a graphical user interface (GUI) to specify LMI systems in a straightforward symbolic manner. Typing

```
lmiedit
```

calls up a window with several editable text areas and various pushbuttons. To specify your LMI system,

1. Declare each matrix variable (name and structure) in the upper half of the worksheet. The structure is characterized by its type (`S` for symmetric block diagonal, `R` for unstructured, and `G` for other structures) and by an additional "structure" matrix. This matrix contains specific information about the structure and corresponds to the second argument of `lmivar` (see "lmivar" on page 8-11 for details).

   Please use *one line per matrix variable* in the text editing areas.

2 Specify the LMIs as MATLAB expressions in the lower half of the worksheet. For instance, the LMI

$$\begin{pmatrix} A^T X + XA & XB \\ B^T X & -I \end{pmatrix} < 0$$

is entered by typing

```
[a'*x+x*a x*b; b'*x  1] < 0
```

if `x` is the name given to the matrix variable X in the upper half of the worksheet. The left- and right-hand sides of the LMIs should be *valid* MATLAB expressions.

Once the LMI system is fully specified, the following tasks can be performed by pressing the corresponding button:

- Visualize the sequence of `lmivar`/`lmiterm` commands needed to describe this LMI system (`view commands` buttons). Conversely, the LMI system defined by a particular sequence of `lmivar`/`lmiterm` commands can be displayed as a MATLAB expression by clicking on the `describe...` buttons.

  Beginners can use this facility as a tutorial introduction to the `lmivar` and `lmiterm` commands.
- Save the symbolic description of the LMI system as a MATLAB string (`save` button). This description can be reloaded later on by pressing the `load` button.
- Read a sequence of `lmivar`/`lmiterm` commands from a file (`read` button). You can then click on "`describe the matrix variables`" or "`describe the LMIs...`" to visualize the symbolic expression of the LMI system specified by these commands. The file should describe a single LMI system but may otherwise contain any sequence of MATLAB commands.

  This feature is useful for code validation and debugging.

  Write in a file the sequence of `lmivar`/`lmiterm` commands needed to describe a particular LMI system (`write` button).

  This is helpful to develop code and prototype MATLAB functions based on the LMI Lab.

- Generate the internal representation of the LMI system by pressing `create`. The result is written in a MATLAB variable named after the LMI system (if the "name of the LMI system" is set to `mylmi`, the internal representation is written in the MATLAB variable `mylmi`). Note that all LMI-related data should be defined in the MATLAB workspace at this stage.

  The internal representation can be passed directly to the LMI solvers or any other LMI Lab function.

As with `lmiterm`, you can use various shortcuts when entering LMI expressions at the keyboard. For instance, zero blocks can be entered simply as `0` and need not be dimensioned. Similarly, the identity matrix can be entered as `1` without dimensioning. Finally, *upper diagonal* LMI blocks need not be fully specified. Rather, you can just type `(*)` in place of each such block.

Though fairly general, `lmiedit` is not as flexible as `lmiterm` and the following limitations should be kept in mind:

- Parentheses cannot be used around matrix variables. For instance, the expression

  ```
  (a*x+b)'*c + c'*(a*x+b)
  ```

  is invalid when x is a variable name. By contrast,

  ```
  (a+b)'*x + x'*(a+b)
  ```

  is perfectly valid.
- Loops and `if` statements are ignored.
- When turning `lmiterm` commands into a symbolic description of the LMI system, an error is issued if the first argument of `lmiterm` cannot be evaluated. Use the LMI and variable identifiers supplied by `newlmi` and `lmivar` to avoid such difficulties.

Figure 8-1 shows how to specify the feasibility problem of "Example 8.1" on page 8-9 with `lmiedit`.

How It All Works

Users familiar with MATLAB may wonder how `lmivar` and `lmiterm` physically update the internal representation `LMISYS` since `LMISYS` is not an argument to these functions. In fact, all updating is performed through global variables for maximum speed. These global variables are initialized by `setlmis`, cleared by

`getlmis`, and not visible in the workspace. Even though this artifact is transparent from the user's viewpoint, be sure to

- Invoke `getlmis` only once and after completely specifying the LMI system
- Refrain from using the command `clear global` before the LMI system description is ended with `getlmis`

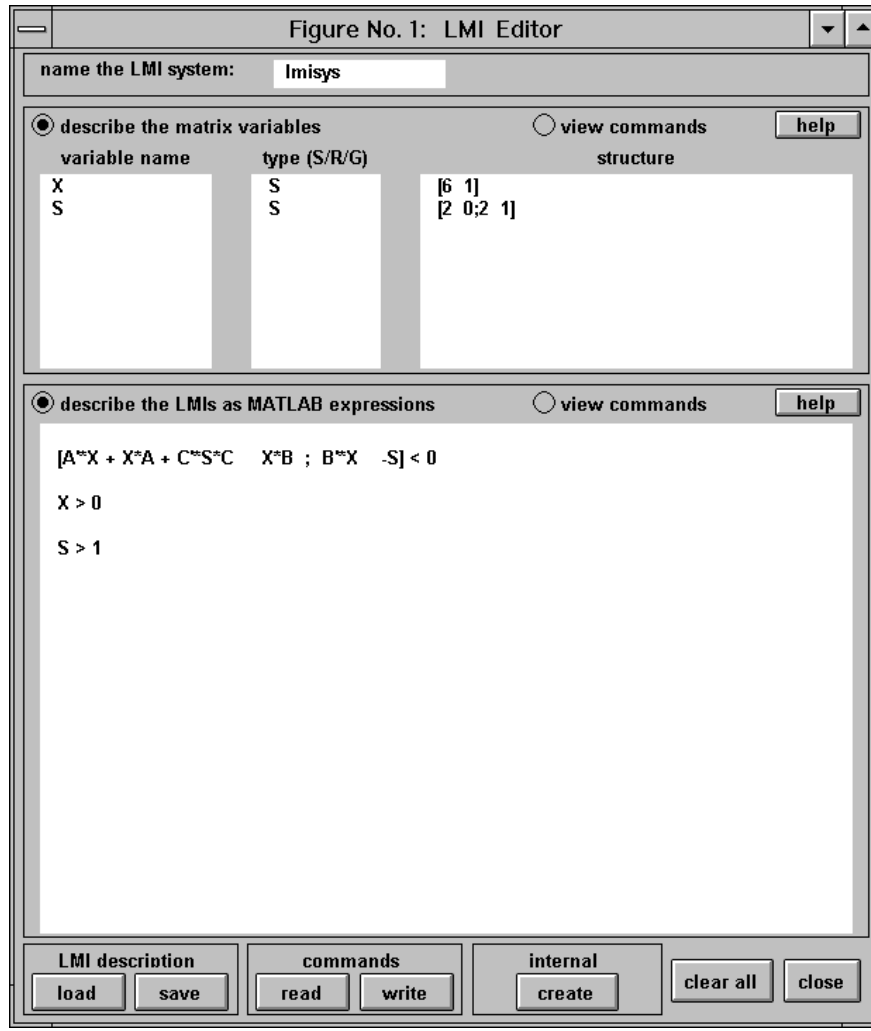


Figure 8-1: Graphical user interface lmiedit

Retrieving Information

Recall that the full description of an LMI system is stored as a single vector called the internal representation. The user should not attempt to read or retrieve information directly from this vector. Three functions called `lmiinfo`, `lminbr`, and `matnbr` are provided to extract and display all relevant information in a user-readable format.

lmiinfo

`lmiinfo` is an interactive facility to retrieve qualitative information about LMI systems. This includes the number of LMIs, the number of matrix variables and their structure, the term content of each LMI block, etc. To invoke `lmiinfo`, enter

```
lmiinfo(LMISYS)
```

where `LMISYS` is the internal representation of the LMI system produced by `getlmis`.

lminbr and matnbr

These two functions return the number of LMIs and the number of matrix variables in the system. To get the number of matrix variables, for instance, enter

```
matnbr(LMISYS)
```

LMI Solvers

LMI solvers are provided for the following three generic optimization problems (here x denotes the vector of decision variables, i.e., of the free entries of the matrix variables $X_1, \ldots, X_K$):

- Feasibility problem

  Find $x \in \mathbf{R}^N$ (or equivalently matrices $X_1, \ldots, X_K$ with prescribed structure) that satisfies the LMI system

  $$A(x) < B(x)$$

  The corresponding solver is called `feasp`.

- Minimization of a linear objective under LMI constraints

  Minimize $c^T x$ over $x \in \mathbf{R}^N$ subject to $A(x) < B(x)$

  The corresponding solver is called `mincx`.

- Generalized eigenvalue minimization problem

  Minimize λ over $x \in \mathbf{R}^N$ subject to

  $$C(x) < D(x)$$

  $$0 < B(x)$$

  $$A(x) < \lambda B(x).$$

  The corresponding solver is called `gevp`.

Note that $A(x) < B(x)$ above is a shorthand notation for general structured LMI systems with decision variables $x = (x_1, \ldots, x_N)$.

The three LMI solvers `feasp`, `mincx`, and `gevp` take as input the internal representation `LMISYS` of an LMI system and return a feasible or optimizing value x^* of the decision variables. The corresponding values of the matrix variables $X_1, \ldots, X_K$ are derived from x^* with the function `dec2mat`. These solvers are C-MEX implementations of the polynomial-time Projective Algorithm Projective Algorithm of Nesterov and Nemirovski [3, 2].

For generalized eigenvalue minimization problems, it is necessary to distinguish between the standard LMI constraints $C(x) < D(x)$ and the linear-fractional LMIs

$$A(x) < \lambda B(x)$$

attached to the minimization of the generalized eigenvalue λ. When using `gevp`, you should follow these three rules to ensure proper specification of the problem:

- Specify the LMIs involving λ as $A(x) < B(x)$ (*without* the λ)
- Specify them *last* in the LMI system. `gevp` systematically assumes that the last L LMIs are linear-fractional if L is the number of LMIs involving λ
- Add the constraint $0 < B(x)$ or any other constraint that enforces it. This positivity constraint is required for well-posedness of the problem and is not automatically added by `gevp` (see the "Command Reference" chapter for details).

An initial guess `xinit` for x can be supplied to `mincx` or `gevp`. Use `mat2dec` to derive `xinit` from given values of the matrix variables $X_1, \ldots, X_K$. Finally, various options are available to control the optimization process and the solver behavior. These options are described in detail in the "Command Reference" chapter.

The following example illustrates the use of the `mincx` solver.

Example 8.2. Consider the optimization problem

Minimize Trace(X) subject to

$$A^T X + XA + XBB^T X + Q < \qquad \textbf{(8-9)}$$

with data

$$A = \begin{pmatrix} -1 & -2 & 1 \\ 3 & 2 & 1 \\ 1 & -2 & -1 \end{pmatrix};\ B = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix};\ Q = \begin{pmatrix} 1 & -1 & 0 \\ -1 & -3 & -12 \\ 0 & -12 & -36 \end{pmatrix}$$

It can be shown that the minimizer X^* is simply the stabilizing solution of the algebraic Riccati equation

$$A^T X + XA + XBB^T X + Q = 0$$

This solution can be computed directly with the Riccati solver `care` and compared to the minimizer returned by `mincx`.

From an LMI optimization standpoint, problem (8-9) is equivalent to the following linear objective minimization problem:

$$\text{Minimize Trace(X) subject to } \begin{pmatrix} A^TX + XA + Q & XB \\ B^TX & -I \end{pmatrix} < 0 \qquad \textbf{(8-10)}$$

Since Trace(X) is a linear function of the entries of X, this problem falls within the scope of the `mincx` solver and can be numerically solved as follows:

1 Define the LMI constraint (8-9) by the sequence of commands

```
setlmis([])
X = lmivar(1,[3 1]) % variable X, full symmetric

lmiterm([1 1 1 X],1,a,'s')
lmiterm([1 1 1 0],q)
lmiterm([1 2 2 0], 1)
lmiterm([1 2 1 X],b',1)

LMIs = getlmis
```

2 Write the objective Trace(X) as c^Tx where x is the vector of free entries of X. Since c should select the diagonal entries of X, it is obtained as the decision vector corresponding to $X = I$, that is,

```
c = mat2dec(LMIs,eye(3))
```

Note that the function `defcx` provides a more systematic way of specifying such objectives (see “Specifying c^Tx Objectives for mincx” on page 8-38 for details).

3 Call `mincx` to compute the minimizer `xopt` and the global minimum `copt = c'*xopt` of the objective:

```
options = [1e 5,0,0,0,0]
```

```
[copt,xopt] = mincx(LMIs,c,options)
```

Here `1e 5` specifies the desired relative accuracy on `copt`.

The following trace of the iterative optimization performed by `mincx` appears on the screen:

```
Solver for linear objective minimization under LMI constraints

Iterations : Best objective value so far

     1
     2               8.511476
     3              13.063640
***          new lower bound:          34.023978
     4              15.768450
***          new lower bound:          25.005604
     5              17.123012
***          new lower bound:          21.306781
     6              17.882558
***          new lower bound:          19.819471
     7              18-339853
***          new lower bound:          19.189417
     8              18.552558
***          new lower bound:          18.919668
     9              18.646811
***          new lower bound:          18.803708
    10              18.687324
***          new lower bound:          18.753903
    11              18.705715
***          new lower bound:          18.732574
    12              18.712175
```

```
***           new lower bound:          18.723491
      13              18.714880
***           new lower bound:          18.719624
      14              18.716094
***           new lower bound:          18.717986
      15              18.716509
***           new lower bound:          18.717297
      16              18.716695
***           new lower bound:          18.716873


Result: feasible solution of required accuracy
   best objective value:  18.716695
   guaranteed relative accuracy: 9.50e 06
   f-radius saturation: 0.000% of R = 1.00e+09
```

The iteration number and the best value of $c^T x$ at the current iteration appear in the left and right columns, respectively. Note that no value is displayed at the first iteration, which means that a feasible x satisfying the constraint (8-10) was found only at the second iteration. Lower bounds on the global minimum of $c^T x$ are sometimes detected as the optimization progresses. These lower bounds are reported by the message

```
*** new lower bound: xxx
```

Upon termination, `mincx` reports that the global minimum for the objective Trace(X) is –18.716695 with relative accuracy of at least 9.5-by-10^{-6}. This is the value `copt` returned by `mincx`.

4 `mincx` also returns the optimizing vector of decision variables `xopt`. The corresponding optimal value of the matrix variable X is given by

```
Xopt = dec2mat(LMIs,xopt,X)
```

which returns

$$X_{\text{opt}} = \begin{pmatrix} -6.3542 & -5.8895 & 2.2046 \\ -5.8895 & -6.2855 & 2.2201 \\ 2.2046 & 2.2201 & -6.0771 \end{pmatrix}$$

This result can be compared with the stabilizing Riccati solution computed by `care`:

```
Xst = care(a,b,q, 1)
norm(Xopt-Xst)

ans =
   6.5390e 05
```

From Decision to Matrix Variables and Vice Versa

While LMIs are specified in terms of their matrix variables $X_1, \ldots, X_K$, the LMI solvers optimize the vector x of free scalar entries of these matrices, called the decision variables. The two functions `mat2dec` and `dec2mat` perform the conversion between these two descriptions of the problem variables.

Consider an LMI system with three matrix variables X_1, X_2, X_3. Given particular values `X1`, `X2`, `X3` of these variables, the corresponding value `xdec` of the vector of decision variables is returned by `mat2dec`:

```
xdec = mat2dec(LMISYS,X1,X2,X3)
```

An error is issued if the number of arguments following `LMISYS` differs from the number of matrix variables in the problem (see `matnbr`).

Conversely, given a value `xdec` of the vector of decision variables, the corresponding value of the k-th matrix is given by `dec2mat`. For instance, the value `X2` of the second matrix variable is extracted from `xdec` by

```
X2 = dec2mat(LMISYS,xdec,2)
```

The last argument indicates that the second matrix variable is requested. It could be set to the matrix variable identifier returned by `lmivar`.

The total numbers of matrix variables and decision variables are returned by `matnbr` and `decnbr`, respectively. In addition, the function `decinfo` provides precise information about the mapping between decision variables and matrix variable entries (see the “Command Reference” chapter).

Validating Results

The LMI Lab offers two functions to analyze and validate the results of an LMI optimization. The function `evallmi` evaluates all variable terms in an LMI system for a given value of the vector of decision variables, for instance, the feasible or optimal vector returned by the LMI solvers. Once this evaluation is performed, the left- and right-hand sides of a particular LMI are returned by `showlmi`.

In the LMI problem considered in "Example 8.2" on page 8-23, you can verify that the minimizer `xopt` returned by `mincx` satisfies the LMI constraint (8-10) as follows:

```
evlmi = evallmi(LMIs,xopt)
[lhs,rhs] = showlmi(evlmi,1)
```

The first command evaluates the system for the value `xopt` of the decision variables, and the second command returns the left- and right-hand sides of the first (and only) LMI. The negative definiteness of this LMI is checked by

```
eig(lhs-rhs)

ans =
    2.0387e 04
    3.9333e 05
    1.8917e 07
    4.6680e+01
```

Modifying a System of LMIs

Once specified, a system of LMIs can be modified in several ways with the functions `dellmi`, `delmvar`, and `setmvar`.

dellmi

The first possibility is to remove an entire LMI from the system with `dellmi`. For instance, suppose that the LMI system of "Example 8.1" on page 8-9 is described in `LMISYS` and that we want to remove the positivity constraint on X. This is done by

```
NEWSYS = dellmi(LMISYS,2)
```

where the second argument specifies deletion of the second LMI. The resulting system of two LMIs is returned in `NEWSYS`.

The LMI identifiers (*initial* ranking of the LMI in the LMI system) are not altered by deletions. As a result, the last LMI

$$S > I$$

remains known as the third LMI even though it now ranks second in the modified system. To avoid confusion, it is safer to refer to LMIs via the identifiers returned by `newlmi`. If `BRL`, `Xpos`, and `Slmi` are the identifiers attached to the three LMIs (8-6)–(8-8), `Slmi` keeps pointing to $S > I$ even after deleting the second LMI by

```
NEWSYS = dellmi(LMISYS,Xpos)
```

dellmi

Another way of modifying an LMI system is to delete a matrix variable, that is, to remove all variable terms involving this matrix variable. This operation is performed by `delmvar`. For instance, consider the LMI

$$A^TX + XA + BW + W^TB^T + I < 0$$

with variables $X = X^T \in \mathbf{R}^{4\times4}$ and W $\in \mathbf{R}^{2\times4}$. This LMI is defined by

```
setlmis([])
X = lmivar(1,[4 1]) % X
W = lmivar(2,[2 4]) % W
```

```
lmiterm([1 1 1 X],1,A,'s')
lmiterm([1 1 1 W],B,1,'s')
lmiterm([1 1 1 0],1)

LMISYS = getlmis
```

To delete the variable `W`, type the command

```
NEWSYS = delmvar(LMISYS,W)
```

The resulting `NEWSYS` now describes the Lyapunov inequality

$$A^TX + XA + I < 0$$

Note that `delmvar` automatically removes all LMIs that depended only on the deleted matrix variable.

The matrix variable identifiers are not affected by deletions and continue to point to the same matrix variable. For subsequent manipulations, it is therefore advisable to refer to the remaining variables through their identifier. Finally, note that deleting a matrix variable is equivalent to setting it to the zero matrix of the same dimensions with `setmvar`.

setmvar

The function `setmvar` is used to set a matrix variable to some given value. As a result, this variable is removed from the problem and all terms involving it become constant terms. This is useful, for instance, to fix`setmvar` some variables and optimize with respect to the remaining ones.

Consider again “Example 8.1” on page 8-9 and suppose we want to know if the peak gain of G itself is less than one, that is, if

$$\|G\|_\infty < 1$$

This amounts to setting the scaling matrix D (or equivalently, $S = D^TD$) to a multiple of the identity matrix. Keeping in mind the constraint $S > I$, a legitimate choice is $S = 2–\beta\psi–I$. To set S to this value, enter

```
NEWSYS = setmvar(LMISYS,S,2)
```

The second argument is the variable identifier `S`, and the third argument is the value to which S should be set. Here the value 2 is shorthand for $2-\beta\psi-I$. The resulting system `NEWSYS` reads

$$\begin{pmatrix} A^TX + XA + 2C^TC & XB \\ B^TX & -2I \end{pmatrix} < 0$$

$$X > 0$$

$$2I > I$$

Note that the last LMI is now free of variable and trivially satisfied. It could, therefore, be deleted by

```
NEWSYS = dellmi(NEWSYS,3)
```

or

```
NEWSYS = dellmi(NEWSYS,Slmi)
```

if `Slmi` is the identifier returned by `newlmi`.

Advanced Topics

This last section gives a few hints for making the most out of the LMI Lab. It is directed toward users who are comfortable with the basics described above.

Structured Matrix Variables

Fairly complex matrix variable structures and interdependencies can be specified with `lmivar`. Recall that the symmetric block-diagonal or rectangular structures are covered by Types 1 and 2 of `lmivar` provided that the matrix variables are independent. To describe more complex structures or correlations between variables, you must use Type 3 and specify each entry of the matrix variables directly in terms of the free scalar variables of the problem (the so-called decision variables).

With Type 3, each entry is specified as either 0 or $\pm x_n$ where x_n is the n-th decision variable. The following examples illustrate how to specify nontrivial matrix variable structures with `lmivar`. We first consider the case of uncorrelated matrix variables.

Example 8.3. Suppose that the problem variables include a 3-by-3 symmetric matrix X and a 3-by-3 symmetric Toeplitz matrix

$$Y = \begin{pmatrix} y_1 & y_2 & y_3 \\ y_2 & y_1 & y_2 \\ y_3 & y_2 & y_1 \end{pmatrix}$$

The variable Y has three independent entries, hence involves three decision variables. Since Y is independent of X, these decision variables should be labeled $n + 1$, $n + 2$, $n + 3$ where n is the number of decision variables involved in X. To retrieve this number, define the variable X (Type 1) by

```
setlmis([])
[X,n] =
lmivar(1,[3 1])
```

The second output argument `n` gives the total number of decision variables used so far (here `n = 6`). Given this number, Y can be defined by

```
Y = lmivar(3,n+[1 2 3;2 1 2;3 2 1])
```

or equivalently by

```
Y = lmivar(3,toeplitz(n+[1 2 3]))
```

where `toeplitz` is a standard MATLAB function. For verification purposes, we can visualize the decision variable distributions in X and Y with `decinfo`:

```
lmis = getlmis
decinfo(lmis,X)

ans =
   1   2   4
   2   3   5
   4   5   6

decinfo(lmis,Y)

ans =
   7   8   9
   8   7   8
   9   8   7
```

The next example is a problem with interdependent matrix variables.

Example 8.4. Consider three matrix variables X, Y, Z with structure

$$X = \begin{pmatrix} x & 0 \\ 0 & y \end{pmatrix}, \quad Y = \begin{pmatrix} z & 0 \\ 0 & t \end{pmatrix}, \quad Z = \begin{pmatrix} 0 & -x \\ -t & 0 \end{pmatrix}$$

where x, y, z, t are independent scalar variables. To specify such a triple, first define the two independent variables X and Y (both of Type 1) as follows:

```
setlmis([])
[X,n,sX] = lmivar(1,[1 0;1 0])
[Y,n,sY] = lmivar(1,[1 0;1 0])
```

The third output of `lmivar` gives the entry-wise dependence of X and Y on the decision variables $(x_1, x_2, x_3, x_4) := (x, y, z, t)$:

```
sX =
   1   0
   0   2
```

```
sY =
    3    0
    0    4
```

Using Type 3 of `lmivar`, you can now specify the structure of `Z` in terms of the decision variables $x_1 = x$ and $x_4 = t$:

```
[Z,n,sZ] = lmivar(3,[0  sX(1,1);-sY(2,2) 0])
```

Since `sX(1,1)` refers to x_1 while `sY(2,2)` refers to x_4 , this defines the variable

$$Z = \begin{pmatrix} 0 & -x_1 \\ -x_4 & 0 \end{pmatrix} = \begin{pmatrix} 0 & -x \\ -t & 0 \end{pmatrix}$$

as confirmed by checking its entry-wise dependence on the decision variables:

```
sZ =
    0     1
     4   0
```

Complex-Valued LMIs

The LMI solvers are written for real-valued matrices and cannot directly handle LMI problems involving complex-valued matrices. However, complex-valued LMIs can be turned into real-valued LMIs by observing that a complex Hermitian matrix $L(x)$ satisfies

$$L(x) < 0$$

if and only if

$$\begin{pmatrix} \mathrm{Re}(L(x)) & \mathrm{Im}(L(x)) \\ -\mathrm{Im}(L(x)) & \mathrm{Re}(L(x)) \end{pmatrix} < 0$$

This suggests the following systematic procedure for turning complex LMIs into real ones:

- Decompose every complex matrix variable X as

  $$X = X_1 + jX_2$$

  where X_1 and X_2 are real

- Decompose every complex matrix coefficient A as

  $A = A_1 + jA_2$

  where A_1 and A_2 are real
- Carry out all complex matrix products. This yields affine expressions in X_1, X_2 for the real and imaginary parts of each LMI, and an equivalent real-valued LMI is readily derived from the above observation.

