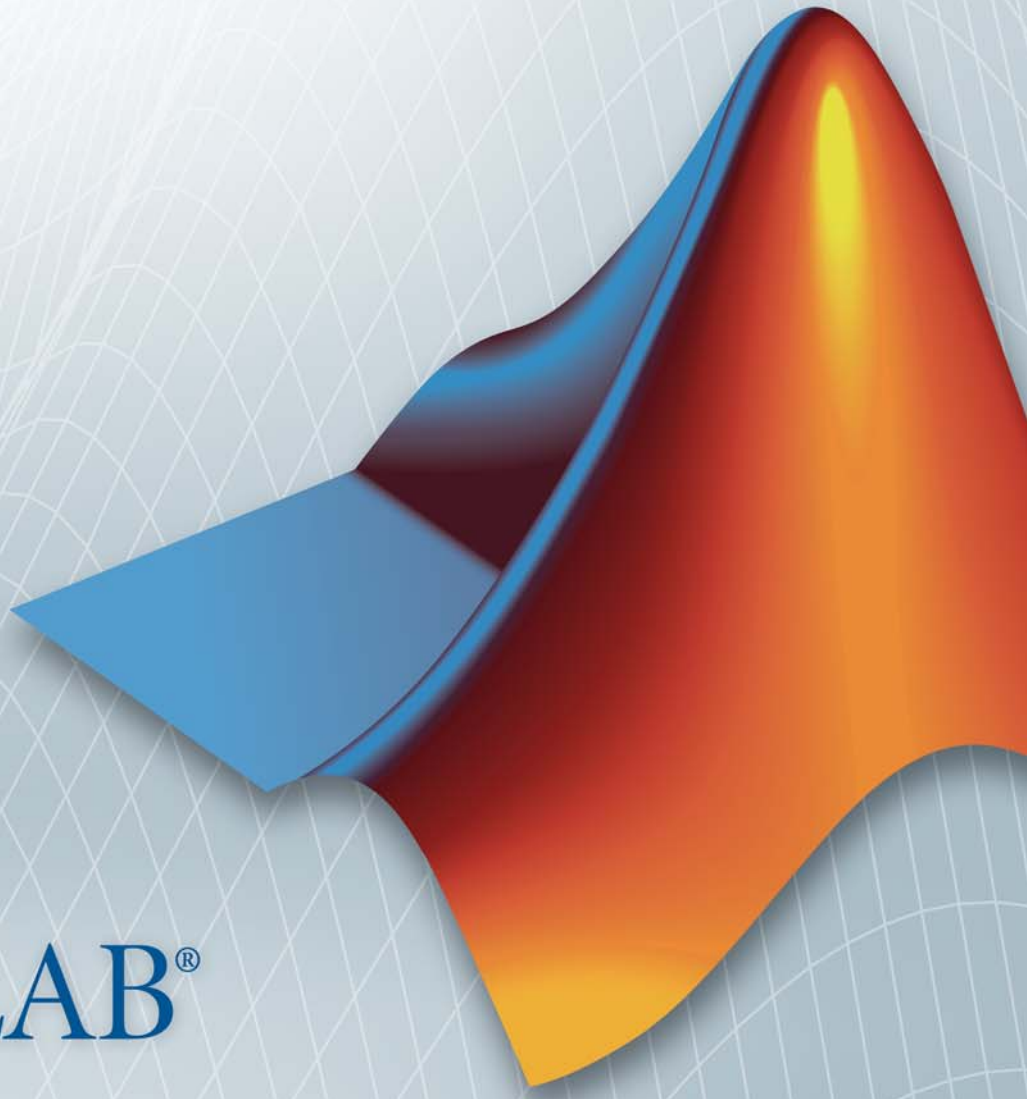
Robust Control Toolbox™

Getting Started Guide

R2011b

Gary Balas
Richard Chiang
Andy Packard
Michael Safonov



MATLAB®

MathWorks®

How to Contact MathWorks

`www.mathworks.com`	Web
`comp.soft-sys.matlab`	Newsgroup
`www.mathworks.com/contact_TS.html`	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.
3 Apple Hill Drive
Natick, MA 01760-2098

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

Robust Control Toolbox™ Getting Started Guide

Revision History

September 2005	First printing	New for Version 3.0.2 (Release 14SP3)
March 2006	Online only	Revised for Version 3.1 (Release 2006a)
September 2006	Online only	Revised for Version 3.1.1 (Release 2006b)
March 2007	Online only	Revised for Version 3.2 (Release 2007a)
September 2007	Online only	Revised for Version 3.3 (Release 2007b)
March 2008	Online only	Revised for Version 3.3.1 (Release 2008a)
October 2008	Online only	Revised for Version 3.3.2 (Release 2008b)
March 2009	Online only	Revised for Version 3.3.3 (Release 2009a)
September 2009	Online only	Revised for Version 3.4 (Release 2009b)
March 2010	Online only	Revised for Version 3.4.1 (Release 2010a)
September 2010	Online only	Revised for Version 3.5 (Release 2010b)
April 2011	Online only	Revised for Version 3.6 (Release 2011a)
September 2011	Online only	Revised for Version 4.0 (Release 2011b)

Contents

1 Introduction

2 Multivariable Loop Shaping

3 Model Reduction for Robust Control

4 Robustness Analysis

5 H-Infinity and Mu Synthesis

6 Tuning Fixed Control Architectures

Examples

A

Index

1

Introduction

Product Overview

Robust Control Toolbox™ provides tools for analyzing and automatically tuning control systems for performance and robustness. You can create uncertain models by combining nominal dynamics with uncertain elements, such as an uncertain parameter or unmodeled dynamics. You can analyze the impact of plant model uncertainty on control system performance and identify worst-case combinations of uncertain elements. Using H-infinity or mu-synthesis techniques, you can design controllers that maximize robust stability and performance. The toolbox can automatically tune both SISO and MIMO robust controllers, including decentralized control architectures modeled in Simulink®. You can validate your design by calculating worst-case gain and phase margins and worst-case sensitivity to disturbances.

Required Software

Robust Control Toolbox software requires that you have installed Control System Toolbox™ software.

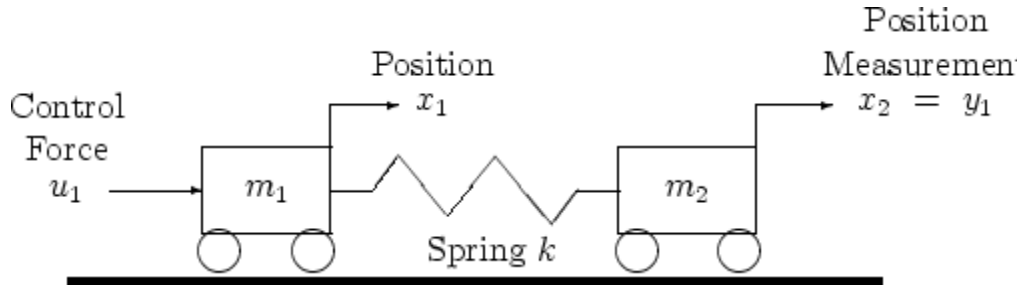
Modeling Uncertainty

At the heart of robust control is the concept of an uncertain LTI system. Model uncertainty arises when system gains or other parameters are not precisely known, or can vary over a given range. Examples of real parameter uncertainties include uncertain pole and zero locations and uncertain gains. You can also have unstructured uncertainties, by which is meant complex parameter variations satisfying given magnitude bounds.

With Robust Control Toolbox software you can create uncertain LTI models as MATLAB® objects specifically designed for robust control applications. You can build models of complex systems by combining models of subsystems using addition, multiplication, and division, as well as with Control System Toolbox commands like `feedback` and `lft`.

Example: ACC Benchmark Problem

For instance, consider the two-cart "ACC Benchmark" system [13] consisting of two frictionless carts connected by a spring shown as follows.



ACC Benchmark Problem

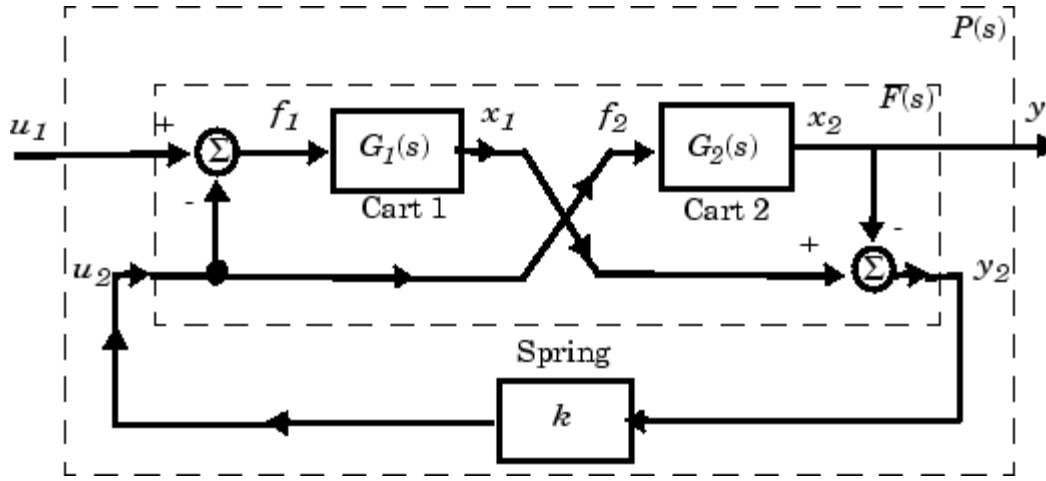
The system has the block diagram model shown below, where the individual carts have the respective transfer functions.

$$G_1(s) = \frac{1}{m_1 s^2}$$

$$G_2(s) = \frac{1}{m_2 s^2}.$$

The parameters m_1, m_2, and k are uncertain, equal to one plus or minus 20%:

```
m1 = 1 - 0.2
m2 = 1 - 0.2
k = 1 - 0.2
```



"ACC Benchmark" Two-Cart System Block Diagram $y_1 = P(s)\,u_1$

The upper dashed-line block has transfer function matrix $F(s)$:

$$F(s) = \begin{bmatrix} 0 \\ G_1(s) \end{bmatrix} \begin{bmatrix} 1 & -1 \end{bmatrix} + \begin{bmatrix} 1 \\ -1 \end{bmatrix} \begin{bmatrix} 0 & G_2(s) \end{bmatrix}.$$

This code builds the uncertain system model `P` shown above:

```
% Create the uncertain real parameters m1, m2, & k
m1 = ureal('m1',1,'percent',20);
m2 = ureal('m2',1,'percent',20);
k  = ureal('k',1,'percent',20);

s = zpk('s'); % create the Laplace variable s
G1 = ss(1/s^2)/m1; % Cart 1
G2 = ss(1/s^2)/m2; % Cart 2

% Now build F and P
```

```
F = [0;G1]*[1 -1]+[1;-1]*[0,G2];
P = lft(F,k) % close the loop with the spring k
```

The variable `P` is a SISO uncertain state-space (USS) object with four states and three uncertain parameters, `m1`, `m2`, and `k`. You can recover the nominal plant with the command

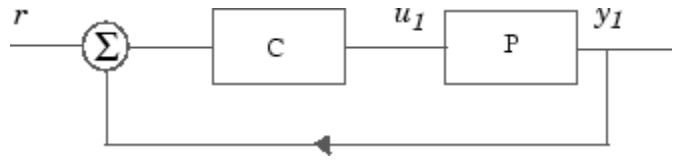
```
zpk(P.nominal)
```

which returns

```
Zero/pole/gain:
      1
--------------
s^2 (s^2  + 2)
```

If the uncertain model P(s) has LTI negative feedback controller

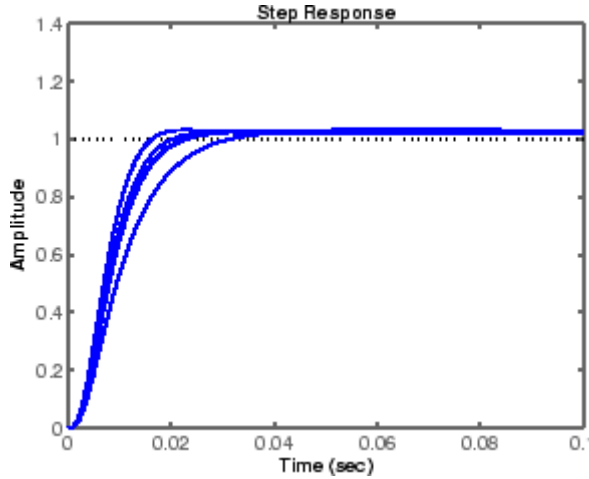
$$C(s) = \frac{100(s+1)^3}{(0.001s+1)^3}$$



then you can form the controller and the closed-loop system $y_1 = T(s)\ u_1$ and view the closed-loop system's step response on the time interval from t=0 to t=0.1 for a Monte Carlo random sample of five combinations of the three uncertain parameters `k`, `m1`, and `m2` using this code:

```
C=100*ss((s+1)/(.001*s+1))^3 % LTI controller
T=feedback(P*C,1); % closed-loop uncertain system
step(usample(T,5),.1);
```

The resulting plot is shown below.



Monte Carlo Sampling of Uncertain System's Step Response

Worst-Case Performance

To be robust, your control system should meet your stability and performance requirements for all possible values of uncertain parameters. Monte Carlo parameter sampling via `usample` can be used for this purpose as shown in Monte Carlo Sampling of Uncertain System's Step Response on page 1-6, but Monte Carlo methods are inherently hit or miss. With Monte Carlo methods, you might need to take an impossibly large number of samples before you hit upon or near a worst-case parameter combination.

Robust Control Toolbox software gives you a powerful assortment of *robustness analysis* commands that let you directly calculate upper and lower bounds on worst-case performance without random sampling.

Worst-Case Robustness Analysis Commands	
`loopmargin`	Comprehensive analysis of feedback loop
`loopsens`	Sensitivity functions of feedback loop
`ncfmargin`	Normalized coprime stability margin of feedback loop
`robustperf`	Robust performance of uncertain systems
`robuststab`	Stability margins of uncertain systems
`wcgain`	Worst-case gain of an uncertain system
`wcmargin`	Worst-case gain/phase margins for feedback loop
`wcsens`	Worst-case sensitivity functions of feedback loop

Example: ACC Two-Cart Benchmark Problem

Returning to the "Example: ACC Benchmark Problem" on page 1-3, the closed loop system is:

```
T=feedback(P*C,1); % Closed-loop uncertain system
```

This uncertain state-space model `T` has three uncertain parameters, `k`, `m1`, and `m2`, each equal to 1±20% uncertain variation. To analyze whether the closed-loop system `T` is robustly stable for all combinations of values for these three parameters, you can execute the commands:

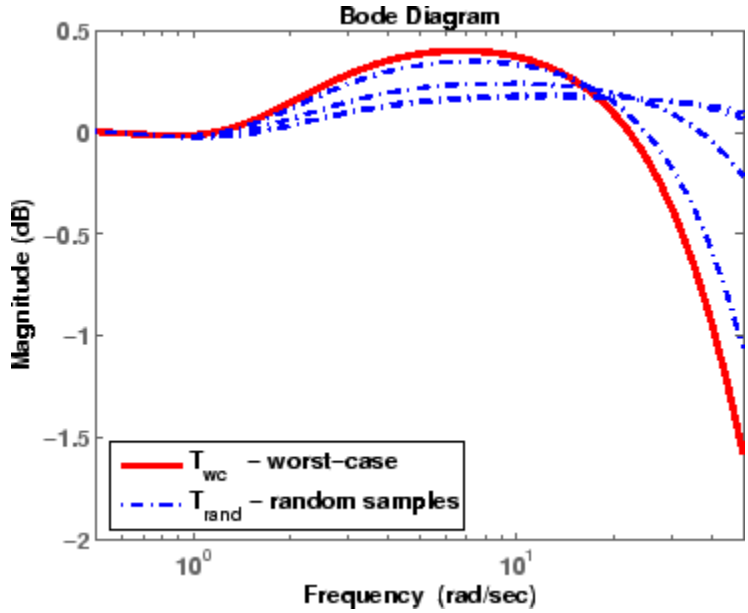
```
[StabilityMargin,Udestab,REPORT] = robuststab(T);
REPORT
```

This displays the `REPORT`:

```
Uncertain System is robustly stable to modeled uncertainty.
 -- It can tolerate up to 311% of modeled uncertainty.
 -- A destabilizing combination of 500% the modeled uncertainty exists,
    causing an instability at 44.3 rad/s.
```

The report tells you that the control system is robust for all parameter variations in the ±20% range, and that the smallest destabilizing combination of real variations in the values `k`, `m1`, and `m2` has sizes somewhere between 311% and 500% greater than ±20%, i.e., between ±62.2% and ±100%. The value `Udestab` returns an estimate of the 500% destabilizing parameter variation combination:

```
Udestab =
     k: 1.2174e-005
    m1: 1.2174e-005
    m2: 2.0000.
```



Uncertain System Closed-Loop Bode Plots

You have a comfortable safety margin of between 311% to 500% larger than the anticipated ±20% parameter variations before the closed loop goes unstable. But how much can closed-loop performance deteriorate for parameter variations constrained to lie strictly within the anticipated ±20% range? The following code computes worst-case peak gain of `T`, and estimates the frequency and parameter values at which the peak gain occurs:

```
[PeakGain,Uwc] = wcgain(T);
Twc=usubs(T,Uwc);
% Worst case closed-loop system T
Trand=usample(T,4);
% 4 random samples of uncertain system T
bodemag(Twc,'r',Trand,'b-.',{.5,50}); % Do bode plot
legend('T_{wc}   - worst-case',...
'T_{rand} - random samples',3);
```

The resulting plot is shown in Uncertain System Closed-Loop Bode Plots on page 1-9.

Synthesis of Robust MIMO Controllers

You can design controllers for multiinput-multioutput (MIMO) LTI models with your Robust Control Toolbox software using the following command.