For LMIs without outer factor, a streamlined version of this procedure consists of replacing any occurrence of the matrix variable $X = X_1 + jX_2$ by

$$\begin{pmatrix} X_1 & X_2 \\ -X_2 & X_1 \end{pmatrix}$$

and any fixed matrix $A = A_1 + jA_2$, including real ones,by

$$\begin{pmatrix} A_1 & A_2 \\ -A_2 & A_1 \end{pmatrix}$$

For instance, the real counterpart of the LMI system

$$M^H X M < X, \qquad X = X^H > I \tag{8-11}$$

reads (given the decompositions $M = M_1 + jM_2$ and $X = X_1 + jX_2$ with M_j, X_j real):

$$\begin{pmatrix} M_1 & M_2 \\ -M_2 & M_1 \end{pmatrix}^T \begin{pmatrix} X_1 & X_2 \\ -X_2 & X_1 \end{pmatrix} \begin{pmatrix} M_1 & M_2 \\ -M_2 & M_1 \end{pmatrix} < \begin{pmatrix} X_1 & X_2 \\ -X_2 & X_1 \end{pmatrix}$$

$$\begin{pmatrix} X_1 & X_2 \\ -X_2 & X_1 \end{pmatrix} < I$$

Note that $X = X^H$ in turn requires that $X_1 = X^H$ and $X_2 + X_2^T 2 = 0$. Consequently, X_1 and X_2 should be declared as symmetric and skew-symmetric matrix variables, respectively.

Assuming, for instance, that $M \in \mathbf{C}^{5\times 5}$, the LMI system (8-11) would be specified as follows:

```
M1=real(M), M2=imag(M)
bigM=[M1 M2;-M2 M1]
setlmis([])

% declare bigX=[X1 X2;-X2 X1] with X1=X1' and X2+X2'=0:

[X1,n1,sX1] = lmivar(1,[5 1])
[X2,n2,sX2] = lmivar(3,skewdec(5,n1))
bigX = lmivar(3,[sX1 sX2;-sX2 sX1])

% describe the real counterpart of (1.12):

lmiterm([1 1 1 0],1)
lmiterm([ 1 1 1 bigX],1,1)
lmiterm([2 1 1 bigX],bigM',bigM)
lmiterm([ 2 1 1 bigX],1,1)

lmis = getlmis
```

Note the three-step declaration of the structured matrix variable `bigX` = $\begin{pmatrix} X_1 & X_2 \\ -X_2 & X_1 \end{pmatrix}$:

1. Specify X_1 as a (real) symmetric matrix variable and save its structure description `sX1` as well as the number `n1` of decision variables used in X_1.
2. Specify X_2 as a skew-symmetric matrix variable using Type 3 of `lmivar` and the utility `skewdec`. The command `skewdec(5,n1)` creates a 5-by–5 skew-symmetric structure depending on the decision variables `n1 + 1`, `n1 + 2`,...
3. Define the structure of `bigX` in terms of the structures `sX1` and `sX2` of X_1 and X_2.

See the previous subsection for more details on such structure manipulations.

Specifying c^Tx Objectives for mincx

The LMI solver `mincx` minimizes linear objectives of the form c^Tx where x is the vector of decision variables. In most control problems, however, such objectives are expressed in terms of the matrix variables rather than of x. Examples include Trace(X) where X is a symmetric matrix variable, or u^TXu where u is a given vector.

The function `defcx` facilitates the derivation of the c vector when the objective is an affine function of the *matrix variables*. For the sake of illustration, consider the linear objective

$$\text{Trace}(X) + x_0^T P x_0$$

where X and P are two symmetric variables and x_0 is a given vector. If `lmisys` is the internal representation of the LMI system and if x_0, X, P have been declared by

```
x0 = [1;1]
setlmis([])
X = lmivar(1,[3 0])
P = lmivar(1,[2 1])
   :
   :
lmisys = getlmis
```

the c vector such that $c^Tx = \text{Trace}(X) + x_0^T P x_0$ can be computed as follows:

```
n = decnbr(lmisys)
c = zeros(n,1)

for j=1:n,
   [Xj,Pj] = defcx(lmisys,j,X,P)
   c(j) = trace(Xj) + x0'*Pj*x0
end
```

The first command returns the number of decision variables in the problem and the second command dimensions c accordingly. Then the `for` loop performs the following operations:

1. Evaluate the matrix variables X and P when all entries of the decision vector x are set to zero except $x_j := 1$. This operation is performed by the function `defcx`. Apart from `lmisys` and `j`, the inputs of `defcx` are the identifiers `X` and

`P` of the variables involved in the objective, and the outputs `Xj` and `Pj` are the corresponding values.

2 Evaluate the objective expression for `X := Xj` and `P := Pj`. This yields the j-th entry of `c` by definition.

In our example the result is

```
c =
   3
   1
   2
   1
```

Other objectives are handled similarly by editing the following generic skeleton:

```
n = decnbr( LMI system )
c = zeros(n,1)
for j=1:n,
   [ matrix values ] = defcx( LMI system,j,
matrix identifiers)
   c(j) = objective(matrix values)
end
```

Feasibility Radius

When solving LMI problems with `feasp`, `mincx`, or `gevp`, it is possible to constrain the solution x to lie in the ball

$$x^T x < R^2$$

where $R > 0$ is called the feasibility radius. This specifies a maximum (Euclidean norm) magnitude for x and avoids getting solutions of very large norm. This may also speed up computations and improve numerical stability. Finally, the feasibility radius bound regularizes problems with redundant variable sets. In rough terms, the set of scalar variables is redundant when an equivalent problem could be formulated with a smaller number of variables.

The feasibility radius R is set by the third entry of the options vector of the LMI solvers. Its default value is $R = 109$. Setting R to a negative value means “no rigid bound,” in which case the feasibility radius is increased during the

optimization if necessary. This “flexible bound” mode may yield solutions of large norms.

Well-Posedness Issues

The LMI solvers used in the LMI Lab are based on interior-point optimization techniques. To compute feasible solutions, such techniques require that the system of LMI constraints be strictly feasible, that is, the feasible set has a nonempty interior. As a result, these solvers may encounter difficulty when the LMI constraints are feasible but not *strictly feasible*, that is, when the LMI

$$L(x) \succeq 0$$

has solutions while

$$L(x) < 0$$

has no solution.

For feasibility problems, this difficulty is automatically circumvented by `feasp`, which reformulates the problem

$$\text{Find } x \text{ such that } L(x) \leq 0 \tag{8-12}$$

as

$$\text{Minimize } t \text{ subject to } Lx < t \times I. \tag{8-13}$$

In this modified problem, the LMI constraint is always strictly feasible in x, t and the original LMI (8-12) is feasible if and only if the global minimum $t_{\min}$ of (8-13) satisfies

$$t_{\min} \succeq 0$$

For feasible but not strictly feasible problems, however, the computational effort is typically higher as `feasp` strives to approach the global optimum $t_{\min} = 0$ to a high accuracy.

For the LMI problems addressed by `mincx` and `gevp`, nonstrict feasibility generally causes the solvers to fail and to return an “infeasibility” diagnosis. Although there is no universal remedy for this difficulty, it is sometimes possible to eliminate underlying algebraic constraints to obtain a strictly feasible problem with fewer variables.

Another issue has to do with homogeneous feasibility problems such as

$$A^T P + P A < 0, \qquad P > 0$$

While this problem is technically well-posed, the LMI optimization is likely to produce solutions close to zero (the trivial solution of the nonstrict problem). To compute a nontrivial Lyapunov matrix and easily differentiate between feasibility and infeasibility, replace the constraint $P > 0$-by-$P > \alpha I$ with $\alpha > 0$. Note that this does not alter the problem due to its homogeneous nature.

Semi-Definite *B(x)* in gevp Problems

Consider the generalized eigenvalue minimization problem

$$\text{Minimize } \lambda \text{ subject to } A(x) < \lambda B(x),\ B(x) > 0,\ C(x) < 0. \tag{8-14}$$

Technically, the positivity of $B(x)$ for some $x \in \mathbf{R}^n$ is required for the well-posedness of the problem and the applicability of polynomial-time interior-point methods. Hence problems where

$$B(x) = \begin{pmatrix} B_1(x) & 0 \\ 0 & 0 \end{pmatrix}, \quad \text{with } B_1(x) > 0 \text{ strictly feasible}$$

cannot be directly solved with `gevp`. A simple remedy consists of replacing the constraints

$$A(x) < B(x), \qquad B(x) > 0$$

by

$$A(x) < \begin{pmatrix} Y & 0 \\ 0 & 0 \end{pmatrix}, \quad Y < \lambda B_1(x), \quad B_1(x) > 0$$

where Y is an additional symmetric variable of proper dimensions. The resulting problem is equivalent to (8-14) and can be solved directly with `gevp`.

Efficiency and Complexity Issues

As explained in the beginning of the chapter, the term-oriented description of LMIs used in the LMI Lab typically leads to higher efficiency than the canonical representation

$$A_0 + x_1 A_1 + \ldots + x_N A_N < 0. \tag{8-15}$$

This is no longer true, however, when the number of variable terms is nearly equal to or greater than the number N of decision variables in the problem. If your LMI problem has few free scalar variables but many terms in each LMI, it is therefore preferable to rewrite it as (8-15) and to specify it in this form. Each scalar variable x_j is then declared independently and the LMI terms are of the form $x_j A_j$.

If M denotes the total row size of the LMI system and N the total number of scalar decision variables, the flop count per iteration for the `feasp` and `mincx` solvers is proportional to

- N^3 when the least-squares problem is solved via. Cholesly factorization of the Hessian matrix (default) [2]
- M-by-N^2 when numerical instabilities warrant the use of QR factorization instead

While the theory guarantees a worst-case iteration count proportional to M, the number of iterations actually performed grows slowly with M in most problems. Finally, while `feasp` and `mincx` are comparable in complexity, `gevp` typically demands more computational effort. Make sure that your LMI problem cannot be solved with `mincx` before using `gevp`.

Solving $M + P^T X Q + Q^T X^T P < 0$

In many output-feedback synthesis problems, the design can be performed in two steps:

1. Compute a closed-loop Lyapunov function via LMI optimization.

2. Given this Lyapunov function, derive the controller state-space matrices by solving an LMI of the form

   $$M + P^T X Q + Q^T X^T P < 0 \tag{8-16}$$

   where M, P, Q are given matrices and X is an unstructured m-by-n matrix variable.

It turns out that a particular solution X_c of (8-16) can be computed via simple linear algebra manipulations [1]. Typically, X_c corresponds to the center of the ellipsoid of matrices defined by (8-16).

The function `basiclmi` returns the "explicit" solution X_c:

```
Xc = basiclmi(M,P,Q)
```

Since this central solution sometimes has large norm, `basiclmi` also offers the option of computing an approximate least-norm solution of (8-16). This is done by

```
X = basiclmi(M,P,Q,'Xmin')
```

and involves LMI optimization to minimize $\|X\|$.

References

[1] Gahinet, P., and P. Apkarian, "A Linear Matrix Inequality Approach to H_∞ Control," *Int. J. Robust and Nonlinear Contr.*, 4 (1994), pp. 421–448.

[2] Nemirovski, A., and P. Gahinet, "The Projective Method for Solving Linear Matrix Inequalities," *Proc. Amer. Contr. Conf.*, 1994, pp. 840–844.

[3] Nesterov, Yu, and A. Nemirovski, *Interior Point Polynomial Methods in Convex Programming: Theory and Applications*, SIAM Books, Philadelphia, 1994.

[4] Shamma, J.S., "Robustness Analysis for Time-Varying Systems," *Proc. Conf. Dec. Contr.*, 1992, pp. 3163–3168.

9

Command Reference

This chapter gives a detailed description of all LMI Control Toolbox functions. Functions are grouped by application in tables at the beginning of this chapter. In addition, information on each function is aavailable through the online Help facility.

List of Functions

Linear Algebra	
`basiclmi`	Solve $M + P^T X Q + Q^T X^T P < 0$
`care`	Continuous-time Riccati solver
`dare`	Discrete-time Riccati solver
`dricpen`	Generalized Riccati solver (discrete-time)
`ricpen`	Generalized Riccati solver (continuous-time)

Linear Time-Invariant Systems	
`ltisys`	Specify a `SYSTEM` matrix
`ltiss`	Extract state-space data
`ltitf`	Compute the transfer function (SISO)
`sbalanc`	Balance a state-space realization via diagonal similarity
`sinfo`	Return information about a `SYSTEM` matrix
`sinv`	Invert an LTI system
`splot`	Plot the time and frequency responses of LTI systems
`spol`	Return the poles of an LTI system
`sresp`	Frequency response of an LTI system
`ssub`	Extract a subsystem from a system

Interconnection of Systems	
`sadd`	Form the parallel interconnection of systems
`sconnect`	Describe general control loops

Interconnection of Systems	
sdiag	Append (concatenate) several systems
slft	Form linear-fractional interconnections
sloop	Form elementary feedback interconnections
smult	Form the series interconnection of systems

Polytopic and Parameter-Dependent Systems (P-Systems)	
aff2pol	Polytopic representation of an affine parameter-dependent system
pdsimul	Simulation of parameter-dependent systems along a parameter trajectory
psinfo	Retrieve information about a P-system
psys	Specify a P-system
pvec	Specify a vector of uncertain or time-varying parameters
pvinfo	Interpret the output of pvec

Dynamical Uncertainty	
ublock	Specify an uncertainty block
udiag	Specify block-diagonal uncertainty from individual uncertainty blocks
uinfo	Display the characteristics of uncertainty structures
aff2lft	Linear-fractional representation of an affine parameter-dependent system

Robustness Analysis	
`decay`	Quadratic decay rate
`dnorminf`	RMS gain of a discrete-time LTI system
`muperf`	Robust performance of an uncertain LTI system
`mustab`	Robust stability of an uncertain LTI system
`norminf`	H_∞ norm (RMS gain) of an LTI system
`norm2`	H_2 performance of an LTI system
`quadperf`	Quadratic performance
`quadstab`	Quadratic stability
`pdlstab`	Robust stability analysis via parameter-dependent Lyapunov functions
`popov`	Popov criterion

H_∞ Control and Loop Shaping

Continuous Time	
`hinflmi`	LMI-based H_∞ synthesis
`hinfmix`	Mixed H_2/H_∞ synthesis with regional pole placement
`hinfric`	Riccati-based H_∞ synthesis
`hinfpar`	Extract state-space data from the plant `SYSTEM` matrix
`lmireg`	Specify LMI regions for pole placement
`magshape`	Graphical specification of the shaping filters
`msfsyn`	Multi-model/objective state-feedback synthesis
`sconnect`	Specify general control structures

Discrete Time	
`dhinflmi`	LMI-based H_∞ synthesis
`dhinfric`	Riccati-based H_∞ synthesis
`hinfpar`	Retrieve the plant state-space data

Gain Scheduling	
hinfgs	Synthesis of robust gain-scheduled controllers
pdsimul	Simulation along parameter trajectories

LMI Lab: Specifying and Solving LMIs

Specification of LMIs	
lmiedit	GUI for LMI specification
setlmis	Initialize the LMI description
lmivar	Define a new matrix variable
lmiterm	Specify the term content of an LMI
newlmi	Attach an identifying tag to new LMIs
getlmis	Get the internal description of the LMI system

LMI Solvers	
feasp	Feasibility of a system of LMIs
gevp	Generalized eigenvalue minimization under LMI constraints
mincx	Minimization of a linear objective under LMI constraints
dec2mat	Convert the output of the solvers into values of the matrix variables
defcx	Help define $c^T x$ objectives for `mincx`

Evaluation of LMIs/Validation of Results	
evallmi	Evaluate all variable terms for given values of the decision variables
showlmi	Return the left- and right-hand sides of an evaluated LMI
setmvar	Set a matrix variable to a given value

LMI Lab: Additional Facilities

Information Retrieval	
decinfo	Visualize the mapping between matrix variable entries and decision variables
decnbr	Get the number of decision variables, that is, the number of free scalar variables in a problem
lmiinfo	Inquire about an existing system of LMIs
lminbr	Get the number of LMIs in a problem
matnbr	Get the number of matrix variables in a problem

Manipulations of LMI Variables	
decinfo	Express each matrix variable entry in terms of the decision variables
dec2mat	Return the matrix variable values given a vector of decision variables
mat2dec	Return the vector of decision variables given the matrix variable values

Modification of LMI Systems	
dellmi	Delete an LMI from the system
delmvar	Remove a matrix variable from the problem
setmvar	Instantiate a matrix variable

Purpose

Compute the linear-fractional representation of an affine parameter-dependent system

Syntax

```
[sys,delta] = aff2lft(affsys)
```

Description

The input argument `affsys` describes an affine parameter-dependent system of the form

$$E(p)\dot{x} = A(p)x + B(p)u$$

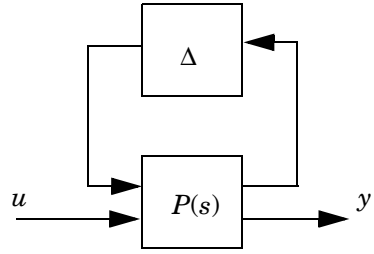
$$y = C(p)x + D(p)u$$

where $p = (p_1, \ldots, p_n)$ is a vector of uncertain or time-varying real parameters taking values in a box:

$$\underline{p}_j \le pj \le \bar{p}_j$$

Such systems are specified with `psys`.

Given this affine parameter-dependent model, the function `aff2lft` returns the equivalent linear-fractional representation



where

- $P(s)$ is an LTI system called the nominal plant
- The block-diagonal operator $\Delta = \mathrm{diag}(\delta_1 I_{r_1}, \ldots, \delta_n I_{r_n})$ accounts for the uncertainty on the real parameters p_j. Each scalar δ_j is real, possibly repeated ($r_j > 1$), and bounded by

$$|\delta_j| \le \frac{\bar{p}_j - \underline{p}_j}{2}$$

The SYSTEM matrix of P and the description of Δ are returned in `sys` and `delta`. Closing the loop with $\Delta \equiv 0$ yields the LTI system

$$E(p_{av})\dot{x} = A(p_{av})x + B(p_{av})u$$
$$y = C(p_{av})x + D(p_{av})u$$

where $p_{av} = \frac{1}{2}(\underline{p} + \overline{p})$ is the vector of average parameter values.

Example

See example in "Sector-Bounded Uncertainty" on page 2-30.

See Also

`psys`, `pvec`, `ublock`, `udiag`

Purpose

Convert affine parameter-dependent models to polytopic ones

Syntax

```
polsys = aff2pol(affsys)
```

Description

`aff2pol` derives a polytopic representation `polsys` of the *affine* parameter-dependent system

$$E(p)\dot{x} = A(p)x + B(p)u \qquad \textbf{(9-1)}$$

$$y = C(p)x + D(p)u \qquad \textbf{(9-2)}$$

where $p = (p_1, \ldots, p_n)$ is a vector of uncertain or time-varying real parameters taking values in a box or a polytope. The description `affsys` of this system should be specified with `psys`.

The vertex systems of `polsys` are the instances of (9-1)–(9-2) at the vertices p_{ex} of the parameter range, i.e., the `SYSTEM` matrices

$$\begin{pmatrix} A(p_{ex}) + jE(p_{ex}) & B(p_{ex}) \\ C(p_{ex}) & D(p_{ex}) \end{pmatrix}$$

for all corners p_{ex} of the parameter box or all vertices p_{ex} of the polytope of parameter values.

Example

"Example 2.1" on page 2-21.

See Also

`psys`, `pvec`, `aff2lft`

basiclmi

Purpose

Compute a solution X of $M + P^{T}XQ + Q^{T}X^{T}P < 0$

Syntax

```
X = basiclmi(M,P,Q,option)
```

Description

`basiclmi` computes a solution X of the LMI

$$M + P^{T}XQ + Q^{T}X^{T}P < 0$$

where M is symmetric and P,Q are matrices of compatible dimensions. If W_P and W_Q are orthonormal bases of the null space of P and Q, this LMI is solvable if and only if

$$W_P^{T} M W_P < 0 \quad \text{and} \quad W_Q^{T} M W_Q < 0$$

The output is `[]` when these conditions are not satisfied.

The solution X is derived by standard linear algebra manipulations. When `option` is set to `'Xmin'`, `basiclmi` computes the least-norm solution by solving the LMI problem

Minimize τ

subject to $M + P^{T}XQ + Q^{T}X^{T}P < 0$

$$\begin{pmatrix} \tau I & X \\ X^{T} & \tau I \end{pmatrix} \geq 0$$

See Also

`feasp`, `mincx`

Purpose

Solve continuous-time Riccati equations (CARE)

Syntax

```
X = care(A,B,Q,R,S,E)
[X,L,G,RR] = care(A,B,Q,R,S,E)
```

Description

This function computes the stabilizing solution X of the Riccati equation

$$A^{\mathrm{T}}XE + E^{\mathrm{T}}XA - (E^{\mathrm{T}}XB + S)R^{-1}(B^{T}XE + S^{T}) + Q = 0$$

where E is an invertible matrix. The solution is computed by unitary deflation of the associated Hamiltonian pencil

$$\mathcal{H} - \lambda\mathcal{L} = \begin{bmatrix} \mathrm{A} & 0 & \mathrm{B} \\ -\mathrm{Q} & -\mathrm{A}^{\mathrm{T}} & -\mathrm{S} \\ \mathrm{S}^{\mathrm{T}} & \mathrm{B}^{\mathrm{T}} & \mathrm{R} \end{bmatrix} - \lambda \begin{bmatrix} \mathrm{E} & 0 & 0 \\ 0 & \mathrm{E}^{\mathrm{T}} & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

For solvability, the pencil $\mathcal{H} - \lambda\mathcal{L}$ must have no finite generalized eigenvalue on the imaginary axis.

If the input arguments `S` and `E` are omitted, the matrices S and E are set to the default values $S = 0$ and $E = I$. The function `care` also returns the vector L of closed-loop eigenvalues, the gain matrix `G` given by

$$G = -R^{-1}(B^{T}XE + S^{T}),$$

and the Frobenius norm `RR` of the relative residual matrix.

Reference

Laub, A. J., "A Schur Method for Solving Algebraic Riccati Equations," *IEEE Trans. Aut. Contr.*, AC–24 (1979), pp. 913–921.

Arnold, W.F., and A.J. Laub, "Generalized Eigenproblem Algorithms and Software for Algebraic Riccati Equations," *Proc. IEEE*, 72 (1984), pp. 1746–1754.

See Also

`ricpen`, `cares`, `dare`

Purpose

Solve discrete-time Riccati equations (DARE)

Syntax

```
X = dare(A,B,Q,R,S,E)
[X,L,G,RR] = dare(A,B,Q,R,S,E)
```

Description

`dare` computes the stabilizing solution X of the algebraic Riccati equation

$$A^TXA - E^TXE - (A^TXB + S)(B^TXB + R)^{-1}(B^TXA + S^T) + Q = 0 \qquad \textbf{(9-3)}$$

where E is an invertible matrix. The solution is computed by unitary deflation of the associated symplectic pencil

$$\mathcal{H} - \lambda\mathcal{L} = \begin{bmatrix} A & 0 & B \\ -Q & -E^T & -S \\ S^T & 0 & R \end{bmatrix} - \lambda \begin{bmatrix} E & 0 & 0 \\ 0 & A^T & 0 \\ 0 & -B^T & 0 \end{bmatrix}$$

For solvability, the pencil $\mathcal{H} - \lambda\mathcal{L}$ must have no finite generalized eigenvalue on the unit circle.

If the input arguments `S` and `E` are omitted, the matrices S and E are set to the default values $S = 0$ and $E = I$. The function `dare` also returns the vector `L` of closed-loop eigenvalues, the gain matrix `G` given by

$$G = -(B^TXB + R)^{-1}(B^TXA + S^T),$$

and the Frobenius norm `RR` of the relative residual matrix.

Remark

The function `dricpen` is a variant of `dare` where the inputs are the two matrices H and L defined above. The syntax is

```
X = dricpen(H,L)
```

or

```
[X1,X2] = dricpen(H,L)
```

This second syntax returns two matrices X_1, X_2 such that

- $\begin{bmatrix} X_1 \\ X_2 \end{bmatrix}$ has orthonormal columns,

- $X = X_2X_1^{-1}$ is the stabilizing solution of (9-3).

Reference

Pappas, T., A.J. Laub, and N.R. Sandell, "On the Numerical Solution of the Discrete-Time Algebraic Riccati Equation," *IEEE Trans. Aut. Contr.*, AC–25 (1980), pp. 631–641.

Arnold, W.F., and A.J. Laub, "Generalized Eigenproblem Algorithms and Software for Algebraic Riccati Equations," *Proc. IEEE*, 72 (1984), pp. 1746–1754.

See Also

`dricpen`, `dares`, `care`

decay

Purpose

Quadratic decay rate of polytopic or affine P-systems

Syntax

```
[drate,P] = decay(ps,options)
```

Description

For affine parameter-dependent systems

$$E(p)\dot{x} = A(p)x, \qquad p(t) = (p_1(t), \ldots, p_n(t))$$

or polytopic systems

$$E(t)\dot{x} = A(t)x, \qquad (A, E) \in \mathrm{Co}\{(A_1, E_1), \ldots, (A_n, E_n)\},$$

`decay` returns the quadratic decay rate drate, i.e., the smallest $\alpha \in \mathbf{R}$ such that

$$A^{\mathrm{T}}QE + EQA^T < \alpha Q$$

holds for some Lyapunov matrix $Q > 0$ and all possible values of (A, E). Two control parameters can be reset via `options(1)` and `options(2)`:

- If `options(1)=0` (default), `decay` runs in fast mode, using the least expensive sufficient conditions. Set `options(1)=1` to use the least conservative conditions.
- `options(2)` is a bound on the condition number of the Lyapunov matrix P. The default is 10^9.

Example

See “Example 3.3” on page 3-9.

See Also

`quadstab`, `pdlstab`, `mustab`, `psys`

Purpose

Describe how the entries of a matrix variable X relate to the decision variables

Syntax

```
decinfo(lmisys)
decX = decinfo(lmisys,X)
```

Description

The function `decinfo` expresses the entries of a matrix variable X in terms of the decision variables $x_1, \ldots, x_N$. Recall that the decision variables are the free scalar variables of the problem, or equivalently, the free entries of all matrix variables described in `lmisys`. Each entry of X is either a hard zero, some decision variable x_n, or its opposite $-x_n$.

If `X` is the identifier of X supplied by `lmivar`, the command

```
decX = decinfo(lmisys,X)
```

returns an integer matrix `decX` of the same dimensions as X whose (i, j) entry is

- 0 if $X(i, j)$ is a hard zero
- n if $X(i, j) = x_n$ (the n-th decision variable)
- $-n$ if $X(i, j) = -x_n$

`decX` clarifies the structure of X as well as its entry-wise dependence on $x_1, \ldots, x_N$. This is useful to specify matrix variables with atypical structures (see `lmivar`).

`decinfo` can also be used in interactive mode by invoking it with a single argument. It then prompts the user for a matrix variable and displays in return the decision variable content of this variable.

Example 1

Consider an LMI with two matrix variables X and Y with structure:

- $X = x\, I_3$ with x scalar
- Y rectangular of size 2-by-1

If these variables are defined by

```
setlmis([])
X = lmivar(1,[3 0])
Y = lmivar(2,[2 1])
    :
    :
lmis = getlmis
```

the decision variables in X and Y are given by

```
dX = decinfo(lmis,X)

dX =
   1   0   0
   0   1   0
   0   0   1

dY = decinfo(lmis,Y)

dY =
   2
   3
```

This indicates a total of three decision variables x_1, x_2, x_3 that are related to the entries of X and Y by

$$X = \begin{pmatrix} x_1 & 0 & 0 \\ 0 & x_1 & 0 \\ 0 & 0 & x_1 \end{pmatrix}, \qquad Y = \begin{pmatrix} x_2 \\ x_3 \end{pmatrix}$$

Note that the number of decision variables corresponds to the number of free entries in X and Y when taking structure into account.

Example 2

Suppose that the matrix variable X is symmetric block diagonal with one 2-by-2 full block and one 2-by-2 scalar block, and is declared by

```
setlmis([])
X = lmivar(1,[2 1;2 0])
       :
lmis = getlmis
```

The decision variable distribution in X can be visualized interactively as follows:

```
decinfo(lmis)

There are 4 decision variables labeled x1 to x4 in this problem.
```

```
Matrix variable Xk of interest (enter k between 1 and 1, or 0 to
quit):

?> 1

The decision variables involved in X1 are among {-x1,...,x4}.
Their entry-wise distribution in X1 is as follows
        (0,j>0,-j<0 stand for 0,xj,-xj, respectively):

X1 :

   1   2   0   0
   2   3   0   0
   0   0   4   0
   0   0   0   4

            *********

Matrix variable Xk of interest (enter k between 1 and 1, or 0 to
quit):

?> 0
```

See Also lmivar, mat2dec, dec2mat, decnbr, decinfo

decnbr

Purpose

Give the total number of decision variables in a system of LMIs

Syntax

```
ndec = decnbr(lmisys)
```

Description

The function `decnbr` returns the number `ndec` of decision variables (free scalar variables) in the LMI problem described in `lmisys`. In other words, `ndec` is the length of the vector of decision variables.

Example

For an LMI system `lmis` with two matrix variables X and Y such that

- X is symmetric block diagonal with one 2-by-2 full block, and one 2-by-2 scalar block
- Y is 2-by-3 rectangular,

the number of decision variables is

```
ndec = decnbr(LMIs)

ndec =
      10
```

This is exactly the number of free entries in X and Y when taking structure into account (see `decinfo` for more details).

See Also

`dec2mat`, `decinfo`, `mat2dec`

Purpose

Given values of the decision variables, derive the corresponding values of the matrix variables

Syntax

```
valX = dec2mat(lmisys,decvars,X)
```

Description

Given a value `decvars` of the vector of decision variables, `dec2mat` computes the corresponding value `valX` of the matrix variable with identifier `X`. This identifier is returned by `lmivar` when declaring the matrix variable.

Recall that the decision variables are all free scalar variables in the LMI problem and correspond to the free entries of the matrix variables $X_1, \ldots, X_K$. Since LMI solvers return a feasible or optimal value of the vector of decision variables, `dec2mat` is useful to derive the corresponding feasible or optimal values of the matrix variables.

Example

See the description of `feasp`.

See Also

`mat2dec`, `decnbr`, `decinfo`

defcx

Purpose

Help specify $c^T x$ objectives for the `mincx` solver

Syntax

```
[V1,...,Vk] = defcx(lmisys,n,X1,...,Xk)
```

Description

`defcx` is useful to derive the c vector needed by `mincx` when the objective is expressed in terms of the matrix variables.

Given the identifiers `X1,...,Xk` of the matrix variables involved in this objective, `defcx` returns the values `V1,...,Vk` of these variables when the n-th decision variable is set to one and all others to zero. See the example in "Specifying cTx Objectives for mincx" on page 8-38 for more details on how to use this function to derive the c vector.

See Also

`mincx`, `decinfo`

Purpose

Remove an LMI from a given system of LMIs

Syntax

```
newsys = dellmi(lmisys,n)
```

Description

`dellmi` deletes the `n`-th LMI from the system of LMIs described in `lmisys`. The updated system is returned in `newsys`.

The ranking `n` is relative to the order in which the LMIs were declared and corresponds to the identifier returned by `newlmi`. Since this ranking is not modified by deletions, it is safer to refer to the remaining LMIs by their identifiers. Finally, matrix variables that only appeared in the deleted LMI are removed from the problem.

Example

Suppose that the three LMIs

$$A_1^T X_1 + X_1 A_1 + Q_1 < 0$$
$$A_2^T X_2 + X_2 A_2 + Q_2 < 0$$
$$A_3^T X_3 + X_3 A_3 + Q_3 < 0$$

have been declared in this order, labeled `LMI1, LMI2, LMI3` with `newlmi`, and stored in `lmisys`. To delete the second LMI, type

```
lmis = dellmi(lmisys,LMI2)
```

`lmis` now describes the system of LMIs

$$A_1^T X_1 + X_1 A_1 + Q_1 < 0 \tag{9-4}$$

$$A_3^T X_3 + X_3 A_3 + Q_3 < 0 \tag{9-5}$$

and the second variable X_2 has been removed from the problem since it no longer appears in the system (9-4)–(9-5).

To further delete (9-5), type

```
lmis = dellmi(lmis,LMI3)
```

or equivalently

```
lmis = dellmi(lmis,3)
```

Note that (9-5) has retained its original ranking after the first deletion.

See Also

`newlmi`, `lmiedit`, `lmiinfo`

Purpose

Delete one of the matrix variables of an LMI problem

Syntax

```
newsys = delmvar(lmisys,X)
```

Description

`delmvar` removes the matrix variable X with identifier `X` from the list of variables defined in `lmisys`. The identifier `X` should be the second argument returned by `lmivar` when declaring X. All terms involving X are automatically removed from the list of LMI terms. The description of the resulting system of LMIs is returned in `newsys`.

Example

Consider the LMI

$$0 < \begin{pmatrix} A^T Y + B^T Y A + Q & CX + D \\ X^T C^T + D^T & -(X + X^T) \end{pmatrix}$$

involving two variables X and Y with identifiers `X` and `Y`. To delete the variable X, type

```
lmisys = delmvar(lmisys,X)
```

Now `lmisys` describes the LMI

$$0 < \begin{pmatrix} A^T Y B + B^T Y A + Q & D \\ D^T & 0 \end{pmatrix}$$

with only one variable Y. Note that Y is still identified by the label `Y`.

See Also

`lmivar`, `setmvar`, `lmiinfo`

dhinflmi

Purpose

LMI-based H_∞ synthesis for discrete-time systems

Syntax

```
gopt = dhinflmi(P,r)
[gopt,K] = dhinflmi(P,r)
[gopt,K,x1,x2,y1,y2] = dhinflmi(P,r,g,tol,options)
```

Description

`dhinflmi` is the counterpart of `hinflmi` for discrete-time plants. Its syntax and usage mirror those of `hinflmi`.

The H_∞ performance γ is achievable if and only if the following system of LMIs has symmetric solutions R and S:

$$\begin{pmatrix} N_{12} & 0 \\ 0 & I \end{pmatrix}^T \begin{pmatrix} ARA^T - R & ARC_1^T & B_1 \\ C_1RA^T & -\gamma I + C_1RC_1^T & D_{11} \\ B_1^T & D_{11}^T & -\gamma I \end{pmatrix} \begin{pmatrix} N_{12} & 0 \\ 0 & I \end{pmatrix} < 0$$

$$\begin{pmatrix} N_{21} & 0 \\ 0 & I \end{pmatrix}^T \begin{pmatrix} A^TSA - S & A^TSB_1 & C_1^T \\ B_1^TSA & -\gamma I + B_1^TSB_1 & D_{11}^T \\ C_1 & D_{11} & -\gamma I \end{pmatrix} \begin{pmatrix} N_{21} & 0 \\ 0 & I \end{pmatrix} < 0$$

$$\begin{pmatrix} R & I \\ I & S \end{pmatrix} \geq 0$$

where N_{12} and N_{21} denote bases of the null spaces of (B_2^T, D_{12}^T) and (C_2, D_{21}), respectively. The function `dhinflmi` returns solutions R = `x1` and S = `y1` of this LMI system for γ = `gopt` (the matrices `x2` and `y2` are set to `gopt`-by-I).

Reference

Gahinet, P. and P. Apkarian, "A Linear Matrix Inequality Approach to H_∞ Control," *Int. J. Robust and Nonlinear Contr.*, 4 (1994), pp. 421–448.

See Also

`dhinfric`, `hinflmi`

Purpose

Riccati-based H_∞ synthesis for discrete-time systems

Syntax

```
gopt = dhinfric(P,r)
[gopt,K] = dhinfric(P,r)
[gopt,K,x1,x2,y1,y2] = dhinfric(P,r,gmin,gmax,tol,options)
```

Description

`dhinfric` is the counterpart of `hinfric` for discrete-time plants $P(s)$ with equations:

$$x_{k+1} = Ax_k + B_1 w_k + B_2 u_k$$
$$z_k = C_1 x_k + D_{11} w_k + D_{12} u_k$$
$$y_k = C_2 x_k + D_{21} w_k + D_{22} u_k.$$

See `dnorminf` for a definition of the RMS gain (H_∞ performance) of discrete-time systems. The syntax and usage of `dhinfric` are identical to those of `hinfric`.

Restriction

D_{12} and D_{21} must have full rank.

See Also

`dhinflmi`, `hinfric`

dnorminf

Purpose

Compute the random mean-squares (RMS) gain of discrete-time systems

Syntax

```
[gain,peakf] = dnorminf(g,tol)
```

Description

For sampled-data LTI systems with state-space equations

$$Ex_{k+1} = Ax_k + Bu_k$$

$$y_k+1 = Cx_k + Du_k$$

and transfer function $G(z) = D + C(zE - A)^{-1}B$, `dnorminf` computes the peak gain

$$\|G\|_\infty = \sup_{\omega \in [0, 2\pi]} \sigma_{max}(G(e^{j\omega})) \qquad \textbf{(9-6)}$$

of the response on the unit circle. When $G(z)$ is stable, this peak gain coincides with the RMS gain or H_∞ norm of G, that is,

$$\|G\|_\infty = \sup_{\substack{u \in t_2 \\ u \neq 0}} \frac{\|y\|_{l_2}}{\|u\|_{l_2}}$$

The input `g` is a `SYSTEM` matrix containing the state-space data *A, B, C, D, E*. On output, `dnorminf` returns the RMS gain `gain` and the "frequency" `peakf` at which this gain is attained, i.e., the value of ω maximizing (9-6). The relative accuracy on the computed RMS gain can be adjusted with the optional input `tol` (the default value is 10^{-2}).

Reference

Boyd, S., V. Balakrishnan, and P. Kabamba, "A Bisection Method for Computing the H_∞ Norm of a Transfer Matrix and Related Problems," *Math. Contr. Sign. Syst.*, 2 (1989), pp. 207–219.

Bruisma, N.A., and M. Steinbuch, "A Fast Algorithm to Compute the H_∞-Norm of a Transfer Function Matrix," *Syst. Contr. Letters*, 14 (1990), pp. 287–293.

Robel, G., "On Computing the Infinity Norm," *IEEE Trans. Aut. Contr.*, AC–34 (1989), pp. 882–884.

See Also

`norminf`

Purpose

Given a particular instance of the decision variables, evaluate all variable terms in the system of LMIs

Syntax

```
evalsys = evallmi(lmisys,decvars)
```

Description

`evallmi` evaluates all LMI constraints for a particular instance `decvars` of the vector of decision variables. Recall that `decvars` fully determines the values of the matrix variables $X_1, \ldots, X_K$. The "evaluation" consists of replacing all terms involving $X_1, \ldots, X_K$ by their matrix value. The output `evalsys` is an LMI system containing only constant terms.

The function `evallmi` is useful for validation of the LMI solvers' output. The vector returned by these solvers can be fed directly to `evallmi` to evaluate all variable terms. The matrix values of the left- and right-hand sides of each LMI are then returned by `showlmi`.

Observation

`evallmi` is meant to operate on the output of the LMI solvers. To evaluate all LMIs for particular instances of the matrix variables $X_1, \ldots, X_K$, first form the corresponding decision vector x with `mat2dec` and then call `evallmi` with x as input.

Example

Consider the feasibility problem of finding $X > 0$ such that

$$A^T X A - X + I < 0$$

where $A = \begin{pmatrix} 0.5 & -0.2 \\ 0.1 & -0.7 \end{pmatrix}$. This LMI system is defined by:

```
setlmis([])
X = lmivar(1,[2 1]) % full symmetric X

lmiterm([1 1 1 X],A',A) % LMI #1: A'*X*A
lmiterm([1 1 1 X],-1,1) % LMI #1: -X
lmiterm([1 1 1 0],1) % LMI #1: I
lmiterm([-2 1 1 X],1,1) % LMI #2: X
lmis = getlmis
```

To compute a solution `xfeas`, call `feasp` by

```
[tmin,xfeas] = feasp(lmis)
```

The result is

```
tmin =
   -4.7117e+00

xfeas' =
   1.1029e+02  -1.1519e+01  1.1942e+02
```

The LMI constraints are therefore feasible since `tmin` < 0. The solution X corresponding to the feasible decision vector `xfeas` would be given by `X = dec2mat(lmis,xfeas,X).`

To check that `xfeas` is indeed feasible, evaluate all LMI constraints by typing

```
evals = evallmi(lmis,xfeas)
```

The left- and right-hand sides of the first and second LMIs are then given by

```
[lhs1,rhs1] = showlmi(evals,1)
[lhs2,rhs2] = showlmi(evals,2)
```

and the test

```
eig(lhs1-rhs1)
ans =
   -8.2229e+01
   -5.8163e+01
```

confirms that the first LMI constraint is satisfied by `xfeas`.

See Also

`showlmi`, `setmvar`, `dec2mat`, `mat2dec`

Purpose

Find a solution to a given system of LMIs

Syntax

```
[tmin,xfeas] = feasp(lmisys,options,target)
```

Description

The function `feasp` computes a solution `xfeas` (if any) of the system of LMIs described by `lmisys`. The vector `xfeas` is a particular value of the decision variables for which all LMIs are satisfied.

Given the LMI system

$$N^{\mathrm{T}} L\mathrm{x} N \le M^{\mathrm{T}} R(\mathrm{x}) M, \qquad \textbf{(9-7)}$$

`xfeas` is computed by solving the auxiliary convex program:

Minimize t subject to $\mathcal{N}^{\mathrm{T}}\mathcal{L}(\mathrm{x})\mathcal{N} - \mathcal{M}^{\mathrm{T}}\mathcal{R}(\mathrm{x})\mathcal{M} \le \mathrm{tI}$.

The global minimum of this program is the scalar value `tmin` returned as first output argument by `feasp`. The LMI constraints are feasible if `tmin` ð 0 and strictly feasible if `tmin` < 0. If the problem is feasible but not strictly feasible, `tmin` is positive and very small. Some post-analysis may then be required to decide whether `xfeas` is close enough to feasible.

The optional argument `target` sets a target value for `tmin`. The optimization code terminates as soon as a value of t below this target is reached. The default value is `target` = 0.

Note that `xfeas` is a solution in terms of the decision variables and not in terms of the matrix variables of the problem. Use `dec2mat` to derive feasible values of the matrix variables from `xfeas`.

Control Parameters

The optional argument `options` gives access to certain control parameters for the optimization algorithm. This five-entry vector is organized as follows:

- `options(1)` is not used
- `options(2)` sets the maximum number of iterations allowed to be performed by the optimization procedure (100 by default)
- `options(3)` resets the *feasibility radius*. Setting `options(3)` to a value $R >$ 0 further constrains the decision vector $x = (x_1, \ldots, x_N)$ to lie within the ball

$$\sum_{i=1}^{N} x_i^2 < R^2$$

In other words, the Euclidean norm of `xfeas` should not exceed R. The feasibility radius is a simple means of controlling the magnitude of solutions. Upon termination, `feasp` displays the *f-radius saturation,* that is, the norm of the solution as a percentage of the feasibility radius R.

The default value is $R = 10^9$. Setting `options(3)` to a negative value activates the "flexible bound" mode. In this mode, the feasibility radius is initially set to 10^8, and increased if necessary during the course of optimization

- `options(4)` helps speed up termination. When set to an integer value $J > 0$, the code terminates if t did not decrease by more than one percent in relative terms during the last J iterations. The default value is 10. This parameter trades off speed vs. accuracy. If set to a small value (< 10), the code terminates quickly but without guarantee of accuracy. On the contrary, a large value results in natural convergence at the expense of a possibly large number of iterations.
- `options(5) = 1` turns off the trace of execution of the optimization procedure. Resetting `options(5)` to zero (default value) turns it back on.

Setting `option(i)` to zero is equivalent to setting the corresponding control parameter to its default value. Consequently, there is no need to redefine the entire vector when changing just one control parameter. To set the maximum number of iterations to 10, for instance, it suffices to type

```
options=zeros(1,5)   % default value for all parameters
options(2)=10
```

Memory Problems

When the least-squares problem solved at each iteration becomes ill conditioned, the `feasp` solver switches from Cholesky-based to QR-based linear algebra (see"Memory Problems" on page 9-74 for details). Since the QR mode typically requires much more memory, MATLAB may run out of memory and display the message

```
??? Error using ==> feaslv
Out of memory. Type HELP MEMORY for your options.
```

You should then ask your system manager to increase your swap space or, if no additional swap space is available, set `options(4) = 1`. This will prevent switching to QR and `feasp` will terminate when Cholesky fails due to numerical instabilities.

Example

Consider the problem of finding $P > I$ such that

$$A_1^T P + P A_1 < 0 \tag{9-8}$$

$$A_2^T P + P A_2 < 0 \tag{9-9}$$

$$A_3^T P + P A_3 < 0 \tag{9-10}$$

with data

$$A_1 = \begin{pmatrix} -1 & 2 \\ 1 & -3 \end{pmatrix}, \quad A_2 = \begin{pmatrix} -0.8 & 1.5 \\ 1.3 & -2.7 \end{pmatrix}, \quad A_3 = \begin{pmatrix} -1.4 & 0.9 \\ 0.7 & -2.0 \end{pmatrix}$$

This problem arises when studying the quadratic stability of the polytope of matrices $\mathrm{Co}\{A_1, A_2, A_3\}$ (see"Quadratic Stability" on page 3-6 for details).

To assess feasibility with `feasp`, first enter the LMIs (9-8)–(9-10) by:

```
setlmis([])
p = lmivar(1,[2 1])

lmiterm([1 1 1 p],1,a1,'s') % LMI #1
lmiterm([2 1 1 p],1,a2,'s') % LMI #2
lmiterm([3 1 1 p],1,a3,'s') % LMI #3
lmiterm([-4 1 1 p],1,1) % LMI #4: P
lmiterm([4 1 1 0],1) % LMI #4: I
lmis = getlmis
```

Then call `feasp` to find a feasible decision vector:

```
[tmin,xfeas] = feasp(lmis)
```

This returns `tmin = -3.1363`. Hence (9-8)–(9-10) is feasible and the dynamical system $\dot{x} = A(t)x$ is quadratically stable for $A(t) \in \mathrm{Co}\{A_1, A_2, A_3\}$.

To obtain a Lyapunov matrix P proving the quadratic stability, type

```
P = dec2mat(lmis,xfeas,p)
```

This returns

$$P = \begin{pmatrix} 270.8 & 126.4 \\ 126.4 & 155.1 \end{pmatrix}$$

It is possible to add further constraints on this feasibility problem. For instance, you can bound the Frobenius norm of P by 10 while asking `tmin` to be less than or equal to –1. This is done by

```
[tmin,xfeas] = feasp(lmis,[0,0,10,0,0],-1)
```

The third entry `10` of options sets the feasibility radius to 10 while the third argument `-1` sets the target value for `tmin`. This yields `tmin = -1.1745` and a matrix `P` with largest eigenvalue $\lambda_{\max}(P) = 9.6912$.

Reference

The feasibility solver `feasp` is based on Nesterov and Nemirovski's Projective Method described in

Nesterov, Y., and A. Nemirovski, *Interior Point Polynomial Methods in Convex Programming: Theory and Applications*, SIAM, Philadelphia, 1994.

Nemirovski, A., and P. Gahinet, "The Projective Method for Solving Linear Matrix Inequalities," *Proc. Amer. Contr. Conf.*, 1994, Baltimore, Maryland, pp. 840–844.

The optimization is performed by the C-MEX file `feaslv.mex`.

See Also

`mincx`, `gevp`, `dec2mat`

Purpose

Retrieve the *D, G* scaling matrices in problems

Syntax

```
D = getdg(ds,i)
G = getdg(gs,i)
```

Description

The function `mustab` computes the mixed-μ upper bound over a grid of frequencies `fs` and returns the corresponding *D, G* scalings compacted in two matrices `ds` and `gs`. The values of *D, G* at the frequency `fs(i)` are extracted from `ds` and `gs` by `getdg`.

See Also

`mustab`

getlmis

Purpose

Get the internal description of an LMI system

Syntax

```
lmisys = getlmis
```

Description

After completing the description of a given LMI system with `lmivar` and `lmiterm`, its internal representation `lmisys` is obtained with the command

```
lmisys = getlmis
```

This MATLAB representation of the LMI system can be forwarded to the LMI solvers or any other LMI-Lab function for subsequent processing.