Robust Control Synthesis Commands	
`h2hinfsyn`	Mixed H_2/H_∞ controller synthesis
`h2syn`	H_2 controller synthesis
`hinfsyn`	H_∞ controller synthesis
`loopsyn`	H_∞ loop-shaping controller synthesis
`ltrsyn`	Loop-transfer recovery controller synthesis
`mixsyn`	H_∞ mixed-sensitivity controller synthesis
`ncfsyn`	H_∞ normalized coprime factor controller synthesis
`sdhinfsyn`	Sampled-data H_∞ controller synthesis

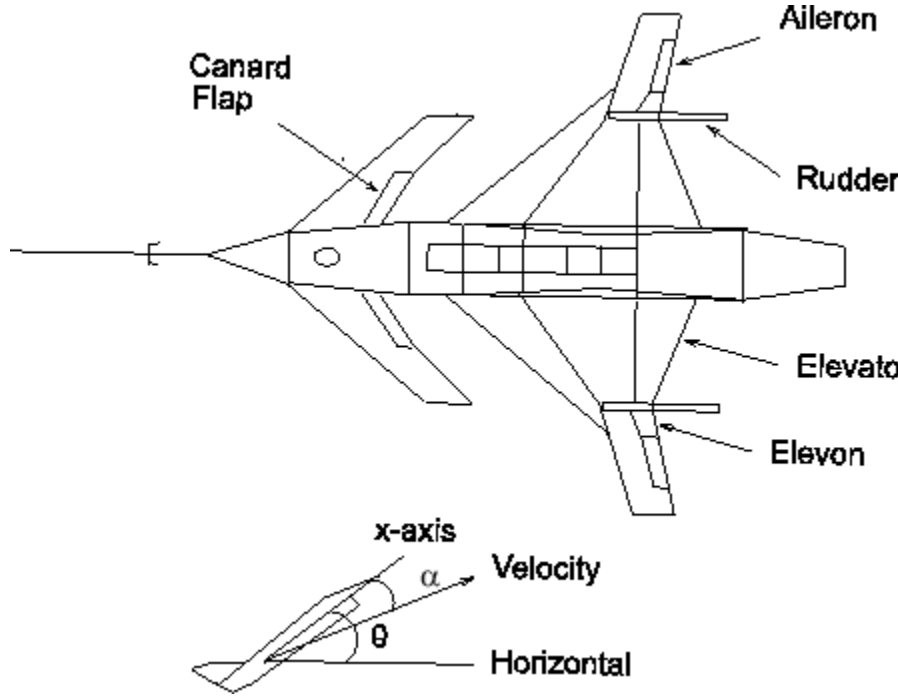
Example: Designing a Controller with LOOPSYN

One of the most powerful yet simple controller synthesis tools is `loopsyn`. Given an LTI plant, you specify the shape of the open-loop systems frequency response plot that you want, then `loopsyn` computes a stabilizing controller that best approximates your specified loop shape.

For example, consider the 2-by-2 NASA HiMAT aircraft model (Safonov, Laub, and Hartmann [8]) depicted in the following figure. The control variables are elevon and canard actuators (δ_e and δ_c). The output variables are angle of attack (α) and attitude angle (θ). The model has six states:

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{bmatrix} = \begin{bmatrix} \dot{\alpha} \\ \alpha \\ \dot{\theta} \\ \theta \\ x_e \\ x_\delta \end{bmatrix}$$

where x_e and x_δ are elevator and canard actuator states.



Aircraft Configuration and Vertical Plane Geometry

You can enter the state-space matrices for this model with the following code:

```
% NASA HiMAT model G(s)
ag =[ -2.2567e-02  -3.6617e+01  -1.8897e+01  -3.2090e+01   3.2509e+00  -7.6257e-01;
       9.2572e-05  -1.8997e+00   9.8312e-01  -7.2562e-04  -1.7080e-01  -4.9652e-03;
       1.2338e-02   1.1720e+01  -2.6316e+00   8.7582e-04  -3.1604e+01   2.2396e+01;
       0            0   1.0000e+00            0            0            0;
       0            0            0            0  -3.0000e+01            0;
       0            0            0            0            0  -3.0000e+01];
bg = [ 0     0;
       0     0;
       0     0;
       0     0;
       30     0;
```

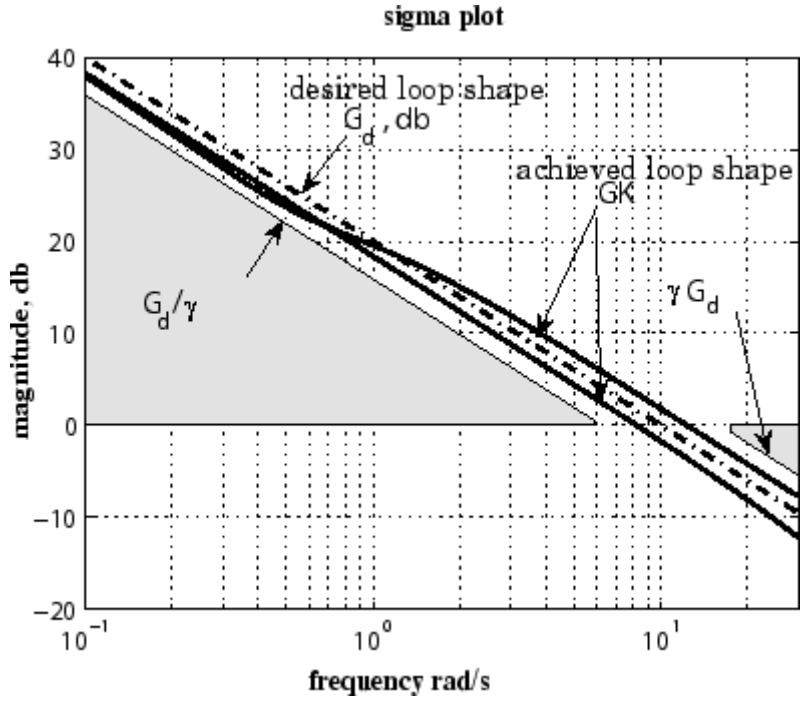
```
       0    30];
cg = [ 0     1     0     0     0     0;
       0     0     0     1     0     0];
dg = [ 0     0;
       0     0];
G=ss(ag,bg,cg,dg);
```

To design a controller to shape the frequency response (`sigma`) plot so that the system has approximately a bandwidth of 10 rad/s, you can set as your target desired loop shape $G_d(s)$=10/s, then use `loopsyn(G,Gd)` to find a loop-shaping controller for `G` that optimally matches the desired loop shape `Gd` by typing:

```
s=zpk('s'); w0=10; Gd=w0/(s+.001);
[K,CL,GAM]=loopsyn(G,Gd); % Design a loop-shaping controller K

% Plot the results
sigma(G*K,'r',Gd,'k-.',Gd/GAM,'k:',Gd*GAM,'k:',{.1,30})
figure ;T=feedback(G*K,eye(2));
sigma(T,ss(GAM),'k:',{.1,30});grid
```

The value of γ= `GAM` returned is an indicator of the accuracy to which the optimal loop shape matches your desired loop shape and is an upper bound on the resonant peak magnitude of the closed-loop transfer function `T=feedback(G*K,eye(2))`. In this case, γ = 1.6024 = 4 dB — see the next figure.



MIMO Robust Loop Shaping with `loopsyn(G,Gd)`

The achieved loop shape matches the desired target `Gd` to within about γ dB.

Model Reduction and Approximation

Complex models are not always required for good control. Unfortunately, however, optimization methods (including methods based on H_∞, H_2, and μ-synthesis optimal control theory) generally tend to produce controllers with at least as many states as the plant model. For this reason, Robust Control Toolbox software offers you an assortment of model-order reduction commands that help you to find less complex low-order approximations to plant and controller models.

Model Reduction Commands	
`reduce`	Main interface to model approximation algorithms
`balancmr`	Balanced truncation model reduction
`bstmr`	Balanced stochastic truncation model reduction
`hankelmr`	Optimal Hankel norm model approximations
`modreal`	State-space modal truncation/realization
`ncfmr`	Balanced normalized coprime factor model reduction
`schurmr`	Schur balanced truncation model reduction
`slowfast`	State-space slow-fast decomposition
`stabsep`	State-space stable/antistable decomposition
`imp2ss`	Impulse response to state-space approximation

Among the most important types of model reduction methods are minimize bounds methods on additive, multiplicative, and normalized coprime factor (NCF) model error. You can access all three of these methods using the command reduce.

Example: NASA HiMAT Controller Order Reduction

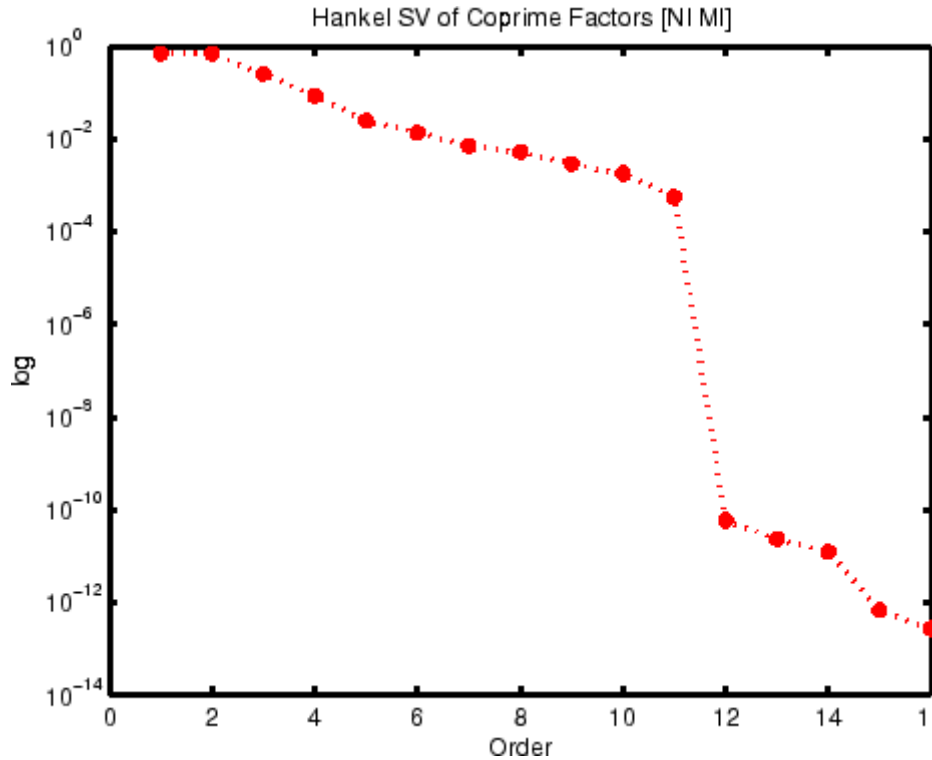
For instance, the NASA HiMAT model considered in the last section has eight states, and the optimal loop-shaping controller turns out to have 16 states. Using model reduction, you can remove at least some of the states without appreciably affecting stability or closed-loop performance. For controller

order reduction, the NCF model reduction is particularly useful, and it works equally well with controllers that have poles anywhere in the complex plane.