See Also

`setlmis`, `lmivar`, `lmiterm`, `newlmi`

Purpose

Generalized eigenvalue minimization under LMI constraints

Syntax

```
[lopt,xopt] = gevp(lmisys,nlfc,options,linit,xinit,target)
```

Description

`gevp` solves the generalized eigenvalue minimization problem

Minimize λ subject to:

$$\mathcal{C}(x) < \mathcal{D}(x) \qquad \textbf{(9-11)}$$

$$0 < \mathcal{B}(x) \qquad \textbf{(9-12)}$$

$$\mathcal{A}(x) < \lambda\mathcal{B}(x) \qquad \textbf{(9-13)}$$

where $C(x) < D(x)$ and $A(x) < \lambda B(x)$ denote systems of LMIs. Provided that (9-11)–(9-12) are jointly feasible, `gevp` returns the global minimum `lopt` and the minimizing value `xopt` of the vector of decision variables x. The corresponding optimal values of the matrix variables are obtained with `dec2mat`.

The argument `lmisys` describes the system of LMIs (9-11)–(9-13) for $\lambda = 1$. The LMIs involving λ are called the *linear-fractional constraints* while (9-11)–(9-12) are regular LMI constraints. The number of linear-fractional constraints (9-13) is specified by `nlfc`. All other input arguments are optional. If an initial feasible pair (λ_0, x_0) is available, it can be passed to `gevp` by setting `linit` to λ_0 and `xinit` to x_0. Note that `xinit` should be of length `decnbr(lmisys)` (the number of decision variables). The initial point is ignored when infeasible. Finally, the last argument `target` sets some target value for λ. The code terminates as soon as it has found a feasible pair (λ, x) with λ ð `target`.

Caution

When setting up your `gevp` problem, be cautious to

- Always specify the linear-fractional constraints (9-13) *last* in the LMI system. `gevp` systematically assumes that the last `nlfc` LMI constraints are linear fractional
- Add the constraint $B(x) > 0$ or any other LMI constraint that enforces it (see Remark below). This positivity constraint is required for regularity and well-posedness of the optimization problem (see the discussion in “Well-Posedness Issues” on page 8-40).

Control Parameters

The optional argument `options` gives access to certain control parameters of the optimization code. In `gevp`, this is a five-entry vector organized as follows:

- `options(1)` sets the desired relative accuracy on the optimal value `lopt` (default = 10^{-2}).
- `options(2)` sets the maximum number of iterations allowed to be performed by the optimization procedure (100 by default).
- `options(3)` sets the feasibility radius. Its purpose and usage are as for `feasp`.
- `options(4)` helps speed up termination. If set to an integer value $J > 0$, the code terminates when the progress in λ over the last J iterations falls below the desired relative accuracy. By progress, we mean the amount by which λ decreases. The default value is 5 iterations.
- `options(5) = 1` turns off the trace of execution of the optimization procedure. Resetting `options(5)` to zero (default value) turns it back on.

Setting `option(i)` to zero is equivalent to setting the corresponding control parameter to its default value.

Example

Given

$$A_1 = \begin{pmatrix} -1 & 2 \\ 1 & -3 \end{pmatrix}, \quad A_2 = \begin{pmatrix} -0.8 & 1.5 \\ 1.3 & -2.7 \end{pmatrix}, \quad A_3 = \begin{pmatrix} -1.4 & 0.9 \\ 0.7 & -2.0 \end{pmatrix},$$

consider the problem of finding a single Lyapunov function $V(x) = x^T P x$ that proves stability of

$$\dot{x} = A_i x \quad (i = 1, 2, 3)$$

and maximizes the decay rate $-\frac{dV(x)}{dt}$. This is equivalent to minimizing α subject to

$$I < P \tag{9-14}$$

$$A_1^T P + P A_1 < \alpha P \tag{9-15}$$

$$A_2^T P + P A_2 < \alpha P \tag{9-16}$$

$$A_3^T P + P A_3 < \alpha P \tag{9-17}$$

To set up this problem for `gevp`, first specify the LMIs (9-15)–(9-17) with $\alpha = 1$:

```
setlmis([]);
p = lmivar(1,[2 1])

lemiterm([1 1 1 0],1) % P > I : I
lemiterm([ 1 1 1 p],1,1) % P > I : P
lemiterm([2 1 1 p],1,a1,'s') % LFC # 1 (lhs)
lemiterm([ 2 1 1 p],1,1) % LFC # 1 (rhs)
lemiterm([3 1 1 p],1,a2,'s') % LFC # 2 (lhs)
lemiterm([ 3 1 1 p],1,1) % LFC # 2 (rhs)
lemiterm([4 1 1 p],1,a3,'s') % LFC # 3 (lhs)
lemiterm([ 4 1 1 p],1,1) % LFC # 3 (rhs)
lmis = getlmis
```

Note that the linear fractional constraints are defined last as required. To minimize α subject to (9-15)–(9-17), call `gevp` by

```
[alpha,popt]=gevp(lmis,3)
```

This returns `alpha = -0.122` as optimal value (the largest decay rate is therefore 0.122). This value is achieved for

$$P = \begin{pmatrix} 5.58 & -8.35 \\ -8.35 & 18.64 \end{pmatrix}$$

Remark

Generalized eigenvalue minimization problems involve standard LMI constraints (9-11) and linear fractional constraints (9-13). For well-posedness, the positive definiteness of $B(x)$ must be enforced by adding the constraint $B(x) > 0$ to the problem. Although this could be done automatically from inside the code, this is not desirable for efficiency reasons. For instance, the set of constraints (9-12) may reduce to a single constraint as in the example above. In this case, the single extra LMI "$P > I$" is enough to enforce positivity of *all*

linear-fractional right-hand sides. It is therefore left to the user to devise the least costly way of enforcing this positivity requirement.

Reference

The solver `gevp` is based on Nesterov and Nemirovski's Projective Method described in

Nesterov, Y., and A. Nemirovski, *Interior Point Polynomial Methods in Convex Programming: Theory and Applications*, SIAM, Philadelphia, 1994.

The optimization is performed by the CMEX file `fpds.mex`.

See Also

`dec2mat`, `decnbr`, `feasp`, `mincx`

Purpose

Synthesis of gain-scheduled H_∞ controllers

Syntax

```
[gopt,pdK,R,S] = hinfgs(pdP,r,gmin,tol,tolred)
```

Description

Given an affine parameter-dependent plant

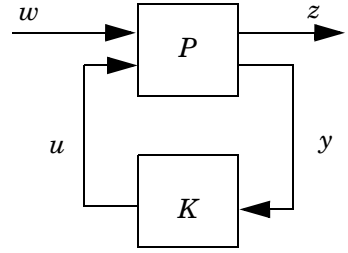
$$P\begin{cases} \dot{x} &= A(p)x + B_1(p)w + B_2u \\ z &= C_1(p)x + D_{11}(p)w + D_{12}u \\ y &= C_2x + D_{21}w + D_{22}u \end{cases}$$

where the time-varying parameter vector $p(t)$ ranges in a box and is measured in real time, `hinfgs` seeks an affine parameter-dependent controller

$$K\begin{cases} \dot{\zeta} &= A_K(p)\zeta + B_K(p)y \\ u &= C_K(p)\zeta + D_K(p)y \end{cases}$$

scheduled by the measurements of $p(t)$ and such that

- K stabilizes the closed-loop system



  for all admissible parameter trajectories $p(t)$
- K minimizes the closed-loop quadratic H_∞ performance from w to z.

The description `pdP` of the parameter-dependent plant P is specified with `psys` and the vector `r` gives the number of controller inputs and outputs (set `r=[`p_2`,m2]` if $y \in \mathbf{R}^{p2}$ and $u \in \mathbf{R}^{m_2}$). Note that `hinfgs` also accepts the polytopic model of P returned, e.g., by `aff2pol` or `sconnect`.

`hinfgs` returns the optimal closed-loop quadratic performance `gopt` and a polytopic description of the gain-scheduled controller `pdK`. To test if a

closed-loop quadratic performance γ is achievable, set the third input `gmin` to γ. The arguments `tol` and `tolred` control the required relative accuracy on `gopt` and the threshold for order reduction (see `hinflmi` for details). Finally, `hinfgs` also returns solutions R, S of the characteristic LMI system.

Controller Implementation

The gain-scheduled controller `pdK` is parametrized by $p(t)$ and characterized by the values K_{Π_j} of $\begin{pmatrix} A_K(p) & B_K(p) \\ C_K(p) & D_K(p) \end{pmatrix}$ at the corners ${}^3_j$ of the parameter box. The command

```
Kj = psinfo(pdK,'sys',j)
```

returns the j-th vertex controller K_{Π_j} while

```
pv = psinfo(pdP,'par')
vertx = polydec(pv)
Pj = vertx(:,j)
```

gives the corresponding corner ${}^3_j$ of the parameter box (`pv` is the parameter vector description).

The controller scheduling should be performed as follows. Given the measurements $p(t)$ of the parameters at time t,

3 Express $p(t)$ as a convex combination of the ${}^3_j$:

$$p(t) = \alpha_1 {}^3_1 + \ldots + \alpha_N {}^3_N, \qquad \alpha_j \geq 0, \qquad \sum_{i=1}^{N} \alpha_j = 1$$

This convex decomposition is computed by `polydec`.

4 Compute the controller state-space matrices at time t as the convex combination of the vertex controllers K_{Π_j}:

$$\begin{pmatrix} A_K(t) & B_K(t) \\ C_K(t) & D_K(t) \end{pmatrix} = \sum_{i=1}^{N} \alpha_j K_{\Pi_j}.$$

5 Use $A_K(t)$, $B_K(t)$, $C_K(t)$, $D_K(t)$ to update the controller state-space equations.

Reference

Apkarian, P., P. Gahinet, and G. Becker, "Self-Scheduled H_∞ Control of Linear Parameter-Varying Systems," submitted to *Automatica*, October 1995.

Becker, G., Packard, P., "Robust Performance of Linear-Parametrically Varying Systems Using Parametrically-Dependent Linear Feedback," *Systems and Control Letters*, 23 (1994), pp. 205–215.

Packard, A., "Gain Scheduling via Linear Fractional Transformations," *Syst. Contr. Letters*, 22 (1994), pp. 79–92.

Example

See "Design Example" on page 7-10.

See Also

`psys`, `pvec`, `pdsimul`, `polydec`, `hinflmi`

hinflmi

Purpose

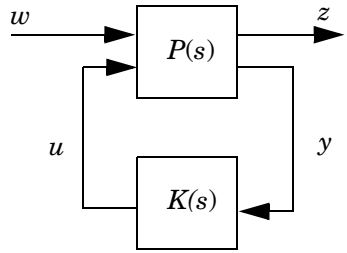
LMI-based H_∞ synthesis for continuous-time plants

Syntax

```
gopt = hinflmi(P,r)
[gopt,K] = hinflmi(P,r)
[gopt,K,x1,x2,y1,y2] = hinflmi(P,r,g,tol,options)
```

Description

`hinflmi` computes an internally stabilizing controller $K(s)$ that minimizes the closed-loop H_∞ performance in the control loop



The H_∞ performance is the RMS gain of the closed-loop transfer function $F(P, K)$ from w to z. This function implements the LMI-based approach to H_∞ synthesis. The `SYSTEM` matrix `P` contains a state-space realization of the plant $P(s)$ and the row vector `r` specifies the number of measurements and controls (set `r = [`p_2 `m2]` when $y \in \mathbf{R}^{p_2}$ and $u \in \mathbf{R}^{m_2}$).

The optimal H_∞ performance `gopt` and an optimal H_∞ controller `K` are returned by

```
[gopt,K] = hinflmi(P,r)
```

Alternatively, a suboptimal controller K that guarantees $\|F(P, K)\|_\infty <$ `g` is obtained with the syntax

```
[gopt,K] = hinflmi(P,r,g)
```

The optional argument `tol` specifies the relative accuracy on the computed optimal performance `gopt`. The default value is 10^{-2}. Finally,

```
[gopt,K,x1,x2,y1,y2] = hinflmi(P,r)
```

also returns solutions R = `x1` and S = `y1` of the characteristic LMIs for γ = `gopt` (see "Practical Considerations" on page 6-18 for details).

Control Parameters

The optional argument `options` resets the following control parameters:

- `options(1)` is valued in [0, 1] with default value 0. Increasing its value reduces the norm of the LMI solution R. This typically yields controllers with slower dynamics in singular problems (rank-deficient D_{12}). This also improves closed-loop damping when $p_{12}(s)$ has $j\omega$-axis zeros.
- `options(2)`: same as `options(1)` for S and singular problems with rank-deficient D_{21}
- when `options(3)` is set to $\varepsilon > 0$, reduced-order synthesis is performed whenever

  $\rho(R^{-1} S^{-1}) \geq (1 - \varepsilon)\ \texttt{gopt}^2$

The default value is $\varepsilon = 10^{-3}$

Remark

The LMI-based approach is directly applicable to singular plants without preliminary regularization.

Reference

Scherer, C., "H_∞ Optimization without Assumptions on Finite or Infinite Zeros," *SIAM J. Contr. Opt.*, 30 (1992), pp. 143–166.

Gahinet, P., and P. Apkarian, "A Linear Matrix Inequality Approach to H_∞ Control," *Int. J. Robust and Nonlinear Contr.*, 4 (1994), pp. 421–448.

Gahinet, P., "Explicit Controller Formulas for LMI-based H_∞ Synthesis," in *Proc. Amer. Contr. Conf.*, 1994, pp. 2396–2400.

Iwasaki, T., and R.E. Skelton, "All Controllers for the General H_∞ Control Problem: LMI Existence Conditions and State-Space Formulas," *Automatica*, 30 (1994), pp. 1307–1317.

See Also

`hinfric`, `hinfmix`, `dhinflmi`, `ltisys`, `slft`

Purpose

Mixed H_2/H_∞ synthesis with pole placement constraints

Syntax

```
[gopt,h2opt,K,R,S] = hinfmix(P,r,obj,region,dkbnd,tol)
```

Description

`hinfmix` performs multi-objective output-feedback synthesis. The control problem is sketched in Figure 9.1.

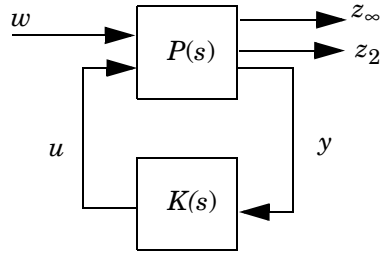


Figure 9-1: Mixed H_2/H_∞ synthesis

If $T_\infty(s)$ and $T_2(s)$ denote the closed-loop transfer functions from w to z_∞ and z_2, respectively, `hinfmix` computes a suboptimal solution of the following synthesis problem:

Design an LTI controller $K(s)$ that minimizes the mixed H_2/H_∞ criterion

$$\alpha\|T_\infty\|_\infty^2 + \beta\|T_2\|_2^2$$

subject to

- $\|T_\infty\|_\infty < \gamma_0$
- $\|T_2\|_2 < \nu_0$
- The closed-loop poles lie in some prescribed LMI region D.

Recall that $\|\cdot\|_\infty$ and $\|\cdot\|_2$ denote the H_∞ norm (RMS gain) and H_2 norm of transfer functions. More details on the motivations and statement of this problem can be found in “Multi-Objective H• Synthesis” on page 5-15.

On input, `P` is the `SYSTEM` matrix of the plant $P(s)$ and `r` is a three-entry vector listing the lengths of z_2, y, and u. Note that z_∞ and/or z_2 can be empty. The

four-entry vector `obj` = [γ_0, ν_0, α, β] specifies the H_2/H_∞ constraints and trade-off criterion, and the remaining input arguments are optional:

- `region` specifies the LMI region for pole placement (the default `region = []` is the open left-half plane). Use `lmireg` to interactively build the LMI region description `region`
- `dkbnd` is a user-specified bound on the norm of the controller feedthrough matrix D_K (see the definition in "LMI Formulation" on page 5-16). The default value is 100. To make the controller $K(s)$ strictly proper, set `dkbnd = 0`.
- `tol` is the required relative accuracy on the optimal value of the trade-off criterion (the default is 10^{-2}).

The function `hinfmix` returns guaranteed H_∞ and H_2 performances `gopt` and `h2opt` as well as the `SYSTEM` matrix `K` of the LMI-optimal controller. You can also access the optimal values of the LMI variables *R, S* via the extra output arguments `R` and `S`.

A variety of mixed and unmixed problems can be solved with `hinfmix` (see list in "The Function hinfmix" on page 5-20). In particular, you can use `hinfmix` to perform pure pole placement by setting `obj = [0 0 0 0]`. Note that both z_∞ and z_2 can be empty in such case.

Example

See the example in"H• Synthesis" on page 5-10.

Reference

Chilali, M., and P. Gahinet, "H_∞ Design with Pole Placement Constraints: An LMI Approach," to appear in *IEEE Trans. Aut. Contr.*, 1995.

Scherer, C., "Mixed H_2 H_∞ Control," to appear in *Trends in Control: A European Perspective*, volume of the special contributions to the ECC 1995.

See Also

`lmireg`, `hinflmi`, `msfsyn`

hinfpar

Purpose

Extract the state-space matrices A, B_1, B_2, . . .from the SYSTEM matrix representation of an H_∞ plant

Syntax

```
[a,b1,b2,c1,c2,d11,d12,d21,d22] = hinfpar(P,r)
b1 = hinfpar(P,r,'b1')
```

Description

In H_∞ control problems, the plant $P(s)$ is specified by its state-space equations

$$\dot{x} = Ax + B_1 w + B_2 u$$

$$z = C_1 x + D_{11} w + D_{12} u$$

$$y = C_2 x + D_{21} w + D_{22} u$$

or their discrete-time counterparts. For design purposes, this data is stored in a single SYSTEM matrix P created by

```
P = ltisys(a,[b1 b2],[c1;c2],[d11 d12;d21 d22])
```

The function hinfpar retrieves the state-space matrices A, B_1, B_2, . . . from P. The second argument r specifies the size of the D_{22} matrix. Set `r = [ny,nu]` where ny is the number of measurements (length of y) and nu is the number of controls (length of u).

The syntax

```
hinfpar(P,r,'b1')
```

can be used to retrieve just the B_1 matrix. Here the third argument should be one of the strings `'a', 'b1', 'b2',....`

See Also

hinfric, hinflmi, ltisys

Purpose

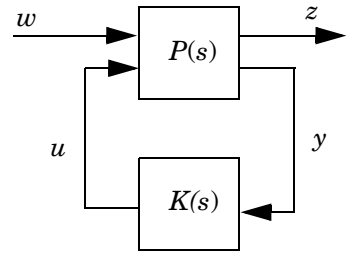
Riccati-based H_∞ synthesis for continuous-time plants

Syntax

```
gopt = hinfric(P,r) [gopt,K] = hinfric(P,r)
   [gopt,K,x1,x2,y1,y2,Preg] = hinfric(P,r,gmin,gmax,...
   tol,options)
```

Description

`hinfric` computes an H_∞ controller $K(s)$ that internally stabilizes the control loop



and minimizes the RMS gain (H_∞ performance) of the closed-loop transfer function T_{wz}. The function `hinfric` implements the Riccati-based approach.

The `SYSTEM` matrix `P` contains a state-space realization

$$\dot{x} = Ax + B_1 w + B_2 u$$

$$z = C_1 x + D_{11} w + D_{12} u$$

$$y = C_2 x + D_{21} w + D_{22} u$$

of the plant $P(s)$ and is formed by the command

```
P = ltisys(a,[b1 b2],[c1;c2],[d11 d12;d21 d22])
```

The row vector `r` specifies the dimensions of D_{22}. Set `r = [`p_2 `m2]` when $y \in \mathbf{R}^{p2}$ and $u \in \mathbf{R}^{m_2}$.

To minimize the H_∞ performance, call `hinfric` with the syntax

```
gopt = hinfric(P,r)
```

This returns the smallest achievable RMS gain from w to z over all stabilizing controllers $K(s)$. An H_∞-optimal controller $K(s)$ with performance `gopt` is computed by

```
[gopt,K] = hinfric(P,r)
```

The H_∞ performance can also be minimized within prescribed bounds `gmin` and `gmax` and with relative accuracy `tol` by typing

```
[gopt,K] = hinfric(P,r,gmin,gmax,tol)
```

Setting both `gmin` and `gmax` to $\gamma > 0$ tests whether the H_∞ performance γ is achievable and returns a controller $K(s)$ such that $\|T_{wz}\|_\infty < \gamma$ when γ is achievable. Setting `gmin` or `gmax` to zero is equivalent to specifying no lower or upper bound.

Finally, the full syntax

```
[gopt,K,x1,x2,y1,y2,Preg] = hinfric(P,r)
```

also returns the solutions X_∞ = `x2/x1` and Y_∞ = `y2/y1` of the H_∞ Riccati equations for γ = `gopt` , as well as the regularized plant `Preg` used to compute the controller in singular problems.

Given the controller `SYSTEM` matrix `K` returned by `hinfric`, a realization of the closed-loop transfer function from w to z is given by

```
Twz = slft(P,K)
```

Control Parameters

`hinfric` automatically regularizes singular H_∞ problems and computes reduced-order controllers when the matrix $I - \gamma^{-2}X_\infty Y_\infty$ is nearly singular. The optional input `options` gives access to the control parameters that govern the regularization and order reduction:

- `options(1)` controls the amount of regularization used when D_{12} or D_{21} is rank-deficient. This tuning parameter is valued in [0,1] with default value 0. Increasing `options(1)` augments the amount of regularization. This generally results in less control effort and in controller state-space matrices of smaller norm, possibly at the expense of a higher closed-loop RMS gain.
- `options(2)` controls the amount of regularization used when $P_{12}(s)$ or $P_{21}(s)$ has $j\omega$-axis zero(s). Such problems are regularized by replacing A by $A + \varepsilon I$ where ε is a small positive number. Increasing `options(2)` increases ε and improves the closed-loop damping. Beware that making ε too large may

destroy stabilizability or detectability when the plant $P(s)$ includes shaping filters with poles close to the imaginary axis. In such cases the optimal H_∞ performance will be infinite (see "Practical Considerations" on page 6-18 for more details).

- when `options(3)` is set to $\varepsilon > 0$, reduced-order synthesis is performed whenever

  $\rho(X_\infty Y_\infty)$ Š $(1 - \varepsilon)$ `gopt`^{2}

  The default value is $\varepsilon = 10^{-3}$

Remark

Setting `gmin = gmax = Inf` computes the optimal LQG controller.

Example

See the example in "H• Synthesis" on page 5-10.

Reference

Doyle, J.C., Glover, K., Khargonekar, P., and Francis, *B*., "State-Space Solutions to Standard H_2 and H_∞ Control Problems," *IEEE Trans. Aut. Contr*., AC–34 (1989), pp. 831–847.

Gahinet, P., and A.J. Laub, "Reliable Computation of flopt in Singular H_∞ Control," in *Proc. Conf. Dec. Contr*., Lake Buena Vista, Fl., 1994, pp. 1527–1532.

Gahinet, P., "Explicit Controller Formulas for LMI-based H_∞ Synthesis," in *Proc. Amer. Contr. Conf*., 1994, pp. 2396–2400.

See Also

`hinflmi`, `hinfmix`, `dhinfric`, `ltisys`, `slft`

lmiedit

Purpose

Specify or display systems of LMIs as MATLAB expressions

Syntax

```
lmiedit
```

Description

`lmiedit` is a graphical user interface for the symbolic specification of LMI problems. Typing `lmiedit` calls up a window with two editable text areas and various pushbuttons. To specify an LMI system,

1. Give it a name (top of the window).
2. Declare each matrix variable (name and structure) in the upper half of the window. The structure is characterized by its type (`S` for symmetric block diagonal, `R` for unstructured, and `G` for other structures) and by an additional structure matrix similar to the second input argument of `lmivar`. Please use one line per matrix variable in the text editing areas.
3. Specify the LMIs as MATLAB expressions in the lower half of the window. An LMI can stretch over several lines. However, do not specify more than one LMI per line.

Once the LMI system is fully specified, you can perform the following operations by pressing the corresponding pushbutton:

- Visualize the sequence of `lmivar`/`lmiterm` commands needed to describe this LMI system (`view commands` buttons)
- Conversely, display the symbolic expression of the LMI system produced by a particular sequence of `lmivar`/`lmiterm` commands (click the `describe...` buttons)
- Save the symbolic description of the LMI system as a MATLAB string (`save` button). This description can be reloaded later on by pressing the `load` button
- Read a sequence of `lmivar`/`lmiterm` commands from a file (`read` button). The matrix expression of the LMI system specified by these commands is then displayed by clicking on `describe the LMIs...`
- Write in a file the sequence of `lmivar`/`lmiterm` commands needed to specify a particular LMI system (`write` button)
- Generate the internal representation of the LMI system by pressing `create`. The result is written in a MATLAB variable with the same name as the LMI system

Remark

Editable text areas have built-in scrolling capabilities. To activate the scroll mode, click in the text area, maintain the mouse button down, and move the mouse up or down. The scroll mode is only active when all visible lines have been used.