For the NASA HiMAT design in the last section, you can type

```
hankelsv(K,'ncf','log');
```

which displays a logarithmic plot of the NCF Hankel singular values — see the following figure.

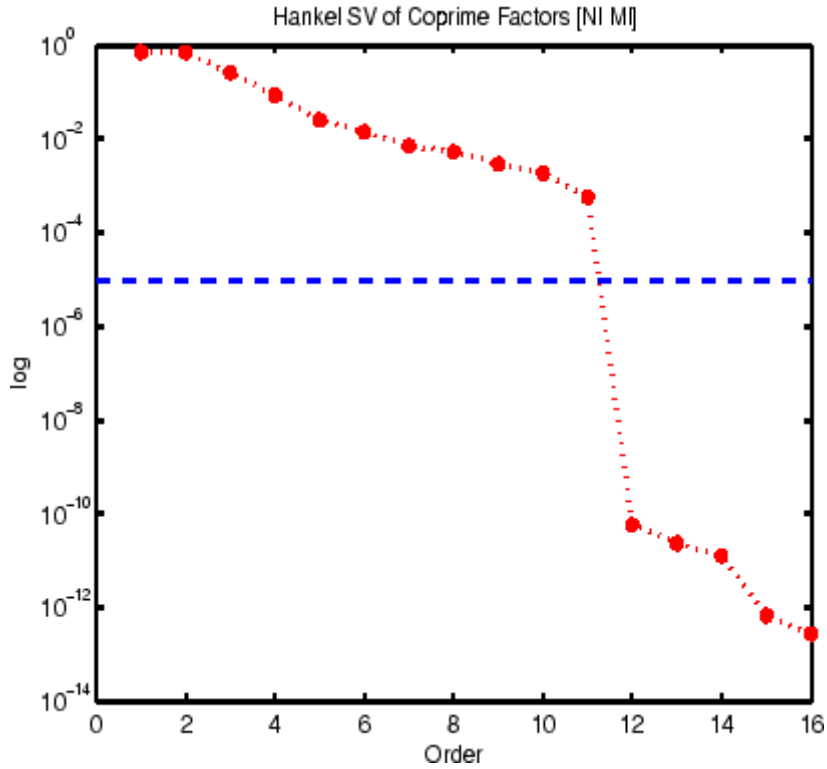


Hankel Singular Values of Coprime Factorization of K

Theory says that, without danger of inducing instability, you can confidently discard at least those controller states that have NCF Hankel singular values that are much smaller than `ncfmargin(G,K)`.

Compute `ncfmargin(G,K)` and add it to your Hankel singular values plot.

```
hankelsv(K,'ncf','log');v=axis;
hold on; plot(v(1:2), ncfmargin(G,K)*[1 1],'--'); hold off
```



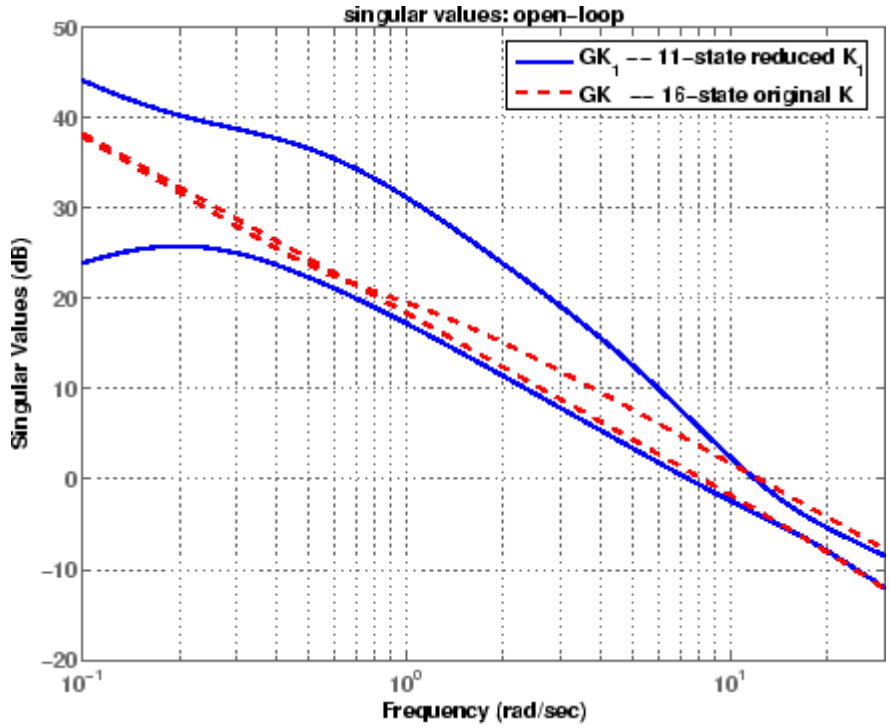
Five of the 16 NCF Hankel Singular Values of HiMAT Controller K Are Small Compared to ncfmargin(G,K)

In this case, you can safely discard 5 of the 16 states of `K` and compute an 11-state reduced controller by typing:

```
K1=reduce(K,11,'errortype','ncf');
```

The result is plotted in the following figure.

```
sigma(G*K1,'b',G*K,'r--',{.1,30});
```



HiMAT with 11-State Controller K1 vs. Original 16-State Controller K

The picture above shows that low-frequency gain is decreased considerably for inputs in one vector direction. Although this does not affect stability, it affects performance. If you wanted to better preserve low-frequency performance, you would discard fewer than five of the 16 states of K.

LMI Solvers

At the core of many emergent robust control analysis and synthesis routines are powerful general-purpose functions for solving a class of convex nonlinear programming problems known as linear matrix inequalities. The LMI capabilities are invoked by Robust Control Toolbox software functions that evaluate worst-case performance, as well as functions like `hinfsyn` and `h2hinfsyn`. Some of the main functions that help you access the LMI capabilities of the toolbox are shown in the following table.

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`	Test feasibility of a system of LMIs
`gevp`	Minimize generalized eigenvalue with LMI constraints
`mincx`	Minimize a linear objective with LMI constraints
`dec2mat`	Convert output of the solvers to values of matrix variables

Evaluation of LMIs/Validation of Results	
`evallmi`	Evaluate for given values of the decision variables
`showlmi`	Return the left and right sides of an evaluated LMI

Complete documentation is available in “LMI Lab”.

Extends Control System Toolbox Capabilities

Robust Control Toolbox software is designed to work with Control System Toolbox software. Robust Control Toolbox software extends the capabilities of Control System Toolbox software and leverages the LTI and plotting capabilities of Control System Toolbox software. The major analysis and synthesis commands in Robust Control Toolbox software accept LTI object inputs, e.g., LTI state-space systems produced by commands such as:

```
G=tf(1,[1 2 3])
G=ss([-1 0; 0 -1], [1;1],[1 1],3)
```

The uncertain system (`USS`) objects in Robust Control Toolbox software generalize the Control System Toolbox LTI `SS` objects and help ease the task of analyzing and plotting uncertain systems. You can do many of the same algebraic operations on uncertain systems that are possible for LTI objects (multiply, add, invert), and Robust Control Toolbox software provides `USS` uncertain system extensions of Control System Toolbox software interconnection and plotting functions like `feedback`, `lft`, and `bode`.

About the Authors

Professor **Andy Packard** is with the Faculty of Mechanical Engineering at the University of California, Berkeley. His research interests include robustness issues in control analysis and design, linear algebra and numerical algorithms in control problems, applications of system theory to aerospace problems, flight control, and control of fluid flow.

Professor **Gary Balas** is with the Faculty of Aerospace Engineering & Mechanics at the University of Minnesota and is president of MUSYN Inc. His research interests include aerospace control systems, both experimental and theoretical.

Dr. **Michael Safonov** is with the Faculty of Electrical Engineering at the University of Southern California. His research interests include control and decision theory.

Dr. **Richard Chiang** is employed by Boeing Satellite Systems, El Segundo, CA. He is a Boeing Technical Fellow and has been working in the aerospace industry over 25 years. In his career, Richard has designed 3 flight control laws, 12 spacecraft attitude control laws, and 3 large space structure vibration controllers, using modern robust control theory and the tools he built in this toolbox. His research interests include robust control theory, model reduction, and in-flight system identification. Working in industry instead of academia, Richard serves a unique role in our team, bridging the gap between theory and reality.

The linear matrix inequality (LMI) portion of Robust Control Toolbox software was developed by these two authors:

Dr. **Pascal Gahinet** is employed by MathWorks. His research interests include robust control theory, linear matrix inequalities, numerical linear algebra, and numerical software for control.

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

The structured H_∞ synthesis (`hinfstruct`) portion of Robust Control Toolbox software was developed by the following author in collaboration with Pascal Gahinet:

Professor **Pierre Apkarian** is with ONERA (The French Aerospace Lab) and the Institut de Mathématiques at Paul Sabatier University, Toulouse, France. His research interests include robust control, LMIs, mathematical programming, and nonsmooth optimization techniques for control.

Bibliography

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

[2] Dorato, P. (editor), *Robust Control*, New York, IEEE Press, 1987.

[3] Dorato, P., and Yedavalli, R.K. (editors), *Recent Advances in Robust Control*, New York, IEEE Press, 1990.

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

[5] El Ghaoui, L., and Niculescu, S., *Recent Advances in LMI Theory for Control*, Philadelphia, PA, SIAM, 2000.

[6] Lehtomaki, N.A., Sandell, Jr., N.R., and Athans, M., "Robustness Results in Linear-Quadratic Gaussian Based Multivariable Control Designs," *IEEE Trans. on Automat. Contr.*, Vol. AC-26, No. 1, Feb. 1981, pp. 75-92.

[7] Safonov, M.G., *Stability and Robustness of Multivariable Feedback Systems*, Cambridge, MA, MIT Press, 1980.

[8] Safonov, M.G., Laub, A.J., and Hartmann, G., "Feedback Properties of Multivariable Systems: The Role and Use of Return Difference Matrix," *IEEE Trans. of Automat. Contr.*, 1981, AC-26(1), pp. 47-65.

[9] Safonov, M.G., Chiang, R.Y., and Flashner, H., "H_∞ Control Synthesis for a Large Space Structure," *Proc. of American Contr. Conf.*, Atlanta, GA, June 15-17, 1988.

[10] Safonov, M.G., and Chiang, R.Y., "CACSD Using the State-Space L_∞ Theory — A Design Example," *IEEE Trans. on Automatic Control*, 1988, AC-33(5), pp. 477-479.

[11] Sanchez-Pena, R.S., and Sznaier, M., *Robust Systems Theory and Applications*, New York, Wiley, 1998.

[12] Skogestad, S., and Postlethwaite, I., *Multivariable Feedback Control*, New York, Wiley, 1996.

[13] Wie, B., and Bernstein, D.S., "A Benchmark Problem for Robust Controller Design," *Proc. American Control Conf.*, San Diego, CA, May 23-25, 1990; also Boston, MA, June 26-28, 1991.

[14] Zhou, K., Doyle, J.C., and Glover, K., *Robust and Optimal Control*, Englewood Cliffs, NJ, Prentice Hall, 1996.

2

Multivariable Loop Shaping

Tradeoff Between Performance and Robustness

When the plant modeling uncertainty is not too big, you can design high-gain, high-performance feedback controllers. High loop gains significantly larger than 1 in magnitude can attenuate the effects of plant model uncertainty and reduce the overall sensitivity of the system to plant noise. But if your plant model uncertainty is so large that you do not even know the sign of your plant gain, then you cannot use large feedback gains without the risk that the system will become unstable. Thus, plant model uncertainty can be a fundamental limiting factor in determining what can be achieved with feedback.

Multiplicative Uncertainty: Given an approximate model of the plant G_0 of a plant G, the *multiplicative uncertainty* Δ_M of the model G_0 is defined

as $\Delta_M = G_0^{-1}(G - G_0)$

or, equivalently,

$$G = (I + \Delta_M)G_0.$$

Plant model uncertainty arises from many sources. There might be small unmodeled time delays or stray electrical capacitance. Imprecisely understood actuator time constants or, in mechanical systems, high-frequency torsional bending modes and similar effects can be responsible for plant model uncertainty. These types of uncertainty are relatively small at lower frequencies and typically increase at higher frequencies.

In the case of single-input/single-output (SISO) plants, the frequency at which there are uncertain variations in your plant of size $|\Delta_M|=2$ marks a critical threshold beyond which there is insufficient information about the plant to reliably design a feedback controller. With such a 200% model uncertainty, the model provides no indication of the phase angle of the true plant, which means that the only way you can reliably stabilize your plant is to ensure that the loop gain is less than 1. Allowing for an additional factor of 2 margin for error, your control system bandwidth is essentially limited

to the frequency range over which your multiplicative plant uncertainty Δ_M has gain magnitude $|\Delta_M|<1$.

Norms and Singular Values

For MIMO systems the transfer functions are matrices, and relevant measures of gain are determined by singular values, H_∞, and H_2 norms, which are defined as follows:

H_2 and H_∞ Norms The H_2-norm is the energy of the impulse response of plant `G`. The H_∞-norm is the peak gain of `G` across all frequencies and all input directions.

Another important concept is the notion of singular values.

Singular Values: The *singular values* of a rank r matrix $A \in C^{m \times n}$, denoted σ_i, are the nonnegative square roots of the eigenvalues of $A^* A$ ordered such that $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_p > 0$, $p \leq \min\{m, n\}$.

If $r < p$ then there are $p - r$ zero singular values, i.e., $\sigma_{r+1} = \sigma r+2 = \ldots = \sigma_p = 0$.

The greatest singular value σ_1 is sometimes denoted

$$\bar{\sigma}(A) = \sigma_1.$$

When A is a square n-by-n matrix, then the nth singular value (i.e., the least singular value) is denoted

$$\bar{\sigma}(A) = \sigma_n.$$

Properties of Singular Values

Some useful properties of singular values are:

$$\bar{\sigma}(A) = \max_{x \in C^h} \frac{\|Ax\|}{\|x\|}$$

$$\underline{\sigma}(A) = \min_{x \in C^h} \frac{\|Ax\|}{\|x\|}$$

These properties are especially important because they establish that the greatest and least singular values of a matrix A are the maximal and minimal "gains" of the matrix as the input vector x varies over all possible directions.

For stable continuous-time LTI systems $G(s)$, the H_2-norm and the H_∞-norms are defined terms of the frequency-dependent singular values of $G(j\omega)$:

H_2-norm:

$$\|G\|_2 \square \left[\frac{1}{2\pi}\right] \int_{-\infty}^{\infty} \sum_{i=1}^{p} \left(\sigma_i\left(G(j\omega)\right)\right)^2 d\omega$$

H_∞-norm:

$$\|G\|_2 \square \sup_{\omega} \bar{\sigma}\left(G(j\omega)\right)$$

where sup denotes the least upper bound.

Typical Loop Shapes, S and T Design

Consider the multivariable feedback control system shown in the following figure. In order to quantify the multivariable stability margins and performance of such systems, you can use the singular values of the closed-loop transfer function matrices from r to each of the three outputs e, u, and y, *viz.*

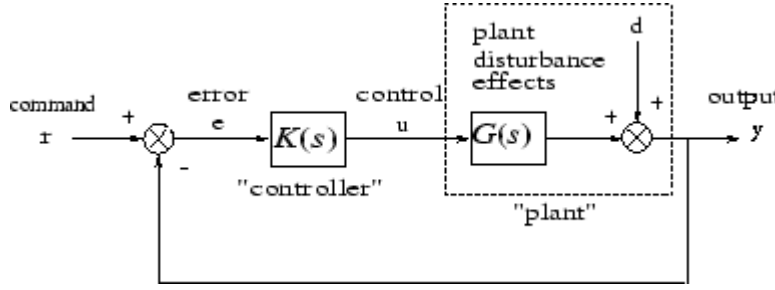
$$S(s) \stackrel{def}{=} \left(I + L(s)\right)^{-1}$$

$$R(s) \stackrel{def}{=} K(s)\left(I + L(s)\right)^{-1}$$

$$T(s) \stackrel{def}{=} L(s)\left(I + L(s)\right)^{-1} = I - S(s)$$

where the $L(s)$ is the loop transfer function matrix

$$L(s) = G(s)K(s). \qquad \textbf{(2-1)}$$



Block Diagram of the Multivariable Feedback Control System

The two matrices $S(s)$ and $T(s)$ are known as the *sensitivity function* and *complementary sensitivity function*, respectively. The matrix $R(s)$ has no common name. The singular value Bode plots of each of the three transfer function matrices $S(s)$, $R(s)$, and $T(s)$ play an important role in robust multivariable control system design. The singular values of the loop transfer function matrix $L(s)$ are important because $L(s)$ determines the matrices $S(s)$ and $T(s)$.

Singular Values

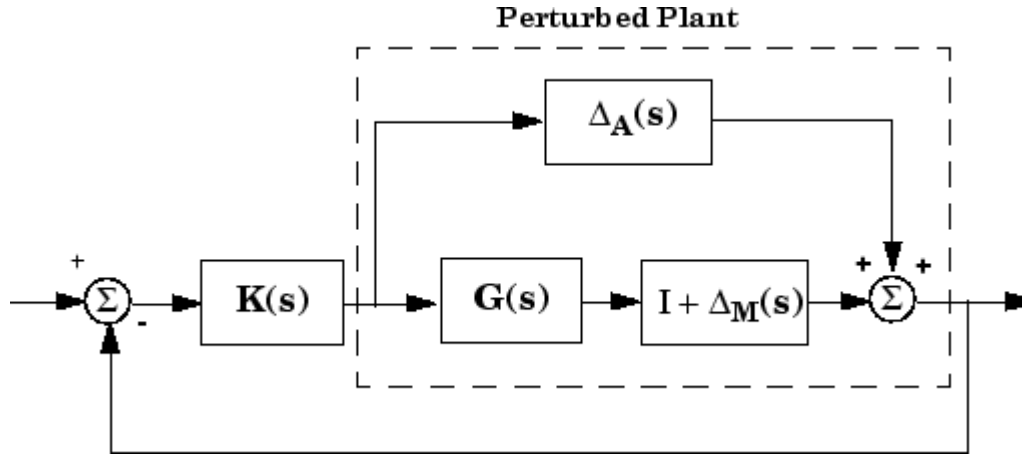
The singular values of $S(j\omega)$ determine the disturbance attenuation, because $S(s)$ is in fact the closed-loop transfer function from disturbance d to plant output y — see Block Diagram of the Multivariable Feedback Control System on page 2-5. Thus a disturbance attenuation performance specification can be written as

$$\bar{\sigma}\left(S(j\omega)\right) \le \left|W_1^{-1}(j\omega)\right| \qquad \textbf{(2-2)}$$

where $\left|W_1^{-1}(j\omega)\right|$ is the desired disturbance attenuation factor. Allowing $\left|W_1(j\omega)\right|$ to depend on frequency ω enables you to specify a different attenuation factor for each frequency ω.