Example

An example and some limitations of `lmiedit` can be found in "The LMI Editor lmiedit" on page 8-16.

See Also

`lmivar`, `lmiterm`, `newlmi`, `lmiinfo`

lmiinfo

Purpose

Interactively retrieve information about the variables and term content of LMIs

Syntax

```
lmiinfo(lmisys)
```

Description

`lmiinfo` provides qualitative information about the system of LMIs `lmisys`. This includes the type and structure of the matrix variables, the number of diagonal blocks in the inner factors, and the term content of each block.

`lmiinfo` is an interactive facility where the user seeks specific pieces of information. General LMIs are displayed as

```
N' * L(x) * N < M' * R(x) * M
```

where `N,M` denote the outer factors and `L,R` the left and right inner factors. If the outer factors are missing, the LMI is simply written as

```
L(x) < R(x)
```

If its right-hand side is zero, it is displayed as

```
N' * L(x) * N < 0
```

Information on the block structure and term content of `L(x)` and `R(x)` is also available. The term content of a block is symbolically displayed as

```
C1 + A1*X2*B1 + B1'*X2*A1' + a2*X1 + x3*Q1
```

with the following conventions:

- `X1, X2, x3` denote the problem variables. Upper-case `X` indicates matrix variables while lower-case `x` indicates scalar variables. The labels 1,2,3 refer to the first, second, and third matrix variable in the order of declaration.
- `Cj` refers to constant terms. Special cases are `I` and `I` (`I` = identity matrix).
- `Aj, Bj` denote the left and right coefficients of variable terms. Lower-case letters such as `a2` indicate a scalar coefficient.
- `Qj` is used exclusively with scalar variables as in `x3*Q1`.

The index `j` in `Aj, Bj, Cj, Qj` is a dummy label. Hence `C1` may appear in several blocks or several LMIs without implying any connection between the corresponding constant terms. Exceptions to this rule are the notations

`A1*X2*A1'` and `A1*X2*B1 + B1'*X2'*A1'` which indicate symmetric terms and symmetric pairs in diagonal blocks.

Example

Consider the LMI

$$0 \ð \begin{pmatrix} -2X + A^T Y B + B^T Y^T A + I & XC \\ C^T X & -zI \end{pmatrix}$$

where the matrix variables are X of Type 1, Y of Type 2, and z scalar. If this LMI is described in `lmis`, information about X and the LMI block structure can be obtained as follows:

```
lmiinfo(lmis)

                  LMI ORACLE
                 ------

This is a system of 1 LMI with 3 variable matrices

Do you want information on
   (v) matrix variables (l) LMIs (q) quit

?> v

Which variable matrix (enter its index k between 1 and 3) ? 1
   X1 is a 2x2 symmetric block diagonal matrix
     its (1,1)-block is a full block of size 2

                      -------

This is a system of 1 LMI with 3 variable matrices
Do you want information on
   (v) matrix variables (l) LMIs (q) quit

?> l

Which LMI (enter its number k between 1 and 1) ? 1

   This LMI is of the form
```

```
            0 < R(x)
where the inner factor(s) has 2 diagonal block(s)


Do you want info on the right inner factor ?

   (w) whole factor (b) only one block
   (o) other LMI (t) back to top level

?> w

Info about the right inner factor

   block (1,1) : I + a1*X1 + A2*X2*B2 + B2'*X2'*A2'

   block (2,1) : A3*X1

   block (2,2) : x3*A4

   (w) whole factor (b) only one block
   (o) other LMI (t) back to top level

                     ------

This is a system of 1 LMI with 3 variable matrices

Do you want information on
   (v) matrix variables (l) LMIs (q) quit

?> q

It has been a pleasure serving you!
```

Note that the prompt symbol is `?>` and that answers are either indices or letters. All blocks can be displayed at once with option `(w)`, or you can prompt for specific blocks with option `(b)`.

Remark

`lmiinfo` does not provide access to the numerical value of LMI coefficients.

See Also

`decinfo`, `lminbr`, `matnbr`, `decnbr`

Purpose

Return the number of LMIs in an LMI system

Syntax

```
k = lminbr(lmisys)
```

Description

`lminbr` returns the number `k` of linear matrix inequalities in the LMI problem described in `lmisys`.

See Also

`lmiinfo`, `matnbr`

lmireg

Purpose

Specify LMI regions for pole placement purposes

Syntax

```
region = lmireg
   region = lmireg(reg1,reg2,...)
```

Description

`lmireg` is an interactive facility to specify the LMI regions involved in multi-objective H_∞ synthesis with pole placement constraints (see `msfsyn` and `hinfmix`). Recall that an LMI region is any convex subset D of the complex plane that can be characterized by an LMI in z and $\bar{z}$, i.e.,

$$D = \{z \in \mathbf{C} : L + Mz + M^T\bar{z} < 0\}$$

for some fixed real matrices M and $L = L^T$. This class of regions encompasses half planes, strips, conic sectors, disks, ellipses, and any intersection of the above.

Calling `lmireg` without argument starts an interactive query/answer session where you can specify the region of your choice. The matrix `region` = [*L, M*] is returned upon termination. This matrix description of the LMI region can be passed directly to `msfsyn` or `hinfmix` for synthesis purposes.

The function `lmireg` can also be used to intersect previously defined LMI regions `reg1, reg2,...`. The output `region` is then the [*L, M*] description of the intersection of these regions.

See Also

`msfsyn`, `hinfmix`

Purpose

Specify the term content of LMIs

Syntax

```
lmiterm(termID,A,B,flag)
```

Description

`lmiterm` specifies the term content of an LMI one term at a time. Recall that *LMI term* refers to the elementary additive terms involved in the block-matrix expression of the LMI. Before using `lmiterm`, the LMI description must be initialized with `setlmis` and the matrix variables must be declared with `lmivar`. Each `lmiterm` command adds one extra term to the LMI system currently described.

LMI terms are one of the following entities:

- outer factors
- constant terms (fixed matrices)
- variable terms AXB or AX^TB where X is a matrix variable and A and B are given matrices called the term coefficients.

When describing an LMI with several blocks, remember to specify **only the terms in the blocks on or below the diagonal** (or equivalently, only the terms in blocks on or above the diagonal). For instance, specify the blocks (1,1), (2,1), and (2,2) in a two-block LMI.

In the calling of `limterm`, `termID` is a four-entry vector of integers specifying the term location and the matrix variable involved.

$$\texttt{termID (1)} = \begin{cases} \texttt{+p} \\ \texttt{-p} \end{cases}$$

where positive p is for terms on the *left-hand side* of the p-th LMI and negative p i s for terms on the *right-hand side* of the p-th LMI.

Recall that, by convention, the left-hand side always refers to the smaller side of the LMI. The index p is relative to the order of declaration and corresponds to the identifier returned by `newlimi`.

$$\texttt{termID (2:3)} = \begin{cases} \texttt{[0, 0]} & \text{for outer factors} \\ \texttt{[i, j]} & \text{for terms in the } (i,j)\text{-th} \\ & \quad \text{block of the left or right inner factor} \end{cases}$$

$$\texttt{termID (4)} = \begin{cases} \texttt{0} & \text{for outer factors} \\ \texttt{x} & \text{for variable terms } AXB \\ -\texttt{x} & \text{for variable terms } AX^TB \end{cases}$$

where `x` is the identifier of the matrix variable `X` as returned by `lmivar`. The arguments `A` and `B` contain the numerical data and are set according to:

Type of Term	A	B
outer factor N	matrix value of N	omit
constant term C	matrix value of C	omit
variable term AXB or AX^TB	matrix value of A (1 if A is absent)	matrix value of B (1 if B is absent)

Note that identity outer factors and zero constant terms need not be specified.

The extra argument `flag` is optional and concerns only conjugated expressions of the form

$$(AXB) + (AXB)^T = AXB + B^TX^{(T)}A^T$$

in *diagonal blocks*. Setting `flag = 's'` allows you to specify such expressions with a single `lmiterm` command. For instance,

```
lmiterm([1 1 1 X],A,1,'s')
```

adds the symmetrized expression $AX + X^TA^T$ to the (1,1) block of the first LMI and summarizes the two commands

```
lmiterm([1 1 1 X],A,1)
lmiterm([1 1 1  X],1,A')
```

Aside from being convenient, this shortcut also results in a more efficient representation of the LMI.

Example

Consider the LMI

$$\begin{pmatrix} 2AX_2A^T - x_3E + DD^T & B^TX_1 \\ X_1^TB & -I \end{pmatrix} < M^T \begin{pmatrix} CX_1C^T + CX_1^TC^T & 0 \\ 0 & -fX_2 \end{pmatrix} M$$

where X_1, X_2 are matrix variables of Types 2 and 1, respectively, and x_3 is a scalar variable (Type 1).

After initializing the LMI description with `setlmis` and declaring the matrix variables with `lmivar`, the terms on the left-hand side of this LMI are specified by:

```
lmiterm([1 1 1 X2],2*A,A') % 2*A*X2*A'
lmiterm([1 1 1 x3],-1,E) % -x3*E
lmiterm([1 1 1 0],D*D') % D*D'
lmiterm([1 2 1 -X1],1,B) % X1'*B
lmiterm([1 2 2 0],-1) % -I
```

Here `X1, X2, X3` should be the variable identifiers returned by `lmivar`.

Similarly, the term content of the right-hand side is specified by:

```
lmiterm([-1 0 0 0],M) % outer factor M
lmiterm([-1 1 1 X1],C,C','s') % C*X1*C'+C*X1'*C'
lmiterm([-1 2 2 X2],-f,1) % -f*X2
```

Note that $CX_1C^T + CX_1^TC^T$ is specified by a single `lmiterm` command with the flag `'s'` to ensure proper symmetrization.

See Also

`setlmis`, `lmivar`, `getlmis`, `lmiedit`, `newlmi`

lmivar

Purpose

Specify the matrix variables in an LMI problem

Syntax

```
X = lmivar(type,struct)
[X,n,sX] = lmivar(type,struct)
```

Description

`lmivar` defines a new matrix variable X in the LMI system currently described. The optional output `X` is an identifier that can be used for subsequent reference to this new variable.

The first argument `type` selects among available types of variables and the second argument `struct` gives further information on the structure of X depending on its type. Available variable types include:

type=1: Symmetric matrices with a block-diagonal structure. Each diagonal block is either full (arbitrary symmetric matrix), scalar (a multiple of the identity matrix), or identically zero.

If X has R diagonal blocks, `struct` is an R-by-2 matrix where

- `struct(r,1)` is the size of the `r`-th block
- `struct(r,2)` is the type of the `r`-th block (`1` for full, `0` for scalar, `1` for zero block).

type=2: Full m-by-n rectangular matrix. Set `struct = [m,n]` in this case.

type=3: Other structures. With Type 3, each entry of X is specified as zero or $\pm x_n$ where x_n is the n-th decision variable.

Accordingly, `struct` is a matrix of the same dimensions as X such that

- `struct(i,j)=0` if $X(i, j)$ is a hard zero
- `struct(i,j)=n` if $X(i, j) = x_n$

  `struct(i,j)= n` if $X(i, j) = -x_n$

Sophisticated matrix variable structures can be defined with Type 3. To specify a variable X of Type 3, first identify how many *free independent entries* are involved in X. These constitute the set of decision variables associated with X. If the problem already involves n decision variables, label the new free variables as $x_{n+1}, \ldots, x_{n+p}$. The structure of X is then defined in terms of $x_{n+1}, \ldots, x_{n+p}$ as indicated above. To help specify matrix variables of Type 3, `lmivar` optionally returns two extra outputs: (1) the total number `n` of scalar decision variables used so far and (2) a matrix `sX` showing the entry-wise

dependence of X on the decision variables $x_1, \ldots, x_n$. For more details, see Example 2 below and "Structured Matrix Variables" on page 8-33.

Example 1

Consider an LMI system with three matrix variables X_1, X_2, X_3 such that

- X_1 is a 3×3 symmetric matrix (unstructured),
- X_2 is a 2×4 rectangular matrix (unstructured),
- X_3 =

$$\begin{pmatrix} \Delta & 0 & 0 \\ 0 & \delta_1 & 0 \\ 0 & 0 & \delta_2 I_2 \end{pmatrix}$$

  where Δ is an arbitrary 5×5 symmetric matrix, δ_1 and δ_2 are scalars, and I_2 denotes the identity matrix of size 2.

These three variables are defined by

```
setlmis([])
X1 = lmivar(1,[3 1]) % Type 1
X2 = lmivar(2,[2 4]) % Type 2 of dim. 2x4
X3 = lmivar(1,[5 1;1 0;2 0]) % Type 1
```

The last command defines X_3 as a variable of Type 1 with one full block of size 5 and two scalar blocks of sizes 1 and 2, respectively.

Example 2

Combined with the extra outputs `n` and `sX` of `lmivar`, Type 3 allows you to specify fairly complex matrix variable structures. For instance, consider a matrix variable X with structure

$$X = \begin{pmatrix} X_1 & 0 \\ 0 & X_2 \end{pmatrix}$$

where X_1 and X_2 are 2-by-3 and 3-by-2 rectangular matrices, respectively. You can specify this structure as follows:

1 Define the rectangular variables X_1 and X_2 by

```
setlmis([])
[X1,n,sX1] = lmivar(2,[2 3])
```

```
[X2,n,sX2] = lmivar(2,[3 2])
```

The outputs `sX1` and `sX2` give the decision variable content of X_1 and X_2:

```
sX1

sX1 =
   1   2   3
   4   5   6

sX2

sX2 =
   7   8
   9   10
   11  12
```

For instance, `sX2(1,1)=7` means that the (1,1) entry of X_2 is the seventh decision variable.

2 Use Type 3 to specify the matrix variable X and define its structure in terms of those of X_1 and X_2 :

```
[X,n,sX] = lmivar(3,[sX1,zeros(2);zeros(3),sX2])
```

The resulting variable `X` has the prescribed structure as confirmed by

```
sX

sX =
   1   2   3   0   0
   4   5   6   0   0
   0   0   0   7   8
   0   0   0   9   10
   0   0   0   11  12
```

See Also

`setlmis`, `lmiterm`, `getlmis`, `lmiedit`, `skewdec`, `symdec`, `delmvar`, `setmvar`

Purpose

Extract the state-space realization of a linear system from its SYSTEM matrix description

Syntax

```
[a,b,c,d,e] = ltiss(sys)
```

Description

Given a SYSTEM matrix sys created with ltisys, this function returns the state-space matrices *A, B, C, D, E*.

Example

The system $G(s) = \frac{-2}{s+1}$ is specified in transfer function form by

```
sys = ltisys('tf', 2,[1 1])
```

A realization of $G(s)$ is then obtained as

```
[a,b,c,d] = ltiss(sys)

a =
    -1

b =
     1

c =
    -2

d =
     0
```

Important

If the output argument e is not supplied, ltiss returns the realization

$(E^{-1}A, E^{-1}B, C, D)$.

While this is immaterial for systems with $E = I$, this should be remembered when manipulating descriptor systems.

See Also

ltisys, ltitf, sinfo

ltisys

Purpose

Pack the state-space realization (*A, B, C, D, E*) into a single `SYSTEM` matrix

Syntax

```
sys = ltisys(a)
sys = ltisys(a,e)
sys = ltisys(a,b,c)
sys = ltisys(a,b,c,d)
sys = ltisys(a,b,c,d,e)
sys = ltisys('tf',num,den)
```

Description

For continuous-time systems,

$$E\dot{x} = Ax + Bu, \qquad y = Cx + Du$$

or discrete-time systems

$$Ex_{k+1} = Ax_k + Bu_k, \quad y_k = Cx_k + Du_k,$$

`ltisys` stores *A, B, C, D, E* into a single matrix `sys` with structure

$$\left[\begin{array}{cc|c} A + j(E - I) & B & \begin{matrix} n \\ 0 \\ \vdots \\ 0 \end{matrix} \\ C & D & \\ \hline 0 & & -\mathrm{Inf} \end{array}\right]$$

The upper right entry n gives the number of states (i.e., $A \in \mathrm{R}^{n \times n}$). This data structure is called a `SYSTEM` matrix. When omitted, `d` and `e` are set to the default values $D = 0$ and $E = I$.

The syntax `sys = ltisys(a)` specifies the autonomous system $\dot{x} = Ax$ while `sys = ltisys(a,e)` specifies $E\dot{x} = Ax$. Finally, you can specify SISO systems by their transfer function

$$G(s) = \frac{n(s)}{d(s)}$$

The syntax is then

```
sys = ltisys('tf',num,den)
```

where `num` and `den` are the vectors of coefficients of the polynomials $n(s)$ and $d(s)$ in descending order.

Examples

The SYSTEM matrix representation of the descriptor system

$$\begin{cases} \varepsilon \dot{x} = -x + 2u \\ y = x \end{cases}$$

is created by

```
sys = ltisys(-1,2,1,0,eps)
```

Similarly, the SISO system with transfer function $G(s) = \frac{s+4}{s+7}$ is specified by

```
sys = ltisys('tf',[1 4],[1 7])
```

See Also

ltiss, ltitf, sinfo

ltitf

Purpose

Compute the transfer function of a SISO system

Syntax

```
[num,den] = ltitf(sys)
```

Description

Given a single-input/single-output system defined in the `SYSTEM` matrix `sys`, `ltitf` returns the numerator and denominator of its transfer function

$$G(s) = \frac{n(s)}{d(s)}$$

The output arguments `num` and `den` are vectors listing the coefficients of the polynomials $n(s)$ and $d(s)$ in descending order. An error is issued if `sys` is not a SISO system.

Example

The transfer function of the system

$$\dot{x} = -x + 2u, \qquad y = x - 1$$

is given by

```
sys = ltisys(-1,2,1,-1)
[n,d] = ltitf(sys)

n =
    -1   1

d =
     1    1
```

The vectors `n` and `d` represent the function

$$G(s) = \frac{-s+1}{s+1}$$

See Also

`ltisys`, `ltiss`, `sinfo`

Purpose

Graphical specification of the shaping filters

Syntax

```
magshape
```

Description

`magshape` is a graphical user interface to construct the shaping filters needed in loop-shaping design. Typing `magshape` brings up a window where the magnitude profile of each filter can be specified graphically with the mouse. Once a profile is sketched, `magshape` computes an interpolating SISO filter and returns its state-space realization. Several filters can be specified in the same `magshape` window as follows:

1. First enter the filter names after `Filter names`. This tells `magshape` to write the `SYSTEM` matrices of the interpolating filters in MATLAB variables with these names.
2. Click on the button showing the name of the filter you are about to specify. To sketch its profile, use the mouse to specify a few characteristic points that define the asymptotes and the slope changes. For accurate fitting by a rational filter, make sure that the slopes are integer multiples of ±20 dB/dec.
3. Specify the filter order after `filter order` and click on the `fit data` button to perform the fitting. The magnitude response of the computed filter is then drawn as a solid line and its `SYSTEM` matrix is written in the MATLAB variable with the same name.
4. Adjustments can be made by moving or deleting some of the specified points. Click on `delete point` or move `point` to activate those modes. You can also increase the order for a better fit. Press `fit data` to recompute the interpolating filter.

If some of the filters are already defined in the MATLAB environment, their magnitude response is plotted when the button bearing their name is clicked. This allows for easy modification of existing filters.

Remark

See `sderiv` for the specification of nonproper shaping filters with a derivative action.

Acknowledgment

The authors are grateful to Prof. Gary Balas and Prof. Andy Packard for providing the cepstrum algorithm used for phase reconstruction.

See Also

`sderiv`, `frfit`, `mrfit`

matnbr

Purpose

Return the number of matrix variables in a system of LMIs

Syntax

```
K = matnbr(lmisys)
```

Description

`matnbr` returns the number `K` of matrix variables in the LMI problem described by `lmisys`.

See Also

`decnbr`, `lmiinfo`, `decinfo`

Purpose

Return the vector of decision variables corresponding to particular values of the matrix variables

Syntax

```
decvec = mat2dec(lmisys,X1,X2,X3,...)
```

Description

Given an LMI system `lmisys` with matrix variables $X_1, \ldots, X_K$ and given values `X1,...,Xk` of $X_1, \ldots, X_K$, `mat2dec` returns the corresponding value `decvec` of the vector of decision variables. Recall that the decision variables are the independent entries of the matrices $X_1, \ldots, X_K$ and constitute the free scalar variables in the LMI problem.

This function is useful, for example, to initialize the LMI solvers `mincx` or `gevp`. Given an initial guess for $X_1, \ldots, X_K$, `mat2dec` forms the corresponding vector of decision variables `xinit`.

An error occurs if the dimensions and structure of `X1,...,Xk` are inconsistent with the description of $X_1, \ldots, X_K$ in `lmisys`.

Example

Consider an LMI system with two matrix variables X and Y such that

- X is A symmetric block diagonal with one 2-by-2 full block and one 2-by-2 scalar block
- Y is a 2-by-3 rectangular matrix

Particular instances of X and Y are

$$X_0 = \begin{pmatrix} 1 & 3 & 0 & 0 \\ 3 & -1 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ 0 & 0 & 0 & 5 \end{pmatrix}, \qquad Y_0 = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

and the corresponding vector of decision variables is given by

```
decv = mat2dec(lmisys,X0,Y0)

decv'

ans =

       1    3    -1   5    1    2    3 4 5 6
```

Note that `decv` is of length 10 since Y has 6 free entries while X has 4 independent entries due to its structure. Use `decinfo` to obtain more information about the decision variable distribution in X and Y.

See Also

`dec2mat`, `decinfo`, `decnbr`

Purpose

Minimize a linear objective under LMI constraints

Syntax

```
[copt,xopt] = mincx(lmisys,c,options,xinit,target)
```

Description

The function `mincx` solves the convex program

$$\text{minimize } c^T x \text{ subject to } \mathcal{N}^T \mathcal{L}(x)\mathcal{N} \leq \mathcal{M}^T \mathcal{R}(x)\mathcal{M} \qquad \textbf{(9-18)}$$

where x denotes the vector of scalar decision variables.

The system of LMIs (9-18) is described by `lmisys`. The vector `c` must be of the same length as x. This length corresponds to the number of decision variables returned by the function `decnbr`. For linear objectives expressed in terms of the matrix variables, the adequate `c` vector is easily derived with `defcx`.

The function `mincx` returns the global minimum `copt` for the objective $c^T x$, as well as the minimizing value `xopt` of the vector of decision variables. The corresponding values of the matrix variables is derived from `xopt` with `dec2mat`.

The remaining arguments are optional. The vector `xinit` is an initial guess of the minimizer `xopt`. It is ignored when infeasible, but may speed up computations otherwise. Note that `xinit` should be of the same length as `c`. As for `target`, it sets some target for the objective value. The code terminates as soon as this target is achieved, that is, as soon as some feasible x such that $c^T x$ ð `target` is found. Set `options` to `[]` to use `xinit` and `target` with the default options.

Control Parameters

The optional argument `options` gives access to certain control parameters of the optimization code. In `mincx`, this is a five-entry vector organized as follows:

- `options(1)` sets the desired relative accuracy on the optimal value `lopt` (default = 10^{-2}).
- `options(2)` sets the maximum number of iterations allowed to be performed by the optimization procedure (100 by default).
- `options(3)` sets the feasibility radius. Its purpose and usage are as for `feasp`.

- `options(4)` helps speed up termination. If set to an integer value $J > 0$, the code terminates when the objective $c^T x$ has not decreased by more than the desired relative accuracy during the last J iterations.
- `options(5) = 1` turns off the trace of execution of the optimization procedure. Resetting `options(5)` to zero (default value) turns it back on.

Setting `option(i)` to zero is equivalent to setting the corresponding control parameter to its default value. See `feasp` for more detail.

Tip for Speed-Up

In LMI optimization, the computational overhead per iteration mostly comes from solving a least-squares problem of the form

$$\min_x \|Ax - b\|$$

where x is the vector of decision variables. Two methods are used to solve this problem: Cholesky factorization of $A^T A$ (default), and QR factorization of A when the normal equation becomes ill conditioned (when close to the solution typically). The message

```
* switching to QR
```

is displayed when the solver has to switch to the QR mode.

Since QR factorization is incrementally more expensive in most problems, it is sometimes desirable to prevent switching to QR. This is done by setting `options(4) = 1`. While not guaranteed to produce the optimal value, this generally achieves a good trade-off between speed and accuracy.