The singular value Bode plots of $R(s)$ and of $T(s)$ are used to measure the stability margins of multivariable feedback designs in the face of additive plant perturbations Δ_A and multiplicative plant perturbations Δ_M, respectively. See the following figure.

Consider how the singular value Bode plot of complementary sensitivity $T(s)$ determines the stability margin for multiplicative perturbations Δ_M. The multiplicative stability margin is, by definition, the "size" of the smallest stable $\Delta_M(s)$ that destabilizes the system in the figure below when $\Delta_A = 0$.



Additive/Multiplicative Uncertainty

Taking $\bar{\sigma}(\Delta_M(j\omega))$ to be the definition of the "size" of $\Delta_M(j\omega)$, you have the following useful characterization of "multiplicative" stability robustness:

Multiplicative Robustness: The size of the smallest destabilizing multiplicative uncertainty $\Delta_M(s)$ is:

$$\bar{\sigma}(\Delta_M(j\omega)) = \frac{1}{\bar{\sigma}(T(j\omega))}.$$

The smaller is $\bar{\sigma}(T(j\omega))$, the greater will be the size of the smallest destabilizing multiplicative perturbation, and hence the greater will be the stability margin.

A similar result is available for relating the stability margin in the face of additive plant perturbations $\Delta_A(s)$ to $R(s)$ if you take $\bar{\sigma}(\Delta_A(j\omega))$ to be the definition of the "size" of $\Delta_A(j\omega)$ at frequency ω.

Additive Robustness: The size of the smallest destabilizing additive uncertainty Δ_A is:

$$\bar{\sigma}\left(\Delta_A(j\omega)\right) = \frac{1}{\bar{\sigma}\left(R(j\omega)\right)}.$$

As a consequence of robustness theorems 1 and 2, it is common to specify the stability margins of control systems via singular value inequalities such as

$$\bar{\sigma}\left(R\{j\omega\}\right) \le \left|W_2^{-1}(j\omega)\right| \tag{2-3}$$

$$\bar{\sigma}\left(T\{j\omega\}\right) \le \left|W_3^{-1}(j\omega)\right| \tag{2-4}$$

where $|W_2(j\omega)|$ and $|W_3(j\omega)|$ are the respective sizes of the largest anticipated additive and multiplicative plant perturbations.

It is common practice to lump the effects of all plant uncertainty into a single fictitious multiplicative perturbation Δ_M, so that the control design requirements can be written

$$\frac{1}{\sigma_i\left(S(j\omega)\right)} \ge \left|W_1(j\omega)\right|; \quad \bar{\sigma}_i\left(T[j\omega]\right) \le \left|W_3^{-1}(j\omega)\right|$$

as shown in Singular Value Specifications on L, S, and T on page 2-11.

It is interesting to note that in the upper half of the figure (above the 0 dB line),

$$\underline{\sigma}\left(L(j\omega)\right) \approx \frac{1}{\bar{\sigma}\left(S(j\omega)\right)}$$

while in the lower half of Singular Value Specifications on L, S, and T on page 2-11 (below the 0 dB line),

$$\underline{\sigma}\left(L(j\omega)\right) \approx \bar{\sigma}\left(T(j\omega)\right).$$

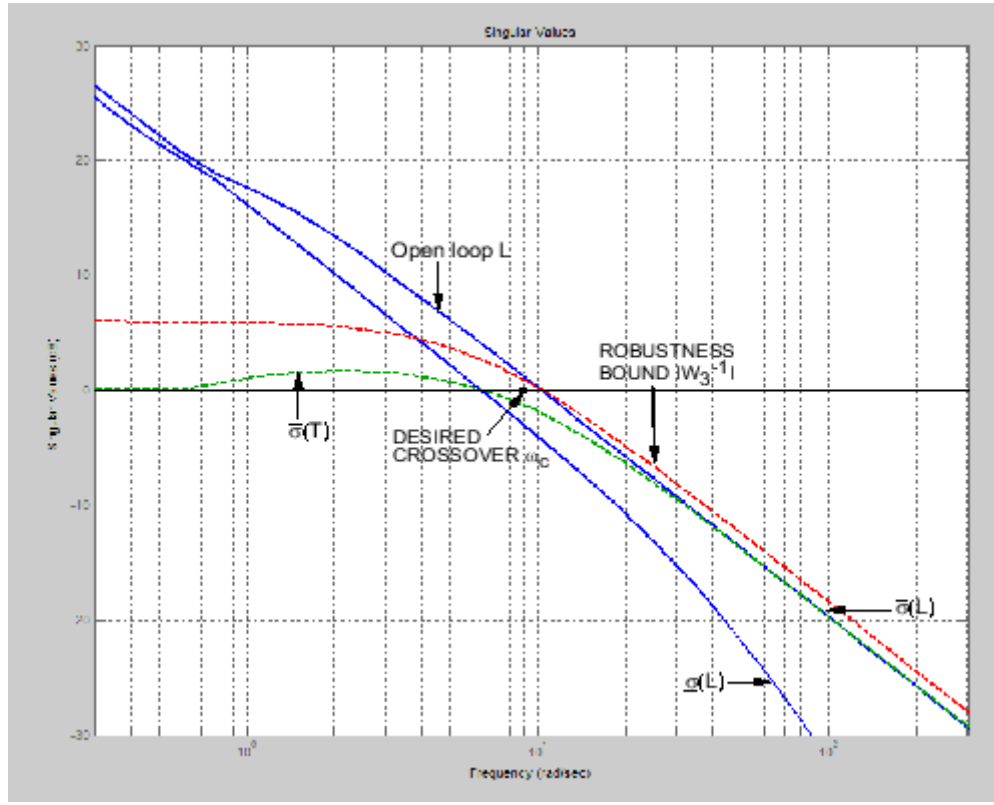
This results from the fact that

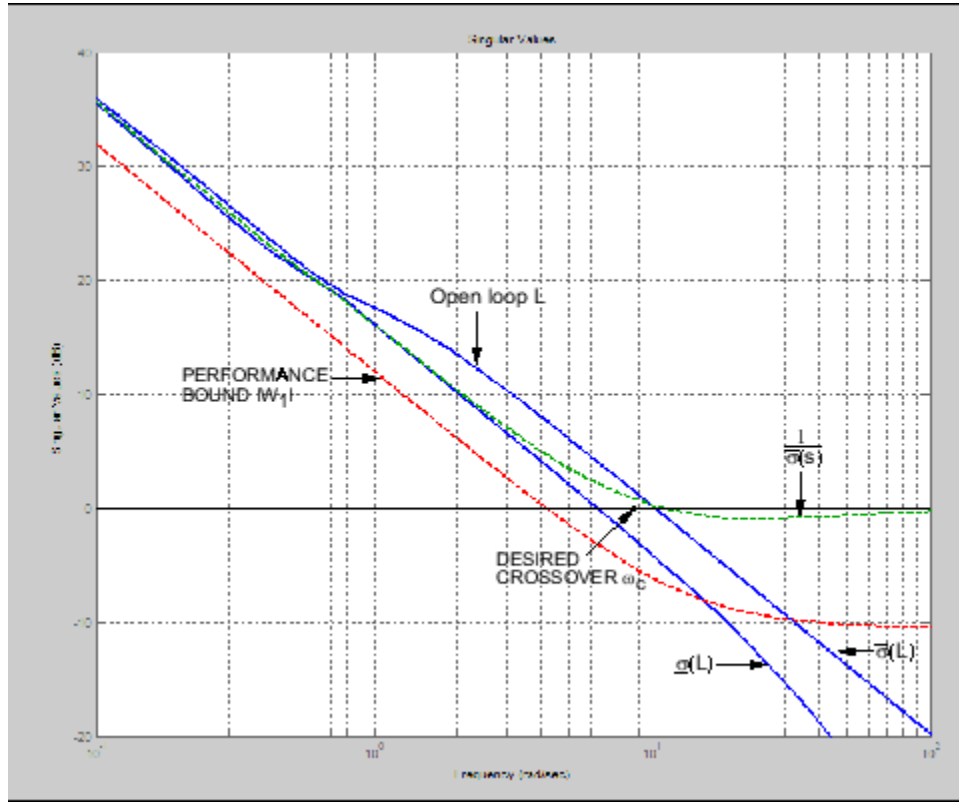
$$S(s) \stackrel{def}{=} \left(I + L(s)\right)^{-1} \approx L(s)^{-1}$$

if $\underline{\sigma}\left(L(s)\right) \gg 1$, and

$$T(s) \stackrel{def}{=} L(s)\left(I + L(s)\right)^{-1} \approx L(s)$$

if $\bar{\sigma}\left(L(s)\right) \ll 1$.


Singular Values
Open loop L
ROBUSTNESS
BOUND $|W_3^{-1}|$
$\bar{\sigma}(T)$
DESIRED
CROSSOVER ω_c
$\bar{\sigma}(L)$
$\underline{\sigma}(L)$
Frequency (rad/sec)



Singular Value Specifications on L, S, and T

Thus, it is not uncommon to see specifications on disturbance attenuation and multiplicative stability margin expressed directly in terms of forbidden regions for the Bode plots of $\sigma_i(L(j\omega))$ as "singular value loop shaping" requirements, either as specified upper/lower bounds or as a target desired loop shape — see the preceding figure.