Memory Problems

QR-based linear algebra (see above) is not only expensiveLMI solvers,memory shortagememory shortage in terms of computational overhead, but also in terms of memory requirement. As a result, the amount of memory required by QR may exceed your swap space for large problems with numerous LMI constraints. In such case, MATLAB issues the error

```
??? Error using ==> pds
Out of memory. Type HELP MEMORY for your options.
```

You should then ask your system manager to increase your swap space or, if no additional swap space is available, set `options(4) = 1`. This will prevent

switching to QR and `mincx` will terminate when Cholesky fails due to numerical instabilities.

Example

See "Example 8.2" on page 8-23.

Reference

The solver `mincx` implements Nesterov and Nemirovski's Projective Method as described in

Nesterov, Yu, and A. Nemirovski, *Interior Point Polynomial Methods in Convex Programming: Theory and Applications*, SIAM, Philadelphia, 1994.

Nemirovski, A., and P. Gahinet, "The Projective Method for Solving Linear Matrix Inequalities," *Proc. Amer. Contr. Conf.*, 1994, Baltimore, Maryland, pp. 840–844.

The optimization is performed by the C-MEX file `pds.mex`.

See Also

`defcx`, `dec2mat`, `decnbr`, `feasp`, `gevp`

msfsyn

Purpose

Multi-model/multi-objective state-feedback synthesis

Syntax

```
[gopt,h2opt,K,Pcl,X] = msfsyn(P,r,obj,region,tol)
```

Description

Given an LTI plant `P` with state-space equations

$$\begin{cases} \dot{x} &= Ax + B_1 w + B_2 u \\ z_\infty &= C_1 x + D_{11} w + D_{12} u \\ z_2 &= C_2 x + D_{22} u \end{cases}$$

`msfsyn` computes a state-feedback control $u = Kx$ that

- Maintains the RMS gain (H_∞ norm) of the closed-loop transfer function T_∞ from w to z_∞ below some prescribed value $\gamma_0 > 0$
- Maintains the H_2 norm of the closed-loop transfer function T_2 from w to z_2 below some prescribed value $\upsilon_0 > 0$
- Minimizes an H_2/H_∞ trade-off criterion of the form

  $$\alpha \|T_\infty\|_\infty^2 + \beta \|T_2\|_2^2$$

- Places the closed-loop poles inside the LMI region specified by `region` (see `lmireg` for the specification of such regions). The default is the open left-half plane.

Set `r = size(d22)` and `obj` $= [\gamma_0, \nu_0, \alpha, \beta]$ to specify the problem dimensions and the design parameters γ_0, ν_0, α, and β. You can perform pure pole placement by setting `obj = [0 0 0 0]`. Note also that z_∞ or z_2 can be empty.

On output, `gopt` and `h2opt` are the guaranteed H_∞ and H_2 performances, `K` is the optimal state-feedback gain, `Pcl` the closed-loop transfer function from w to $\begin{pmatrix} z_\infty \\ z_2 \end{pmatrix}$, and `X` the corresponding Lyapunov matrix.

The function `msfsyn` is also applicable to multi-model problems where `P` is a polytopic model of the plant:

$$\begin{cases} \dot{x} &= A(t)x + B_1(t)w + B_2(t)u \\ z_\infty &= C_1(t)x + D_{11}(t)w + D_{12}(t)u \\ z_2 &= C_2(t)x + D_{22}(t)u \end{cases}$$

with time-varying state-space matrices ranging in the polytope

$$\begin{pmatrix} A(t) & B_1(t) & B_2(t) \\ C_1(t) & D_{11}(t) & D_{12}(t) \\ C_2(t) & 0 & D_{22}(t) \end{pmatrix} \in \mathrm{Co}\left\{ \begin{pmatrix} A_k & B_k & B_{2k} \\ C_{1k} & D_{11k} & D_{12k} \\ C_{2k} & 0 & D_{22k} \end{pmatrix} : k = 1, \ldots, K \right\}$$

In this context, `msfsyn` seeks a state-feedback gain that robustly enforces the specifications over the entire polytope of plants. Note that polytopic plants should be defined with `psys` and that the closed-loop system `Pcl` is itself polytopic in such problems. Affine parameter-dependent plants are also accepted and automatically converted to polytopic models.

Example

See “Design Example” on page 4-13.

See Also

`lmireg`, `psys`, `hinfmix`

mubnd

Purpose

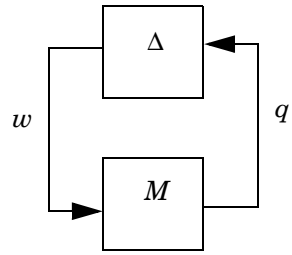
Compute the upper bound for the μ structured singular value

Syntax

```
[mu,D,G] = mubnd(M,delta,target)
```

Description

`mubnd` computes the mixed-μ upper bound `mu` = $\nu_\Delta(M)$ for the matrix $M \in C^{m \times n}$ and the norm-bounded structured perturbation $\Delta = \mathrm{diag}(\Delta_1, \ldots, \Delta n)$ (see "Structured Singular Value" on page 3-17). The reciprocal of `mu` is a guaranteed well-posedness margin for the algebraic loop



Specifically, this loop remains well-posed as long as

`mu` x $\sigma_{\max}(\Delta_i) < \beta_i$

where β_i is the specified bound on Δ_i.

The uncertainty Δ is described by `delta` (see `ublock` and `udiag`). The function `mubnd` also returns the optimal scaling matrices `D` and `G` arising in the computation of `mu`.

To test whether `mu` $\leq \tau$ where $\tau > 0$ is a given scalar, set the optional argument `target` to τ .

See Also

`mustab`, `muperf`

Purpose

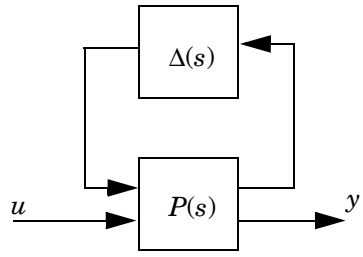
Estimate the robust H_∞ performance of uncertain dynamical systems

Syntax

```
[grob,peakf] = muperf(P,delta,0,freqs)
[margin,peakf] = muperf(P,delta,g,freqs)
```

Description

`muperf` estimates the robust H_∞ performance (worst-case RMS gain) of the uncertain LTI system



where $\Delta(s) = \mathrm{diag}(\Delta_1(s), \ldots, \Delta_n(s))$

is a *linear time-invariant* uncertainty quantified by norm bounds $\sigma_{\max}(\Delta_i(j\omega)) < \beta_i(\omega)$ or sector bounds $\Delta_i \in \{a, b\}$. The robust performance γ_{rob} is the smallest $\gamma > 0$ such that

$$\|y\|_{L_2} < \gamma \|u\|_{L_2}$$

for all bounded input $u(t)$ and instance $\Delta(s)$ of the uncertainty. It is finite if and only if the interconnection is robustly stable.

`muperf` converts the robust performance estimation into a robust stability problem for some augmented Δ structure (see "Robust Performance" on page 3-21). The input arguments `P` and `delta` contain the `SYSTEM` matrix of $P(s)$ and the description of the LTI uncertainty $\Delta(s)$ (see `ublock` and `udiag`). To compute the robust H_∞ performance `grob` for the prescribed uncertainty `delta`, call `muperf` with the syntax

```
[grob,peakf] = muperf(P,delta)
```

`muperf` returns `grob = Inf` when the interconnection is not robustly stable.

Alternatively,

```
[margin,peakf] = muperf(P,delta,g)
```

estimates the robustness of a given H_∞ performance `g`, that is, how much uncertainty $\Delta(s)$ can be tolerated before the RMS gain from u to y exceeds `g`. Provided that

$$\sigma_{max}(\Delta_i(j\omega)) < \texttt{margin} \times \beta_i(\omega)$$

in the norm-bounded case and

$$\Delta_i \in \left\{ \frac{a+b}{2} - \frac{b-a}{2} \times \texttt{margin},\ \frac{a+b}{2} + \frac{b-a}{2} \times \texttt{margin} \right\}$$

in the sector-bounded case, the RMS gain is guaranteed to remain below `g`.

With both syntaxes, `peakf` is the frequency where the worst RMS performance or robustness margin is achieved. An optional vector of frequencies freqs to be used for the μ upper bound evaluation can be specified as fourth input argument.

See Also

`mustab`, `norminf`, `quadperf`

Purpose

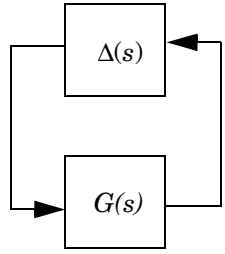
Estimate the robust stability margin of uncertain LTI systems

Syntax

```
[margin,peak,fs,ds,gs] = mustab(sys,delta,freqs)
```

Description

`mustab` computes an estimate `margin` of the robust stability margin for the interconnection



where $G(s)$ is a given LTI system and $\Delta = \mathrm{diag}(\Delta_1, \ldots, \Delta_n)$ is a linear time-invariant uncertainty quantified by norm bounds $\sigma_{\max}(\Delta i(j\omega)) < \beta_i(\omega)$ or sector bounds $\Delta_i \in \{a, b\}$. In the norm-bounded case, the stability of this interconnection is guaranteed for all $\Delta(s)$ satisfying

$$\sigma_{\max}(\Delta_i(j\omega)) < \texttt{margin} \times \beta_i(\omega)$$

See "Robust Stability Analysis" on page 3-19 for the interpretation of `margin` in the sector-bounded case.

The value `margin` is obtained by computing the mixed-μ upper bound $\nu_\Delta(G(j\omega_i))$ over a grid of frequencies ω_i. The command

```
[margin,peak] = mustab(sys,delta)
```

returns the guaranteed stability margin `margin` and the frequency `peakf` where the margin is weakest. The uncertainty $\Delta(s)$ is specified by `delta` (see `ublock` and `udiag`). The optional third argument freqs is a user-supplied vector of frequencies where to evaluate $\nu_\Delta(G(j\omega))$.

To obtain the *D, G* scaling matrices at all tested frequencies, use the syntax

```
[margin,peak,fs,ds,gs] = mustab(sys,delta)
```

Here `fs` is the vector of frequencies used to evaluate ν_Δ and the *D, G* scalings at the frequency `fs(i)` are retrieved by

```
D = getdg(ds,i)
G = getdg(gs,i)
```

Caution

`margin` may overestimate the robust stability margin when the frequency grid `freqs` is too coarse or when the μ upper bound is discontinuous due to real parameter uncertainty. See "Caution" on page 3-21 for more details.

Example

See "Example" on page 3-28.

See Also

`mubnd`, `muperf`, `ublock`, `udiag`

Purpose

Attach an identifying tag to LMIs

Syntax

```
tag = newlmi
```

Description

`newlmi` adds a new LMI to the LMI system currently described and returns an identifier `tag` for this LMI. This identifier can be used in `lmiterm`, `showlmi`, or `dellmi` commands to refer to the newly declared LMI. Tagging LMIs is *optional* and only meant to facilitate code development and readability.

Identifiers can be given mnemonic names to help keep track of the various LMIs. Their value is simply the ranking of each LMI in the system (in the order of declaration). They prove useful when some LMIs are deleted from the LMI system. In such cases, the identifiers are the safest means of referring to the remaining LMIs.

See Also

`setlmis`, `lmivar`, `lmiterm`, `getlmis`, `lmiedit`, `dellmi`

norminf

Purpose

Compute the random mean-squares (RMS) gain of continuous-time systems

Syntax

```
[gain,peakf] = norminf(g,tol)
```

Description

Given the SYSTEM matrix g of an LTI system with state-space equations

$$E\frac{dx}{dt} = Ax + Bu$$
$$y = Cx + Du$$

and transfer function $G(s) = D + C(sE - A)^{-1}B$, norminf computes the peak gain

$$\|G\|_\infty = \sup_\omega \sigma_{max}(G(j\omega))$$

of the frequency response $G(j\omega)$. When $G(s)$ is stable, this peak gain coincides with the RMS gain or H_∞ norm of G, that is,

$$\|G\|_\infty = \sup_{\substack{u \in L_2 \\ u \neq 0}} \frac{\|y\|_{L_2}}{\|u\|_{L_2}}$$

The function norminf returns the peak gain and the frequency at which it is attained in gain and peakf, respectively. The optional input tol specifies the relative accuracy required on the computed RMS gain (the default value is 10^{-2}).

Example

The peak gain of the transfer function

$$G(s) = \frac{100}{s^2 + 0.01s + 100}$$

is given by

```
norminf(ltisys('tf',100,[1 0.01 100]))

ans =

   1.0000e+03
```

Reference

Boyd, S., V. Balakrishnan, and P. Kabamba, "A Bisection Method for Computing the H_∞ Norm of a Transfer Matrix and Related Problems," *Math. Contr. Sign. Syst.*, 2 (1989), pp. 207–219.

Bruisma, N.A., and M. Steinbuch, "A Fast Algorithm to Compute the H_∞-Norm of a Transfer Function Matrix," *Syst. Contr. Letters*, 14 (1990), pp. 287–293.

Robel, G., "On Computing the Infinity Norm," *IEEE Trans. Aut. Contr.*, AC–34 (1989), pp. 882–884.

See Also

`dnorminf`, `muperf`, `quadperf`

norm2

Purpose

Compute the H_2 norm of a continuous-time LTI system

Syntax

```
h2norm = norm2(g)
```

Description

Given a stable LTI system

$$G:\begin{cases}\dot{x} = Ax + Bw \\ y = Cx\end{cases}$$

driven by a white noise w with unit covariance, `norm2` computes the H_2 norm (LQG performance) of G defined by

$$\|G\|_2^2 = \lim_{T\to\infty} E\left\{\frac{1}{T}\int_0^T y^T(t)y(t)dt\right\}$$

$$= \frac{1}{2\pi}\int_{-\infty}^{\infty} G^H(j\omega)G(j\omega)d\omega$$

where $G(s) = C(sI - A)^{-1}B$.

This norm is computed as

$$\|G\|_2 = \sqrt{\mathrm{Trace}(CPC^T)}$$

where P is the solution of the Lyapunov equation

$$AP + PA^T + BB^T = 0$$

See Also

`norminf`

Purpose

Assess the robust stability of a polytopic or parameter-dependent system

Syntax

```
[tau,Q0,Q1,...] = pdlstab(pds,options)
```

Description

`pdlstab` uses parameter-dependent Lyapunov functions to establish the stability of uncertain state-space models over some parameter range or polytope of systems. Only sufficient conditions for the existence of such Lyapunov functions are available in general. Nevertheless, the resulting robust stability tests are always less conservative than quadratic stability tests when the parameters are either time-invariant or slowly varying.

For an affine parameter-dependent system

$$E(p)\dot{x} = A(p)x + B(p)u$$

$$y = C(p)x + D(p)u$$

with $p = (p_1, \ldots, p_n) \in \mathbf{R}^n$, `pdlstab` seeks a Lyapunov function of the form

$$V(x, p) = x^T Q(p)^{-1} x, \qquad Q(p) = Q_0 + p_1 Q_1 + \ldots p_n Q_n$$

such that $dV(x, p)/dt < 0$ along all admissible parameter trajectories. The system description `pds` is specified with `psys` and contains information about the range of values and rate of variation of each parameter p_i.

For a *time-invariant* polytopic system

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

$$y = Cx + Du$$

with

$$\begin{pmatrix} A + jE & B \\ C & D \end{pmatrix} = \sum_{i=1}^{n} \alpha_i \begin{pmatrix} A + jE_i & B_i \\ C_i & D_i \end{pmatrix}, \qquad \alpha_i \geq 0, \qquad \sum_{i=1}^{n} \alpha_i = 1, \tag{9-19}$$

`pdlstab` seeks a Lyapunov function of the form

$$V(x, \alpha) = x^T Q(\alpha)^{-1} x, \qquad Q(\alpha) = \alpha_1 Q_1 + \ldots + \alpha_n Q_n$$

such that $dV(x, \alpha)/dt < 0$ for all polytopic decompositions (9-19).

Several options and control parameters are accessible through the optional argument `options`:

- Setting `options(1)=0` tests robust stability (default)
- When `options(2)=0`, `pdlstab` uses simplified sufficient conditions for faster running times. Set `options(2)=1` to use the least conservative conditions

Remark

For affine parameter-dependent systems with *time-invarian*t parameters, there is equivalence between the robust stability of

$$E(p)\dot{x} = A(p)x \qquad \textbf{(9-20)}$$

and that of the dual system

$$E(p)^T\dot{z} = A(p)^T z \qquad \textbf{(9-21)}$$

However, the second system may admit an affine parameter-dependent Lyapunov function while the first does not.

In such case, `pdlstab` automatically restarts and tests stability on the dual system (9-21) when it fails on (9-20).

See Also

`quadstab`, `mustab`

Purpose

Time response of a parameter-dependent system along a given parameter trajectory

Syntax

```
pdsimul(pds,'traj',tf,'ut',xi,options)
[t,x,y] = pdsimul(pds,pv,'traj',tf,'ut',xi,options)
```

Description

`pdsimul` simulates the time response of an affine parameter-dependent system

$$E(p)\dot{x} = A(p)x + B(p)u$$

$$y = C(p)x + D(p)u$$

along a parameter trajectory $p(t)$ and for an input signal $u(t)$. The parameter trajectory and input signals are specified by two time functions `p=traj(t)` and `u=ut(t)`. If `'ut'` is omitted, the response to a step input is computed by default.

The affine system `pds` is specified with `psys`. The function `pdsimul` also accepts the polytopic representation of such systems as returned by `aff2pol(pds)` or `hinfgs`. The final time and initial state vector can be reset through `tf` and `xi` (their respective default values are 5 seconds and 0). Finally, `options` gives access to the parameters controlling the ODE integration (type `help gear` for details).

When invoked without output arguments, `pdsimul` plots the output trajectories $y(t)$. Otherwise, it returns the vector of integration time points `t` as well as the state and output trajectories `x,y`.

Example

See “Design Example” on page 7-10.

See Also

`psys`, `pvec`, `gear`

popov

Purpose

Perform the Popov robust stability test

Syntax

```
[t,P,S,N] = popov(sys,delta,flag)
```

Description

`popov` uses the Popov criterion to test the robust stability of dynamical systems with possibly nonlinear and/or time-varying uncertainty (see "The Popov Criterion" on page 3-24 for details). The uncertain system must be described as the interconnection of a nominal LTI system `sys` and some uncertainty `delta` (see "Norm-Bounded Uncertainty" on page 2-27). Use `ublock` and `udiag` to specify `delta`.

The command

```
[t,P,S,N] = popov(sys,delta)
```

tests the robust stability of this interconnection. Robust stability is guaranteed if `t` < 0. Then `P` determines the quadratic part x^TPx of the Lyapunov function and `D` and `S` are the Popov multipliers (see p. ?? for details).

If the uncertainty `delta` contains real parameter blocks, the conservatism of the Popov criterion can be reduced by first performing a simple loop transformation (see "Real Parameter Uncertainty" on page 3-25). To use this refined test, call `popov` with the syntax

```
[t,P,S,N] = popov(sys,delta,1)
```

Example

See "Design Example" on page 7-10.

See Also

`mustab`, `quadstab`, `pdlstab`, `ublock`, `aff2lft`

Purpose

Inquire about polytopic or parameter-dependent systems created with psys

Syntax

```
psinfo(ps)
[type,k,ns,ni,no] = psinfo(ps)
pv = psinfo(ps,'par')
sk = psinfo(ps,'sys',k)
sys = psinfo(ps,'eval',p)
```

Description

psinfo is a multi-usage function for queries about a polytopic or parameter-dependent system ps created with psys. It performs the following operations depending on the calling sequence:

- psinfo(ps) displays the type of system (affine or polytopic); the number k of SYSTEM matrices involved in its definition; and the numbers of ns, ni, no of states, inputs, and outputs of the system. This information can be optionally stored in MATLAB variables by providing output arguments.
- pv = psinfo(ps,'par') returns the parameter vector description (for parameter-dependent systems only).
- sk = psinfo(ps,'sys',k) returns the k-th SYSTEM matrix involved in the definition of ps. The ranking k is relative to the list of systems syslist used in psys.
- sys = psinfo(ps,'eval',p) instantiates the system for a given vector p of parameter values or polytopic coordinates.

  For *affine parameter-dependent* systems defined by the SYSTEM matrices S_0, $S_1, \ldots, S_n$, the entries of p should be real parameter values $p_1, \ldots, p_n$ and the result is the LTI system of SYSTEM matrix

  $$S(p) = S_0 + p_1 S_1 + \ldots + p_n S_n$$

  For *polytopic* systems with SYSTEM matrix ranging in

  $$\mathrm{Co}\{S_1, \ldots, S_n\},$$

  the entries of p should be polytopic coordinates $p_1, \ldots, p_n$ satisfying $p_j \geq 0$ and the result is the interpolated LTI system of SYSTEM matrix

  $$S = \frac{p_1 S_1 + \ldots + p_n S_n}{p_1 + \ldots + p_n}$$

See Also

psys, ltisys

Purpose

Specify polytopic or parameter-dependent linear systems

Syntax

```
pols = psys(syslist)
affs = psys(pv,syslist)
```

Description

`psys` specifies state-space models where the state-space matrices can be uncertain, time-varying, or parameter-dependent.

Two types of uncertain state-space models can be manipulated in the LMI Control Toolbox:

- *Polytopic* systems

  $$E(t)\dot{x} = A(t)x + B(t)u$$

  $$y = C(t)x + D(t)u$$

  whose `SYSTEM` matrix takes values in a fixed polytope:

$$\underbrace{\begin{bmatrix} A(t)+jE(t) & B(t) \\ C(t) & D(t) \end{bmatrix}}_{S(t)} \in \mathrm{Co}\left\{ \underbrace{\begin{bmatrix} A_1+jE_1 & B_1 \\ C_1 & D_1 \end{bmatrix}}_{S_1}, \ldots, \underbrace{\begin{bmatrix} A_k+jE_k & B_k \\ C_k & D_k \end{bmatrix}}_{S_k} \right\}$$

  where $S_1, \ldots, S_k$ are given "vertex" systems and

$$\mathrm{Co}\{S_1, \ldots, S_k\} = \left\{ \sum_{i=1}^{k} \alpha_i S_i \ : \ \alpha_i \geq 0, \ \sum_{i=1}^{k} \alpha_i = 1 \right\}$$

  denotes the convex hull of $S_1, \ldots, S_k$ (polytope of matrices with vertices $S_1, \ldots, S_k$)

- *Affine parameter-dependent* systems

  $$E(p)\dot{x} = A(p)x + B(p)u$$

$$y = C(p)x + D(p)u$$

where $A(\cdot)$; $B(\cdot)$, . . ., $E(\cdot)$ are fixed affine functions of some vector $p = (p_1, \ldots, p_n)$ of real parameters, i.e.,

$$\underbrace{\begin{bmatrix} A(p) + jE(p) & B(p) \\ C(p) & D(p) \end{bmatrix}}_{S(p)} =$$

$$\underbrace{\begin{bmatrix} A_0 + jE_0 & B_0 \\ C_0 & D_0 \end{bmatrix}}_{S_0} + p_1 \underbrace{\begin{bmatrix} A_1 + jE_1 & B_1 \\ C_1 & D_1 \end{bmatrix}}_{S_1} + \ldots + p_n \underbrace{\begin{bmatrix} A_n + jE_n & B_n \\ C_n & D_n \end{bmatrix}}_{S_n}$$

where S_0, S_1, . . ., S_n are given `SYSTEM` matrices. The parameters p_i can be time-varying or constant but uncertain.

Both types of models are specified with the function `psys`. The argument `syslist` lists the `SYSTEM` matrices S_i characterizing the polytopic value set or parameter dependence. In addition, the description `pv` of the parameter vector (range of values and rate of variation) is required for affine parameter-dependent models (see `pvec` for details). Thus, a polytopic model with vertex systems S_1, . . ., S_4 is created by

```
pols = psys([s1,s2,s3,s4])
```

while an affine parameter-dependent model with 4 real parameters is defined by

```
affs = psys(pv,[s0,s1,s2,s3,s4])
```

The output is a structured matrix storing all the relevant information.

To specify the autonomous parameter-dependent system

$$\dot{x} = A(p)x, \qquad A(p) = A_0 + p_1A_1 + p_2A_2,$$

type

```
s0 = ltisys(a0)
s1 = ltisys(a1,0)
s2 = ltisys(a2,0)
ps = psys(pv,[s0 s1 s2])
```

Do not forget the `0` in the second and third commands. This `0` marks the independence of the E matrix on p_1, p_2 . Typing

```
s0 = ltisys(a0)
s1 = ltisys(a1)
s2 = ltisys(a2)
ps = psys(pv,[s0 s1 s2])
```

instead would specify the system

$$E(p)\dot{x} = A(p)x, \qquad E(p) = (1+p_1 + p_2)I, \qquad A(p) = A_0 + p_1A_1 + p_2A_2$$

Example

See “Example 2.1” on page 2-21.

See Also

`psinfo`, `pvec`, `aff2pol`, `ltisys`

Purpose

Specify the range and rate of variation of uncertain or time-varying parameters

Syntax

```
pv = pvec('box',range,rates)
pv = pvec('pol',vertices)
```

Description

`pvec` is used in conjunction with `psys` to specify parameter-dependent systems. Such systems are parametrized by a vector $p = (p_1, \ldots, p_n)$ of uncertain or time-varying real parameters p_i. The function `pvec` defines the range of values and the rates of variation of these parameters.

The type `'box'` corresponds to independent parameters ranging in intervals

$$\underline{p}_j \le p_j \le \bar{p}_j$$

The parameter vector p then takes values in a hyperrectangle of $\mathbf{R}^n$ called the parameter box. The second argument `range` is an n-by-2 matrix that stacks up the extremal values $\underline{p}_j$ and $\bar{p}_j$ of each p_j. If the third argument `rates` is omitted, all parameters are assumed time-invariant. Otherwise, `rates` is also an n-by-2 matrix and its j-th row specifies lower and upper bounds $\underline{v}_j$ and $\bar{v}_j$ on $\frac{dp_j}{dt}$:

$$\underline{v}_j \le \frac{dp_j}{dt} \le \bar{v}_j$$

Set $\underline{v}_j$ = `Inf` and $\bar{v}_j$ = `Inf` if $p_j(t)$ can vary arbitrarily fast or discontinuously.

The type `'pol'` corresponds to parameter vectors p ranging in a polytope of the parameter space $\mathbf{R}^n$. This polytope is defined by a set of vertices $V_1, \ldots, V_n$ corresponding to "extremal" values of the vector p. Such parameter vectors are declared by the command

```
pv = pvec('pol',[v1,v2, . . ., vn])
```

where the second argument is the concatenation of the vectors `v1,...,vn`.

The output argument `pv` is a structured matrix storing the parameter vector description. Use `pvinfo` to read the contents of `pv`.

Example

Consider a problem with two time-invariant parameters

$$p_1 \in [-1, 2], \qquad p_2 \in [20, 50]$$

The corresponding parameter vector $p = (p_1, p_2)$ is specified by

```
pv = pvec('box',[-1 2;20 50])
```

Alternatively, this vector can be regarded as taking values in the rectangle drawn in Figure 9.2. The four corners of this rectangle are the four vectors

$$v_1 = \begin{pmatrix} -1 \\ 20 \end{pmatrix}, \quad v_2 = \begin{pmatrix} -1 \\ 50 \end{pmatrix}, \quad v_3 = \begin{pmatrix} 2 \\ 20 \end{pmatrix}, \quad v_4 = \begin{pmatrix} 2 \\ 50 \end{pmatrix}$$

Hence, you could also specify p by

```
pv = pvec('pol',[v1,v2,v3,v4])
```

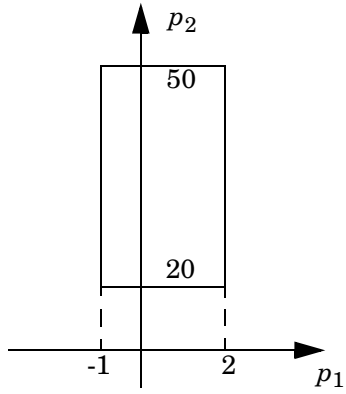


Figure 9-2: Parameter box

See Also

`pvinfo`, `psys`

Purpose

Describe a parameter vector specified with pvec

Syntax

```
[typ,k,nv] = pvinfo(pv)
[pmin,pmax,dpmin,dpmax] = pvinfo(pv,'par',j)
vj = pvinfo(pv,'par',j)
p = pvinfo(pv,'eval',c)
```

Description

pvec retrieves information about a vector $p = (p_1, \ldots, p_n)$ of real parameters declared with pvec and stored in pv. The command pvinfo(pv) displays the type of parameter vector ('box' or 'pol'), the number n of scalar parameters, and for the type 'pol', the number of vertices used to specify the parameter range.

For the type 'box':

```
[pmin,pmax,dpmin,dpmax] = pvinfo(pv,'par',j)
```

returns the bounds on the value and rate of variations of the j-th real parameter p_j. Specifically,

$$\texttt{pmin} \le p_j(t) \le \texttt{pmax}\ ,\ \texttt{dpmin} \le \frac{dp_j}{dt} \le \texttt{dpmax}$$

For the type 'pol':

```
pvinfo(pv,'par',j)
```

returns the j-th vertex of the polytope of $\mathbf{R}^n$ in which p ranges, while

```
pvinfo(pv,'eval',c)
```

returns the value of the parameter vector p given its barycentric coordinates c with respect to the polytope vertices $(V_1, \ldots, V_k)$. The vector c must be of length k and have nonnegative entries. The corresponding value of p is then given by

$$p = \frac{\sum_{i=1}^{k} c_i V_i}{\sum_{i=1}^{k} c_i}$$

See Also

pvec, psys

Purpose

Compute the quadratic H_∞ performance of a polytopic or parameter-dependent system

Syntax

```
[perf,P] = quadperf(ps,g,options)
```

Description

The RMS gain of the time-varying system

$$E(t)\dot{x} = A(t)x + B(t)u, \qquad y = C(t)x + D(t)u \tag{9-22}$$

is the smallest $\gamma > 0$ such that

$$\|y\|_{L_2} \le \gamma \|u\|_{L_2} \tag{9-23}$$

for all input $u(t)$ with bounded energy. A sufficient condition for (9-23) is the existence of a quadratic Lyapunov function

$$V(x) = x^T P x, \qquad P > 0$$

such that

$$\forall u \in L_2, \quad \frac{dV}{dt} + y^T y - \gamma^2 u^T u < 0$$

Minimizing γ over such quadratic Lyapunov functions yields the quadratic H_∞ performance, an upper bound on the true RMS gain.

The command

```
[perf,P] = quadperf(ps)
```

computes the quadratic H_∞ performance `perf` when (9-22) is a polytopic or affine parameter-dependent system `ps` (see `psys`). The Lyapunov matrix P yielding the performance `perf` is returned in `P`.

The optional input `options` gives access to the following task and control parameters:

- If `options(1)=1`, `perf` is the largest portion of the parameter box where the quadratic RMS gain remains smaller than the positive value `g` (for affine parameter-dependent systems only). The default value is 0

- If `options(2)=1`, `quadperf` uses the least conservative quadratic performance test. The default is `options(2)=0` (fast mode)
- `options(3)` is a user-specified upper bound on the condition number of P (the default is 10^9).

Example

See "Example 3.4" on page 3-10.

See Also

`muperf`, `quadstab`, `psys`

quadstab

Purpose

Quadratic stability of polytopic or affine parameter-dependent systems

Syntax

```
[tau,P] = quadstab(ps,options)
```

Description

For affine parameter-dependent systems

$$E(p)\dot{x} = A(p)x, \qquad p(t) = (p_1(t), \ldots, p_n(t))$$

or polytopic systems

$$E(t)\dot{x} = A(t)x, \qquad (A, E) \in \mathrm{Co}\{(A_1, E_1), \ldots, (A_n, E_n)\},$$

`quadstab` seeks a fixed Lyapunov function $V(x) = x^T P x$ with $P > 0$ that establishes quadratic stability (see "Quadratic Lyapunov Functions" on page 3-3 for details). The affine or polytopic model is described by `ps` (see `psys`).

The task performed by `quadstab` is selected by `options(1)`:

- if `options(1)=0` (default), `quadstab` assesses quadratic stability by solving the LMI problem

  Minimize τ over $Q = Q^T$ such that

  $A^T Q E + E Q A^T < \tau I$ for all admissible values of (A, E)

  $$Q > I$$

  The global minimum of this problem is returned in `tau` and the system is quadratically stable if `tau` < 0
- if `options(1)=1`, `quadstab` computes the largest portion of the specified parameter range where quadratic stability holds (only available for affine models). Specifically, if each parameter p_i varies in the interval

  $p_i \in [p_{i0} - \delta_i, p_{i0} + \delta_i]$,

  `quadstab` computes the largest $\theta > 0$ such that quadratic stability holds over the parameter box

  $p_i \in [p_{i0} - \theta\delta_i, p_{i0} + \theta\delta_i]$

  This "quadratic stability margin" is returned in `tau` and `ps` is quadratically stable if `tau` Š 1.

Given the solution Q_{opt} of the LMI optimization, the Lyapunov matrix P is given by $P = Q_{\mathrm{opt}}^{-1}$. This matrix is returned in `P`.

Other control parameters can be accessed through `options(2)` and `options(3)`:

- if `options(2)=0` (default), `quadstab` runs in fast mode, using the least expensive sufficient conditions. Set `options(2)=1` to use the least conservative conditions
- `options(3)` is a bound on the condition number of the Lyapunov matrix P. The default is 10^9.

Example

See "Example 3.1" on page 3-7 and "Example 3.2" on page 3-8.

See Also

`pdlstab`, `mustab`, `decay`, `quadperf`, `psys`

Purpose

Solve continuous-time Riccati equations (CARE)

Syntax

```
X = ricpen(H,L)
[X1,X2] = ricpen(H,L)
```

Description

This function computes the stabilizing solution X of the Riccati equation

$$A^TXE + E^TXA + E^TXFXE - (E^TXB + S)R^{-1}(B^TXE + S^T) + Q = 0 \quad \textbf{(9-24)}$$

where E is an invertible matrix. The solution is computed by unitary deflation of the associated Hamiltonian pencil

$$H - \lambda L = \begin{bmatrix} A & F & B \\ -Q & -A^T & -S \\ S^T & B^T & R \end{bmatrix} - \lambda \begin{bmatrix} E & 0 & 0 \\ 0 & E^T & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

The matrices H and L are specified as input arguments. For solvability, the pencil $H - \lambda L$ must have no finite generalized eigenvalue on the imaginary axis.

When called with two output arguments, `ricpen` returns two matrices X_1, X_2 such that $\begin{bmatrix} X_1 \\ X_2 \end{bmatrix}$ has or thonormal columns, and $X = X_2X_1^{-1}$ is the stabilizing solution of (9-24).

If X_1 is invertible, the solution X is returned directly by the syntax `X = ricpen(H,L)`.

`ricpen` is adapted from `care` and implements the generalized Schur method for solving CAREs.

Reference

Laub, A. J., "A Schur Method for Solving Algebraic Riccati Equations," *IEEE Trans. Aut. Contr.*, AC–24 (1979), pp. 913–921.

Arnold, W.F., and A.J. Laub, "Generalized Eigenproblem Algorithms and Software for Algebraic Riccati Equations," *Proc. IEEE*, 72 (1984), pp. 1746–1754.

See Also care, dricpen

sadd

Purpose

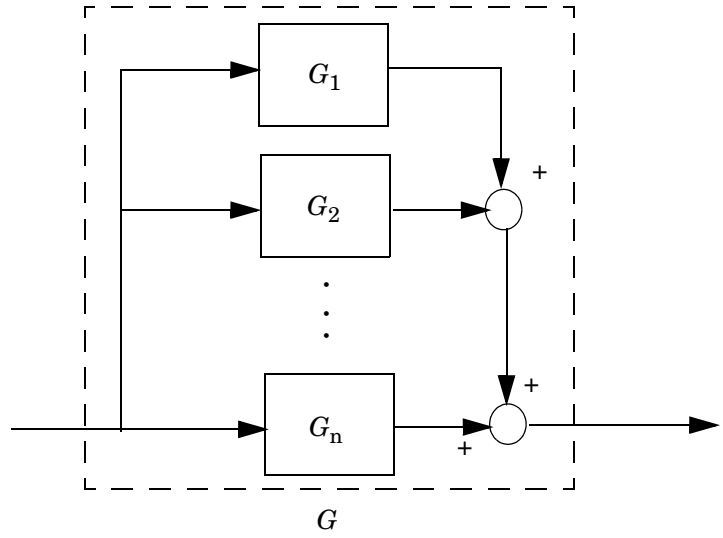
Parallel interconnection of linear systems

Syntax

```
sys = sadd(g1,g2,...)
```

Description

`sadd` forms the parallel interconnection



of the linear systems $G_1, \ldots, G_n$. The arguments `g1, g2,...` are either `SYSTEM` matrices (dynamical systems) or constant matrices (static gains). In addition, one (and at most one) of these systems can be polytopic or parameter-dependent, in which case the resulting system `sys` is of the same nature.

In terms of transfer functions, `sadd` performs the sum

$$G(s) = G_1(s) + G_2(s) + \ldots + G_n(s)$$

See Also

`smult`, `sdiag`, `sloop`, `ltisys`

Purpose

Numerically balance the state-space realization of a linear system

Syntax

```
[a,b,c] = sbalanc(a,b,c,condnbr)
sys = sbalanc(sys,condnbr)
```

Description

`sbalanc` computes a diagonal invertible matrix T that balances the state-space realization (A, B, C) by reducing the magnitude of the off-diagonal entries of A and by scaling the norms of B and C. The resulting realization is of the form

$$(T^{-1}A^T, T^{-1}B, CT)$$

The optional argument `condnbr` specifies an upper bound on the condition number of T. The default value is 10^8.

Example

Computing a realization of

$$W(s) = \frac{s^2 + s + 1}{(s + 500)^2}$$

with the function `tf2ss` yields

```
[a,b,c,d]=tf2ss([1 1 1],[1 1000 500^2])

a =
   -1000 -250000
   1    0
b =
   1
   0
c =
   -999 -249999
d =
   1
```

To reduce the numerical range of this data and improve numerical stability of subsequent computations, you can balance this realization by

```
[a,b,c]=sbalanc(a,b,c)

a =
   -1000.00    -3906.25
       64.00           0
```

```
b =
       64.00
       0

c =
       -15.61      -61.03
```

See Also

ltisys

Purpose

Specify general control structures or system interconnections

Syntax

```
[P,r] = sconnect(inputs,outputs,Kin,G1in,g1,G2in,g2,...)
```

Description

`sconnect` is useful in loop-shaping problems to turn general control structures into the standard linear-fractional interconnection used in H_∞ synthesis. In this context, `sconnect` returns:

- The `SYSTEM` matrix `P` of the standard H_∞ plant $P(s)$
- The vector `r = [p2 m2]` where `p2` and `m2` are the number of input and outputs of the controller, respectively.

More generally, `sconnect` is useful to specify complex interconnections of LTI systems (see Example 1 below).

General control loops or system interconnections are described in terms of signal flows. Specifically, you must specify

- The exogenous inputs, i.e., what signals enter the loop or interconnection
- The ouput signals, i.e., the signals generated by the loop or interconnection
- How the inputs of each dynamical system relate to the exogenous inputs and to the outputs of other systems.

The outputs of a system of name `G` are collectively denoted by `G`. To refer to particular outputs, e.g., the second and third, use the syntax `G(2:3)`.

The arguments of `sconnect` are as follows:

- The first argument `inputs` is a string listing the exogenous inputs. For instance, `inputs='r(2), d'` specifies two inputs, a vector r of size 2 and a scalar input d
- The second argument `outputs` is a string listing the outputs generated by the control loop. Outputs are defined as combinations of the exogenous inputs and the outputs of the dynamical systems. For instance, `outputs='e=r-S; S+d'` specifies two outputs $e = r - y$ and $y + d$ where y is the output of the system of name `S`.
- The third argument `Kin` names the controller and specifies its inputs. For instance, `Kin='K:e'` inserts a controller of name `K` and input e. If no name is specified as in `Kin='e'`, the default name `K` is given to the controller.

- The remaining arguments come in pairs and specify, for each known LTI system in the loop, its input list `Gkin` and its `SYSTEM` matrix `gk`. The input list is a string of the form

  ```
  system name : input1 ; input2 ; ... ; input n
  ```

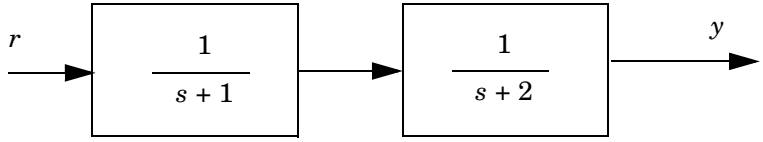
  For instance, `G1in='S: K(2);d'` inserts a system called `S` whose inputs consist of the second output of the controller `K` and of the input signal d.

Note that the names given to the various systems are immaterial provided that they are used consistently throughout the definitions.

Remark

One of the dynamical systems can be polytopic or affine parameter-dependent. In this case, `sconnect` always returns a polytopic model of the interconnection.

Example 1



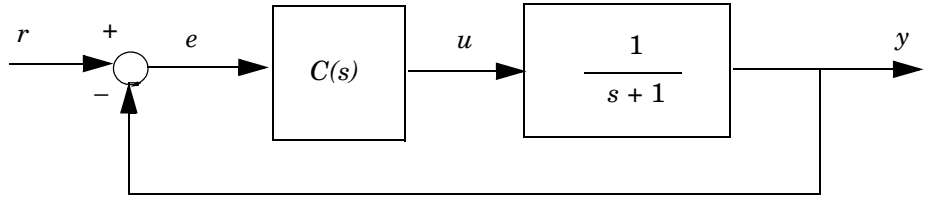
This simple example illustrates the use of `sconnect` to compute interconnections of systems without controller. The `SYSTEM` matrix of this series interconnection is returned by

```
sys1 = ltisys('tf',1,[1 1])
sys2 = ltisys('tf',[1 0],[1 2])
S = sconnect('r','S2',[],'S1:r',sys1,'S2:S1',sys2)
```

Note that `inputK` is set to `[]` to mark the absence of controller. The same result would be obtained with

```
S = smult(sys1,sys2)
```

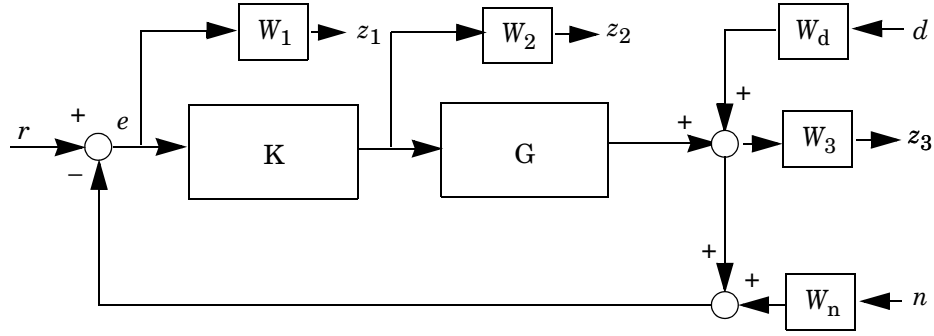
Example 2



The standard plant associated with this simple tracking loop is given by

```
g = ltisys('tf',1,[1 1])
P = sconnect('r','y=G','C:r-y','G:C',g)
```

Example 3



If the SYSTEM matrices of the system G and filters W_1, W_2, W_3, W_d, W_n are stored in the variables `g`, `w1`, `w2`, `w3`, `wd`, `wn`, respectively, the corresponding standard plant $P(s)$ is formed by

```
inputs = 'r;n;d'
outputs = 'W1;W2;W3'
Kin = 'K: e=r-y-Wn'
W3in = 'W3: y=G+Wd'
```

```
[P,r] = sconnect(inputs,outputs,Kin,'G:K',g,'W1:e',w1,...
                    'W2:K',w2,W3in,w3,'Wd:d',wd,'Wn:n',wn)
```

See Also ltisys, slft, ssub, sadd, smult

Purpose

Apply proportional-derivative action to some inputs/outputs of an LTI system

Syntax

```
dsys = sderiv(sys,chan,pd)
```

Description

`sderiv` multiplies selected inputs and/or outputs of the LTI system `sys` by the proportional-derivator $ns + d$. The coefficients n and d are specified by setting

```
pd = [n , d]
```

The second argument `chan` lists the input and output channels to be filtered by $ns + d$. Input channels are denoted by their ranking preceded by a minus sign.

On output, `dsys` is the `SYSTEM` matrix of the corresponding interconnection. An error is issued if the resulting system is not proper.

Example

Consider a SISO loop-shaping problem where the plant $P(s)$ has three outputs corresponding to the transfer functions S, KS, and T. Given the shaping filters

$$w_1(s) = \frac{1}{s}, \qquad w_2(s) = 100, \qquad w_3(s) = 1 + 0.01s,$$

the augmented plant associated with the criterion

$$\left\| \begin{pmatrix} w_1 S \\ w_2 KS \\ w_3 T \end{pmatrix} \right\|_\infty$$

is formed by

```
w1 = ltisys('tf',1,[1 0])
w2 = 100

paug = smult(p,sdiag(w1,w2,1))
paug = sderiv(paug,3,[0.01 1])
```

This last command multiplies the third output of P by the filter w_3.

See Also

`ltisys`, `magshape`

Purpose

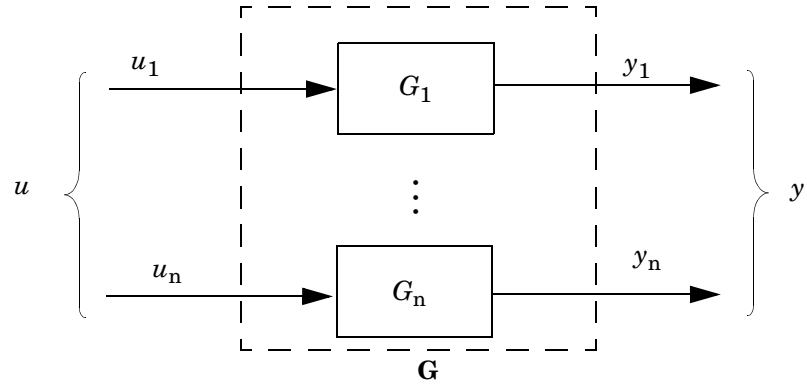
Append (concatenate) linear systems

Syntax

```
g = sdiag(g1,g2,...)
```

Description

`sdiag` returns the system obtained by stacking up the inputs and outputs of the systems `g1,g2,...` as in the diagram.



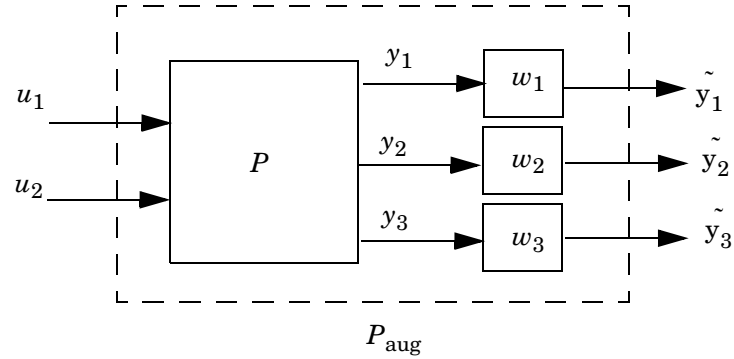
If $G_1(s)$,$G_2(s)$, . . . are the transfer functions of `g1,g2,...`, the transfer function of `g` is

$$G(s) = \begin{pmatrix} G_1(s) & 0 & 0 \\ 0 & G_2(s) & 0 \\ 0 & 0 & \ddots \end{pmatrix}$$

The function `sdiag` takes up to 10 input arguments. One (and at most one) of the systems `g1,g2,...` can be polytopic or parameter-dependent, in which case `g` is of the same nature.

Example

Let p be a system with two inputs u_1, u_2 and three outputs y_1, y_2, y_3. The augmented system



with weighting filters

$$w_1(s) = \frac{1}{s}, \qquad w_2(s) = 10, \qquad w_3(s) = \frac{100s}{s + 100}$$

is created by the commands

```
w1 = ltisys('tf',1,[1 0])
w3 = ltisys('tf',[100 0],[1 100])
paug = smult(p,sdiag(w1,10,w3))
```

See Also

`ltisys`, `ssub`, `sadd`, `smult`, `psys`

setlmis

Purpose

Initialize the description of an LMI system

Syntax

```
setlmis(lmi0)
```

Description

Before starting the description of a new LMI system with `lmivar` and `lmiterm`, type

```
setlmis([])
```

to initialize its internal representation.

To add on to an existing LMI system, use the syntax

```
setlmis(lmi0)
```

where `lmi0` is the internal representation of this LMI system. Subsequent `lmivar` and `lmiterm` commands will then add new variables and terms to the initial LMI system `lmi0`.