Guaranteed Gain/Phase Margins in MIMO Systems

For those who are more comfortable with classical single-loop concepts, there are the important connections between the multiplicative stability margins predicted by $\bar{\sigma}(T)$ and those predicted by classical *M*-circles, as found on the Nichols chart. Indeed in the single-input/single-output case,

$$\bar{\sigma}\left(T(j\omega)\right) = \left|\frac{L(j\omega)}{1+L(j\omega)}\right|$$

which is precisely the quantity you obtain from Nichols chart *M*-circles. Thus, $\|T\|_\infty$ is a multiloop generalization of the closed-loop resonant peak magnitude which, as classical control experts will recognize, is closely related to the damping ratio of the dominant closed-loop poles. Also, it turns out that you can relate $\|T\|_\infty$, $\|S\|_\infty$ to the classical gain margin G_M and phase margin θ_M in each feedback loop of the multivariable feedback system of Block Diagram of the Multivariable Feedback Control System on page 2-5 via the formulas:

$$G_M \geq 1 + \frac{1}{\|T\|_\infty}$$

$$G_M \geq 1 + \frac{1}{1 - \frac{1}{\|S\|_\infty}}$$

$$\theta_M \geq 2\sin^{-1}\left(\frac{1}{2\|T\|_\infty}\right)$$

$$\theta_M \geq 2\sin^{-1}\left(\frac{1}{2\|T\|_\infty}\right).$$

(See [6].) These formulas are valid provided $\|S\|_\infty$ and $\|T\|_\infty$ are larger than 1, as is normally the case. The margins apply even when the gain perturbations or phase perturbations occur simultaneously in several feedback channels.

The infinity norms of S and T also yield gain reduction tolerances. The *gain reduction tolerance* g_m is defined to be the minimal amount by which the gains in each loop would have to be *decreased* in order to destabilize the system. Upper bounds on g_m are as follows:

$$g_M \le 1 - \frac{1}{\|T\|_\infty}$$

$$g_M \le \frac{1}{1 + \frac{1}{\|S\|_\infty}}.$$

Using LOOPSYN to Do H-Infinity Loop Shaping

The command `loopsyn` lets you design a stabilizing feedback controller to optimally shape the open loop frequency response of a MIMO feedback control system to match as closely as possible a desired loop shape Gd — see the preceding figure. The basic syntax of the `loopsyn` loop-shaping controller synthesis command is:

```
K = loopsyn(G,Gd)
```

Here `G` is the LTI transfer function matrix of a MIMO plant model, `Gd` is the target desired loop shape for the loop transfer function `L=G*K`, and `K` is the optimal loop-shaping controller. The LTI controller `K` has the property that it shapes the loop `L=G*K` so that it matches the frequency response of `Gd` as closely as possible, subject to the constraint that the compensator must stabilize the plant model `G`.

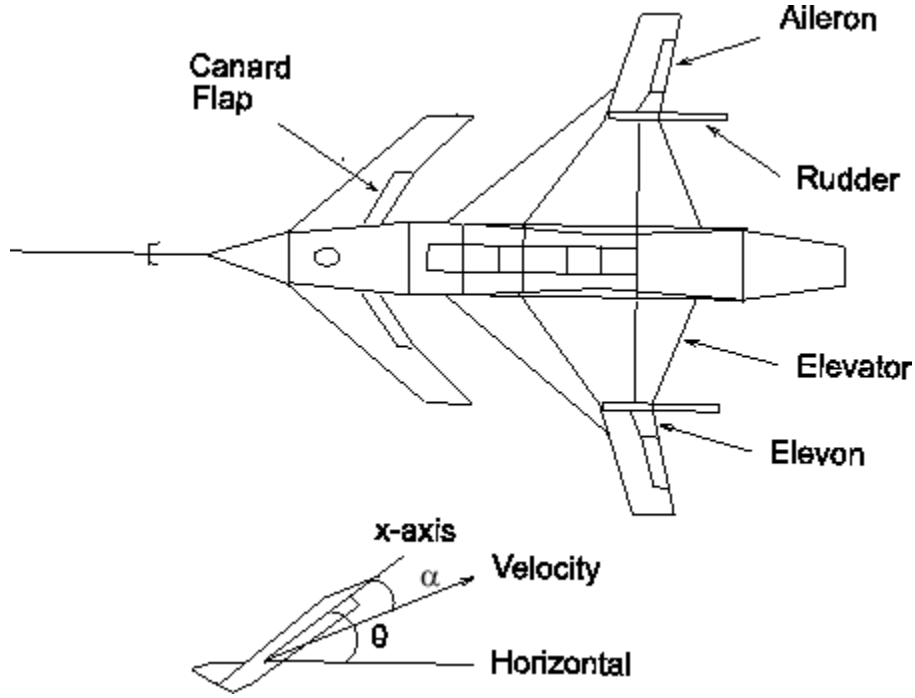
Example: NASA HiMAT Loop Shaping

To see how the `loopsyn` command works in practice to address robustness and performance tradeoffs, consider again the NASA HiMAT aircraft model taken from the paper of Safonov, Laub, and Hartmann [8]. The longitudinal dynamics of the HiMAT aircraft trimmed at 25000 ft and 0.9 Mach are unstable and have two right-half-plane phugoid modes. The linear model has state-space realization $G(s) = C(Is - A)^{-1}B$ with six states, with the first four states representing angle of attack (α) and attitude angle (θ) and their rates of change, and the last two representing elevon and canard control actuator dynamics — see Aircraft Configuration and Vertical Plane Geometry on page 2-15.

```
ag =[
-2.2567e-02  -3.6617e+01  -1.8897e+01  -3.2090e+01   3.2509e+00  -7.6257e-01;
9.2572e-05  -1.8997e+00   9.8312e-01  -7.2562e-04  -1.7080e-01  -4.9652e-03;
1.2338e-02   1.1720e+01  -2.6316e+00   8.7582e-04  -3.1604e+01   2.2396e+01;
0            0   1.0000e+00            0            0            0;
0            0            0            0  -3.0000e+01            0;
0            0            0            0            0  -3.0000e+01];
bg = [0     0;
      0     0;
      0     0;
```

```
       0     0;
      30     0;
       0    30];
cg = [0     1     0     0     0     0;
      0     0     0     1     0     0];
dg = [0     0;
      0     0];
G=ss(ag,bg,cg,dg);
```

The control variables are elevon and canard actuators (δ_e and δ_c). The output variables are angle of attack (α) and attitude angle (θ).



Aircraft Configuration and Vertical Plane Geometry

This model is good at frequencies below 100 rad/s with less than 30% variation between the true aircraft and the model in this frequency range. However as noted in [8], it does not reliably capture very high-frequency behaviors, because it was derived by treating the aircraft as a rigid body and

neglecting lightly damped fuselage bending modes that occur at somewhere between 100 and 300 rad/s. These unmodeled bending modes might cause as much as 20 dB deviation (i.e., 1000%) between the frequency response of the model and the actual aircraft for frequency $\omega > 100$ rad/s. Other effects like control actuator time delays and fuel sloshing also contribute to model inaccuracy at even higher frequencies, but the dominant unmodeled effects are the fuselage bending modes. You can think of these unmodeled bending modes as multiplicative uncertainty of size 20 dB, and design your controller using loopsyn, by making sure that the loop has gain less than –20 dB at, and beyond, the frequency $\omega > 100$ rad/s.

Design Specifications

The singular value design specifications are

- **Robustness Spec.**: –20 dB/decade roll-off slope and –20 dB loop gain at 100 rad/s
- **Performance Spec.**: Minimize the sensitivity function as much as possible.

Both specs can be accommodated by taking as the desired loop shape

$$G_d(s)=8/s$$

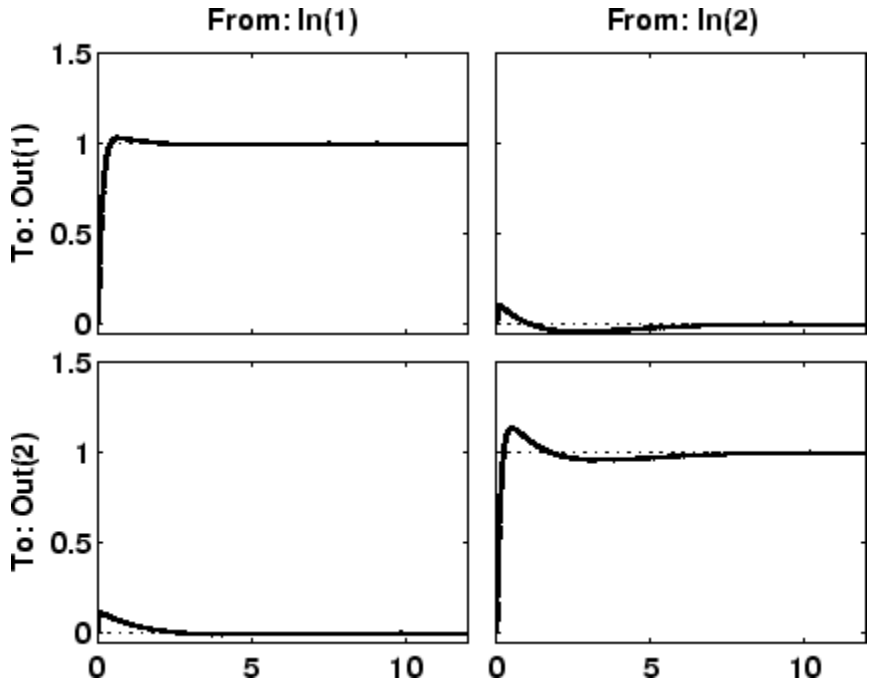
MATLAB Commands for a LOOPSYN Design

```
%% Enter the desired loop shape Gd
s=zpk('s'); % Laplace variable s
Gd=8/s; % desired loop shape
%% Compute the optimal loop shaping controller K
[K,CL,GAM]=loopsyn(G,Gd);
%% Compute the loop L, sensitivity S and
%% complementary sensitivity T:
L=G*K;
I=eye(size(L));
S=feedback(I,L); % S=inv(I+L);
T=I-S;
%% Plot the results:
% step response plots
```