See Also

`getlmis`, `lmivar`, `lmiterm`, `newlmi`

Purpose

Instantiate a matrix variable and evaluate all LMI terms involving this matrix variable

Syntax

```
newsys = setmvar(lmisys,X,Xval)
```

Description

`setmvar` sets the matrix variable X with identifier `X` to the value `Xval`. All terms involving X are evaluated, the constant terms are updated accordingly, and X is removed from the list of matrix variables. A description of the resulting LMI system is returned in `newsys`.

The integer `X` is the identifier returned by `lmivar` when X is declared. Instantiating X with `setmvar` does not alter the identifiers of the remaining matrix variables.

The function `setmvar` is useful to freeze certain matrix variables and optimize with respect to the remaining ones. It saves time by avoiding partial or complete redefinition of the set of LMI constraints.

Example

Consider the system

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

and the problem of finding a stabilizing state-feedback law $u = Kx$ where K is an unknown matrix.

By the Lyapunov Theorem, this is equivalent to finding $P > 0$ and K such that

$$(A + BK)P + P(A + BK)^T + I < 0.$$

With the change of variable $Y := KP$, this condition reduces to the LMI

$$AP + PA^T + BY + Y^TB^T + I < 0.$$

This LMI is entered by the commands

```
n = size(A,1) % number of states
ncon = size(B,2) % number of inputs

setlmis([])
P = lmivar(1,[n 1]) % P full symmetric
Y = lmivar(2,[ncon n]) % Y rectangular

lmiterm([1 1 1 P],A,1,'s') % AP+PA'
```

```
lmiterm([1 1 1 Y],B,1,'s') % BY+Y'B'
lmiterm([1 1 1 0],1) % I
lmis = getlmis
```

To find out whether this problem has a solution K for the particular Lyapunov matrix $P = I$, set P to I by typing

```
news = setmvar(lmis,P,1)
```

The resulting LMI system `news` has only one variable $Y = K$. Its feasibility is assessed by calling `feasp`:

```
[tmin,xfeas] = feasp(news)
Y = dec2mat(news,xfeas,Y)
```

The computed Y is feasible whenever `tmin` < 0.

See Also

`evallmi`, `delmvar`

Purpose

Return the left- and right-hand sides of an LMI after evaluation of all variable terms

Syntax

```
[lhs,rhs] = showlmi(evalsys,n)
```

Description

For given values of the decision variables, the function `evallmi` evaluates all variable terms in a system of LMIs. The left- and right-hand sides of the `n`-th LMI are then constant matrices that can be displayed with `showlmi`. If `evalsys` is the output of `evallmi`, the values `lhs` and `rhs` of these left- and right-hand sides are given by

```
[lhs,rhs] = showlmi(evalsys,n)
```

An error is issued if `evalsys` still contains variable terms.

Example

See the description of `evallmi`.

See Also

`evallmi`, `setmvar`

sinfo

Purpose

Return the number of states, inputs, and outputs of an LTI system

Syntax

```
sinfo(sys)
[ns,ni,no] = sinfo(sys)
```

Description

For a linear system

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

$$y = Cx + Du$$

specified in the `SYSTEM` matrix `sys`, `sinfo` returns the order `ns` of the system (number of states) and the numbers `ni` and `no` of its inputs and outputs, respectively. Note that `ns, ni,` and `no` are the lengths of the vectors x, u, y, respectively.

Without output arguments, `sinfo` displays this information on the screen and also indicates whether the system is in descriptor form ($E \neq I$) or not.

See Also

`ltisys`, `ltiss`, `psinfo`

Purpose

Compute the inverse $G(s)^{-1}$, if it exists, of a system $G(s)$

Syntax

```
h = sinv(g)
```

Description

Given a system `g` with transfer function

$$G(s) = D + C(sE - A)^{-1}B$$

with D square and invertible, `sinv` returns the inverse system with transfer function $H(s) = G(s)^{-1}$. This corresponds to exchanging inputs and outputs:

$$y = G(s)u \quad \Leftrightarrow \quad u = H(s)y$$

See Also

`ltisys`, `sadd`, `smult`, `sdiag`

Purpose

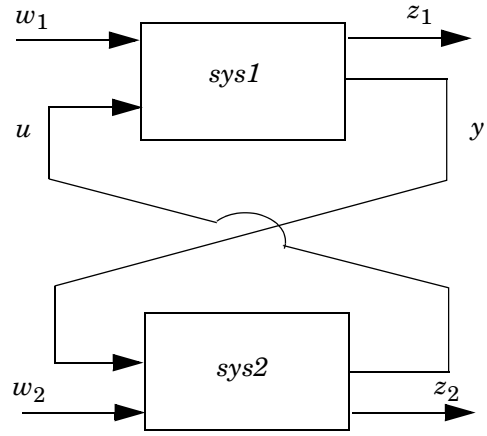
Form the linear-fractional interconnection of two time-invariant systems (Redheffer's star product)

Syntax

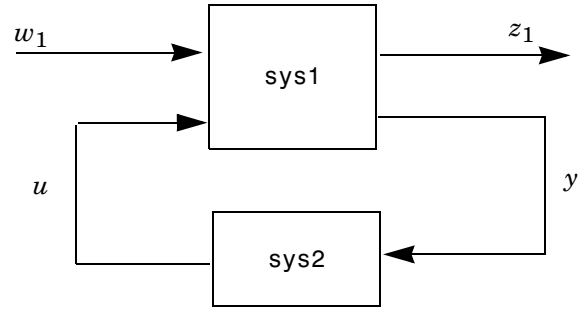
```
sys = slft(sys1,sys2,udim,ydim)
```

Description

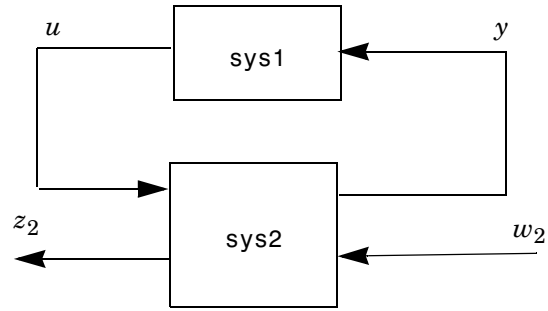
`slft` forms the linear-fractional feedback interconnection



of the two systems `sys1` and `sys2` and returns a realization `sys` of the closed-loop transfer function from $\begin{pmatrix} w_1 \\ w_2 \end{pmatrix}$ to $\begin{pmatrix} z_1 \\ z_2 \end{pmatrix}$. The optional arguments `udim` and `ydim` specify the lengths of the vectors u and y. Note that u enters `sys1` while y is an output of this system.When `udim` and `ydim` are omitted, `slft` forms one of the two interconnections:



or

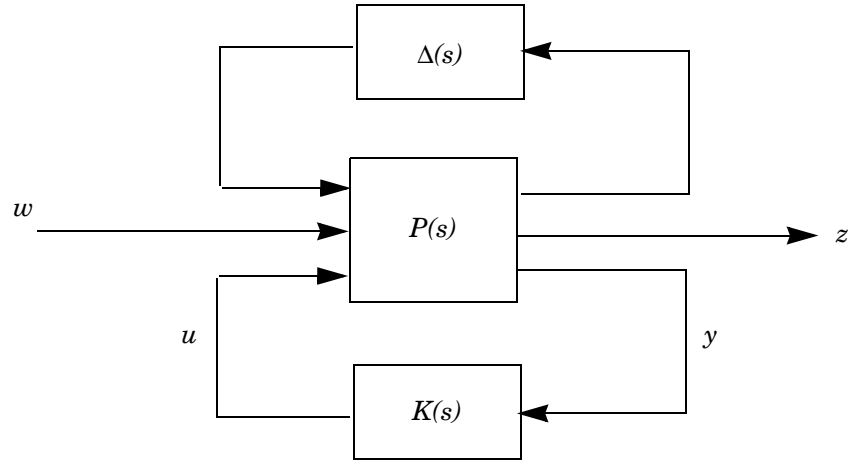


depending on which system has the larger number of inputs/outputs. An error is issued if neither of these interconnections can be performed.

`slft` also performs interconnections of polytopic or affine parameter-dependent systems. An error is issued if the result is neither polytopic nor affine parameter-dependent.

Example

Consider the interconnection



where $\Delta(s)$, $P(s)$, $K(s)$ are LTI systems. A realization of the closed-loop transfer function from w to z is obtained as

```
clsys = slft(delta,slft(p,k))
```

or equivalently as

```
clsys = slft(slft(delta,p),k)
```

See Also

`ltisys`, `sloop`, `sconnect`

Purpose

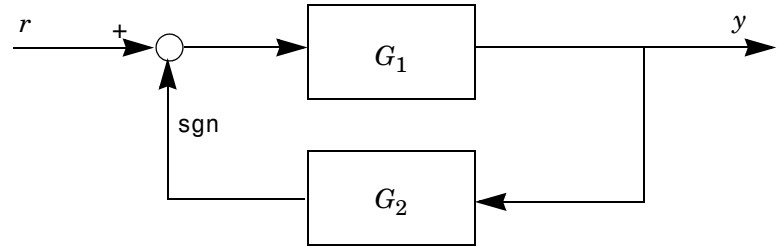
Form the feedback interconnection of two systems

Syntax

```
sys = sloop(g1,g2,sgn)
```

Description

`sloop` forms the interconnection

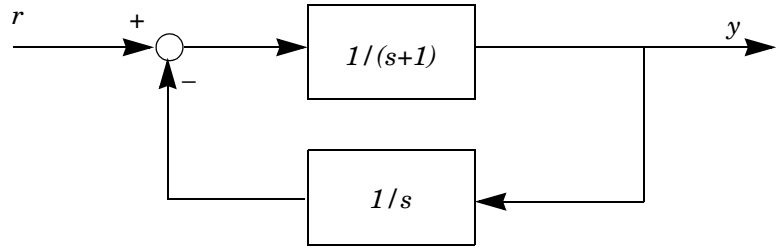


The output `sys` is a realization of the closed-loop transfer function from r to y. The third argument `sgn` specifies either negative feedback (`sgn = -1`) or positive feedback (`sgn = +1`). The default value is `-1`.

In terms of the transfer functions $G_1(s)$ and $G_2(s)$, `sys` corresponds to the transfer function $(I - \varepsilon G_1 G_2)^{-1} G_1$ where ε =`sgn`.

Example

The closed-loop transfer function from r to y in



is obtained by

```
sys = sloop( ltisys('tf',1,[1 1]) , ltisys('tf',1,[1 0]))

[num,den] = ltitf(sys)

num =
```

```
          0    1.00     0.00

   den =
      1.00     1.00     1.00
```

See Also

ltisys, slft, sconnect

Purpose

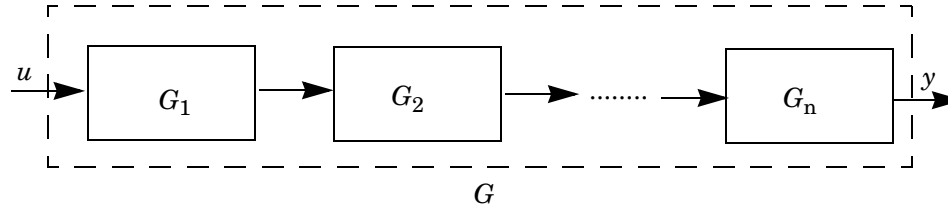
Series interconnection of linear systems

Syntax

```
sys = smult(g1,g2,...)
```

Description

`smult` forms the series interconnection



The arguments `g1, g2,...` are `SYSTEM` matrices containing the state-space data for the systems $G_1, \ldots, G_n$. Constant matrices are also allowed as a representation of static gains. In addition, one (and at most one) of the input arguments can be a polytopic or affine parameter-dependent model, in which case the output `sys` is of the same nature.

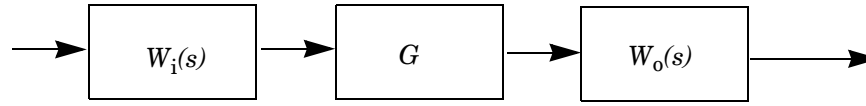
In terms of transfer functions, this interconnection corresponds to the product

$$G(s) = G_n(s) \times \ldots \times G_1(s)$$

Note the reverse order of the terms.

Example

Consider the interconnection



where $W_i(s)$, $W_0(s)$ are given LTI filters and G is the affine parameter-dependent system

$$\dot{x} = \theta_1 x + u$$

$$y = x - \theta_2 u$$

parametrized by θ_1, θ_2. The parameter-dependent system defined by this interconnection is returned by

```
sys = smult(wi,g,wO)
```

While the SYSTEM matrices `wi` and `wO` are created with `ltisys`, the parameter-dependent system `g` should be specified with `psys`.

See Also

`sadd`, `sdiag`, `sloop`, `slft`, `ltisys`

Purpose

Plot the various frequency and time responses of LTI systems

Syntax

```
splot(sys,type,xrange)
splot(sys,T,type,xrange)
```

Description

The first syntax is for continuous-time systems

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

$$y = Cx + Du$$

The first argument `sys` is the `SYSTEM` matrix representation of a system. The optional argument `xrange` is a vector of frequencies or times used to control the x-axis and the number of points in the plot. Finally, the string `type` consists of the first two characters of the diagram to be plotted. Available choices include:

Frequency Response	
type	**plot**
`'bo'`	Bode plot
`'sv'`	Singular value plot
`'ny'`	Nyquist plot
`'li'`	lin-log Nyquist plot
`'ni'`	Black/Nichols chart

Time Response	
type	**plot**
`'st'`	Step response
`'im'`	Impulse response
`'sq'`	Response to a square signal
`'si'`	Response to a sinusoidal signal

The syntax `splot(sys,T,type,xrange)` is used for discrete-time systems, in which case `T` is the sampling period in seconds.

Remark

The Control System Toolbox is needed to run `splot`.

Example

Consider the third-order SISO system with transfer function

$$G(s) = \frac{1}{s(s^2 + 0.001s + 1)}$$

To plot the Nyquist diagram of the frequency response $G(j\omega)$, type

```
g = ltisys('tf',1,[1 0.001 1 0])
splot(g,'ny')
```

The resulting plot appears in Figure 9-3. Due to the integrator and poor damping of the natural frequency at 1 rd/s, this plot is not very informative. In such cases, the lin-log Nyquist plot is more appropriate. Given the gain/phase decomposition

$$G(j\omega) = \gamma(\omega)e^{j\phi(\omega)}$$

of the frequency response, the lin-log Nyquist plot is the curve described by

$$x = \rho(\omega)\cos\varphi(\omega), \qquad y = \rho(\omega)\sin\varphi(\omega)$$

where

$$p(\omega) = \begin{cases} \gamma\omega & \text{if } \gamma\omega \geq 1 \\ 1 + \log_{10}\gamma\omega & \text{if } \gamma\omega > 1 \end{cases}$$

This plot is drawn by the command

```
splot(g,'li')
```

and appears in Figure 9-4. The resulting aspect ratio is more satisfactory thanks to the logarithmic scale for gains larger than one. A more complete contour is drawn in Figure 9-5 by

```
splot(g,'li',logspace(-3,2))
```

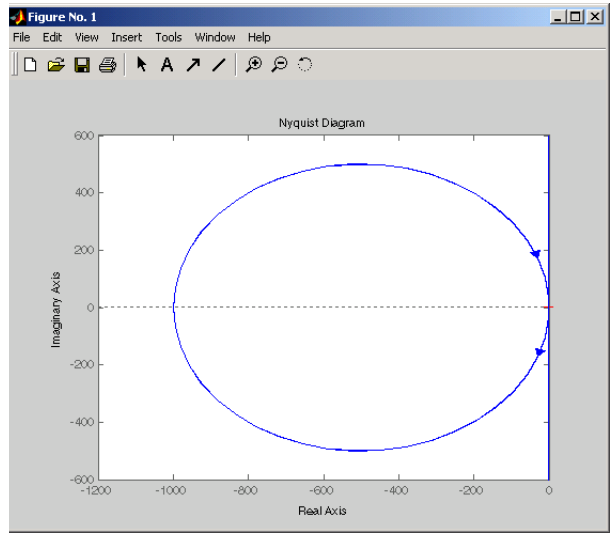


Figure 9-3: splot(g,'ny')

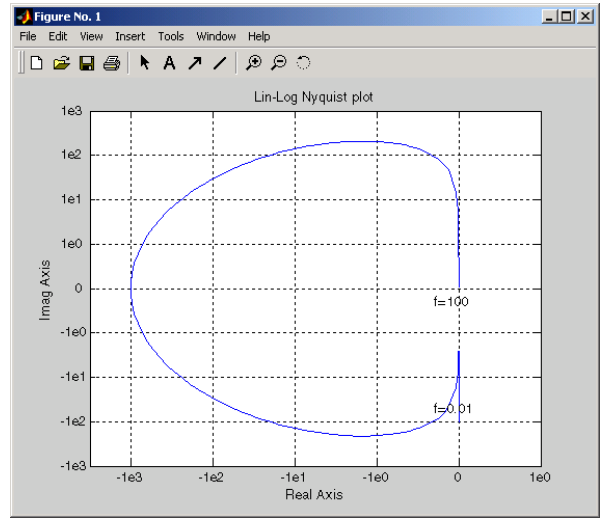


Figure 9-4: splot(g,'li')

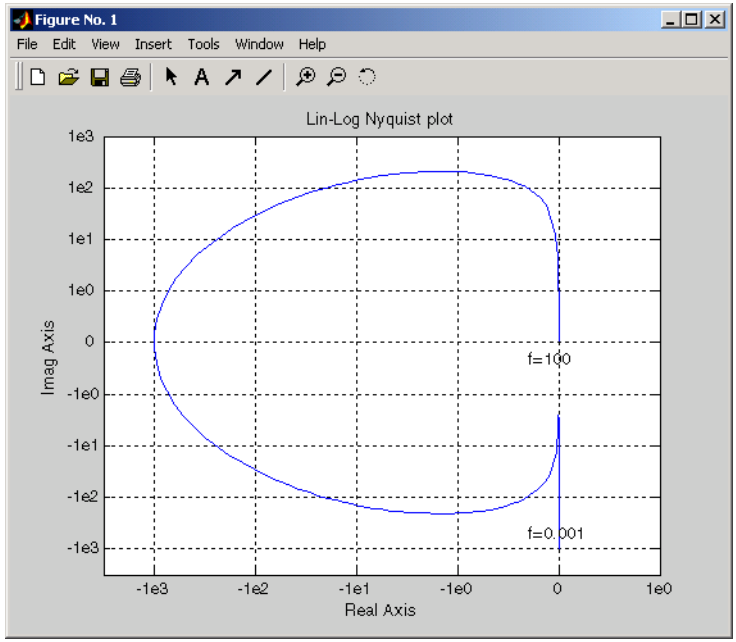


Figure 9-5: splot(g,'li',logspace(-3,2))

See Also

`ltisys`, `bode`, `nyquist`, `sigma`, `step`, `impulse`

Purpose

Return the poles of an LTI system

Syntax

```
poles = spol(sys)
```

Description

The function `spol` computes the poles of the LTI system of `SYSTEM` matrix `sys`. If (*A, B, C, D, E*) denotes the state-space realization of this system, the poles are the generalized eigenvalues of the pencil (*A, E*) in general, and the eigenvalues of *A* when *E* = *I*.

See Also

`ltisys`, `ltiss`

sresp

Purpose

Frequency response of a continuous-time system

Syntax

```
resp = sresp(sys,f)
```

Description

Given a continuous-time system $G(s)$ of realization (A, B, C, D, E), `sresp` computes its frequency response

$$G(j\omega) = D + C(j\omega E - A)^{-1}B$$

at the frequency ω = `f`. The first input `sys` is the `SYSTEM` matrix of G.

See Also

`ltisys`, `splot`, `spol`

Purpose

Select particular inputs and outputs in a MIMO system

Syntax

```
subsys = ssub(sys,inputs,outputs)
```

Description

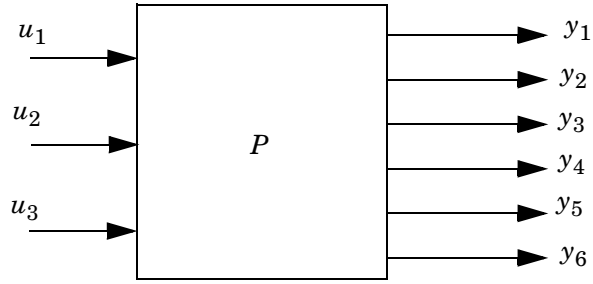
Given an LTI or polytopic/parameter-dependent system `sys`, `ssub` extracts the subsystem `subsys` mapping the inputs selected by `inputs` to the outputs selected by `outputs`. The resulting subsystem has the same number of states as `sys` even though some of these states may no longer be controllable/observable from the remaining inputs and outputs.

Example

If `p` is a system with 3 inputs and 6 outputs, a realization of the transfer function from the input channels 1,3 to the output channels 4,5,6 is obtained as:

```
ssub(p,[1,3],4:6)
```

The second and third arguments are the vectors of indices of the selected inputs/outputs. Inputs and outputs are labeled 1,2,3,... starting from the top of the block diagram:



See Also

`ltisys`, `psys`

ublock

Purpose

Specify the characteristics of an uncertainty block

Syntax

```
block = ublock(dims,bound,type)
```

Description

`ublock` is used to describe individual uncertainty blocks Δi in a block-diagonal uncertainty operator

$$\Delta = \mathrm{diag}(\Delta_1, \ldots, \Delta_n)$$

Such operators arise in the linear-fractional representation of uncertain dynamical systems (see "How to Derive Such Models" on page 2-23).

The first argument `dims` specifies the dimensions of Δi. Set `dim = [m,n]` if Δ_i is m-by-n, i.e., has m outputs and n inputs. The second argument `bound` contains the quantitative information about the uncertainty (norm bound or sector bounds). Finally, `type` is a string specifying the dynamical and structural characteristics of Δ_i.

The string `type` consists of one- or two-letter identifiers marking the uncertainty block properties. Characteristic properties include:

- The dynamical nature: linear time invariant (`'lti'`), linear time varying (`'ltv'`), nonlinear memoryless (`'nlm'`), or arbitrary nonlinear (`'nl'`). The default is `'lti'`
- The structure: `'f'` for full (unstructured) blocks and `'s'` for scalar blocks of the form

  $\Delta_i = \delta_i \, x \, I_{r_i}$

  Scalar blocks often arise when representing parameter uncertainty. They are also referred to as repeated scalars. The default is `'f'`
- The phase information: the block can be complex-valued (`'c'`) or real-valued (`'r'`). Complex-valued means arbitrary phase. The default is `'c'`.

For instance, a real-valued time-varying scalar block is specified by setting `type = 'tvsr'`. The order of the property identifiers is immaterial.

Finally, the quantitative information can be of two types:

- Norm bound: here the second argument bound is either a scalar β for uniform bounds,

$\|\Delta\|_\infty < \beta$,

or a SISO shaping filter $W(s)$ for frequency-weighted bounds $\|W^{-1}\Delta\|_\infty < 1$, that is,

$\sigma_{max}(\Delta(j\omega)) < |W(j\omega)|$ for all ω

- Sector bounds: set `bound = [a b]` to indicate that the response of $\Delta(\cdot)$ lies in the sector $\{a, b\}$. Valid values for a and b include `Inf` and `Inf`.

Example

See the example on p. 2-25 of this manual.

See Also

`udiag`, `uinfo`, `slft`, `aff2lft`

udiag

Purpose

Form block-diagonal uncertainty structures

Syntax

```
delta = udiag(delta1,delta2,....)
```

Description

The function `udiag` appends individual uncertainty blocks specified with `ublock` and returns a complete description of the block-diagonal uncertainty operator

$$\Delta = \mathrm{diag}(\Delta_1, \Delta_2, \ldots)$$

The input arguments `delta1, delta2,...` are the descriptions of $\Delta_1, \Delta_2, \ldots$. If `udiag` is called with k arguments, the output `delta` is a matrix with k columns, the k-th column being the description of Δ_k.

See Also

`ublock`, `udiag`, `slft`, `aff2lft`

Purpose

Display the characteristics of a block-diagonal uncertainty structure

Syntax

```
uinfo(delta)
```

Description

Given the description `delta` of a block-diagonal uncertainty operator

$$\Delta = \mathrm{diag}(\Delta_1, \ldots, \Delta_n).$$

as returned by `ublock` and `udiag`, the function `uinfo` lists the characteristics of each uncertainty block Δ_i.

See Also

`ublock`, `udiag`