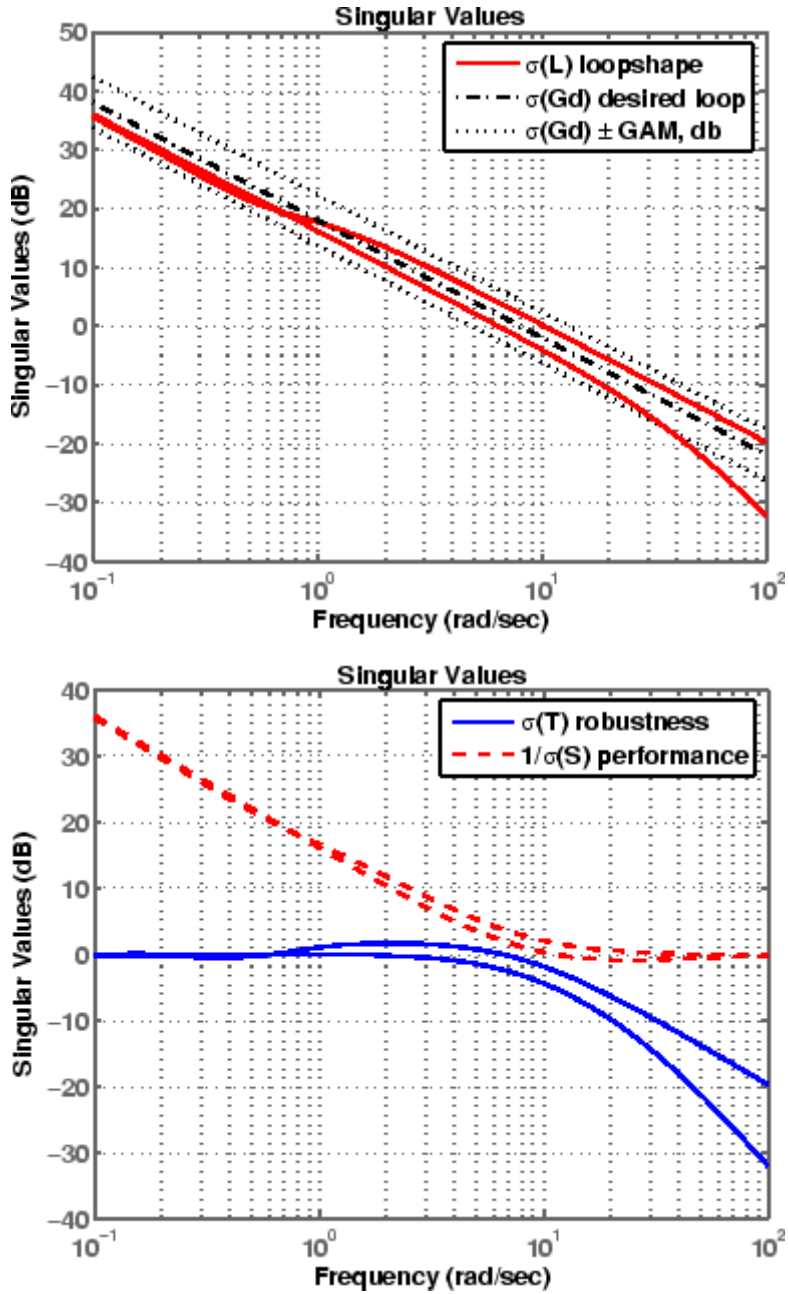
```
step(T);title('\alpha and \theta command step responses');
% frequency response plots
figure;
sigma(I+L,'--',T,':',L,'r--',Gd,'k-.',Gd/GAM,'k:',...
 Gd*GAM,'k:',{.1,100});grid
legend('1/\sigma(S) performance',...
 '\sigma(T) robustness',...
 '\sigma(L) loopshape',...
 '\sigma(Gd) desired loop',...
 '\sigma(Gd) \pm GAM, dB');
```

The plots of the resulting step and frequency response for the NASA HiMAT `loopsyn` loop-shaping controller design are shown in the following figure. The number ±GAM, dB (i.e., 20log10(GAM)) tells you the accuracy with which your `loopsyn` control design matches the target desired loop:

$$\bar{\sigma}(GK),\mathrm{db} \geq |\mathrm{G_d}|,\mathrm{db} - \mathrm{GAM},\mathrm{db} \quad (\omega < \omega_c)$$
$$\bar{\sigma}(GK),\mathrm{db} \geq |\mathrm{G_d}|,\mathrm{db} + \mathrm{GAM},\mathrm{db} \quad (\omega > \omega_c).$$



HiMAT Closed Loop Step Responses



LOOPSYN Design Results for NASA HiMAT

Fine-Tuning the LOOPSYN Target Loop Shape Gd to Meet Design Goals

If your first attempt at `loopsyn` design does not achieve everything you wanted, you will need to readjust your target desired loop shape `Gd`. Here are some basic design tradeoffs to consider:

- **Stability Robustness.** Your target loop `Gd` should have low gain (as small as possible) at high frequencies where typically your plant model is so poor that its phase angle is completely inaccurate, with errors approaching ±180° or more.
- **Performance.** Your `Gd` loop should have high gain (as great as possible) at frequencies where your model is good, in order to ensure good control accuracy and good disturbance attenuation.
- **Crossover and Roll-Off.** Your desired loop shape `Gd` should have its 0 dB crossover frequency (denoted ω_c) between the above two frequency ranges, and below the crossover frequency ω_c it should roll off with a negative slope of between –20 and –40 dB/decade, which helps to keep phase lag to less than –180° inside the control loop bandwidth ($0 < \omega < \omega_c$).

Other considerations that might affect your choice of `Gd` are the right-half-plane poles and zeros of the plant `G`, which impose ffundamental limits on your 0 dB crossover frequency ω_c [12]. For instance, your 0 dB crossover ω_c must be greater than the magnitude of any plant right-half-plane poles and less than the magnitude of any right-half-plane zeros.

$$\max_{\mathrm{Re}(p_i)>0} |p_i| < \omega_c < \min_{\mathrm{Re}(z_i)>0} |z_i|.$$

If you do not take care to choose a target loop shape `Gd` that conforms to these fundamental constraints, then `loopsyn` will still compute the optimal loop-shaping controller `K` for your `Gd`, but you should expect that the optimal loop `L=G*K` will have a poor fit to the target loop shape `Gd`, and consequently it might be impossible to meet your performance goals.

Using MIXSYN for H-Infinity Loop Shaping

A popular alternative approach to `loopsyn` loop shaping is H_∞ *mixed-sensitivity* loop shaping, which is implemented by the Robust Control Toolbox software command:

```
K=mixsyn(G,W1,[],W3)
```

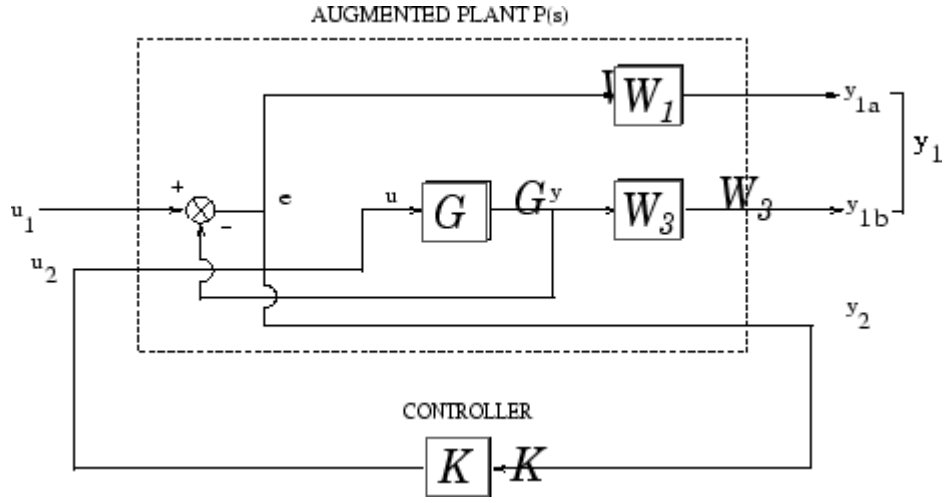
With `mixsyn` controller synthesis, your performance and stability robustness specifications equations (2-2) and (2-4) are combined into a single infinity norm specification of the form

$$\left\| T_{y_1 u_1} \right\|_\infty \le 1$$

where (see MIXSYN H_∞ Mixed-Sensitivity Loop Shaping $T_{y1\ u1}$ on page 2-22):

$$T_{y_1 u_1} \overset{def}{=} \begin{bmatrix} W_1 S \\ W_3 T \end{bmatrix}.$$

The term $\left\| T_{y_1 u_1} \right\|_\infty$ is called a *mixed-sensitivity cost function* because it penalizes both sensitivity $S(s)$ and complementary sensitivity $T(s)$. Loop shaping is achieved when you choose W_1 to have the target loop shape for frequencies $\omega < \omega_c$, and you choose $1/W_3$ to be the target for $\omega > \omega_c$. In choosing design specifications W_1 and W_3 for a `mixsyn` controller design, you need to ensure that your 0 dB crossover frequency for the Bode plot of W_1 is below the 0 dB crossover frequency of $1/W_3$, as shown in Singular Value Specifications on L, S, and T on page 2-11, so that there is a gap for the desired loop shape `Gd` to pass between the performance bound W_1 and your robustness bound W_3^{-1}. Otherwise, your performance and robustness requirements will not be achievable.



MIXSYN H_∞ Mixed-Sensitivity Loop Shaping $T_{y1\ u1}$

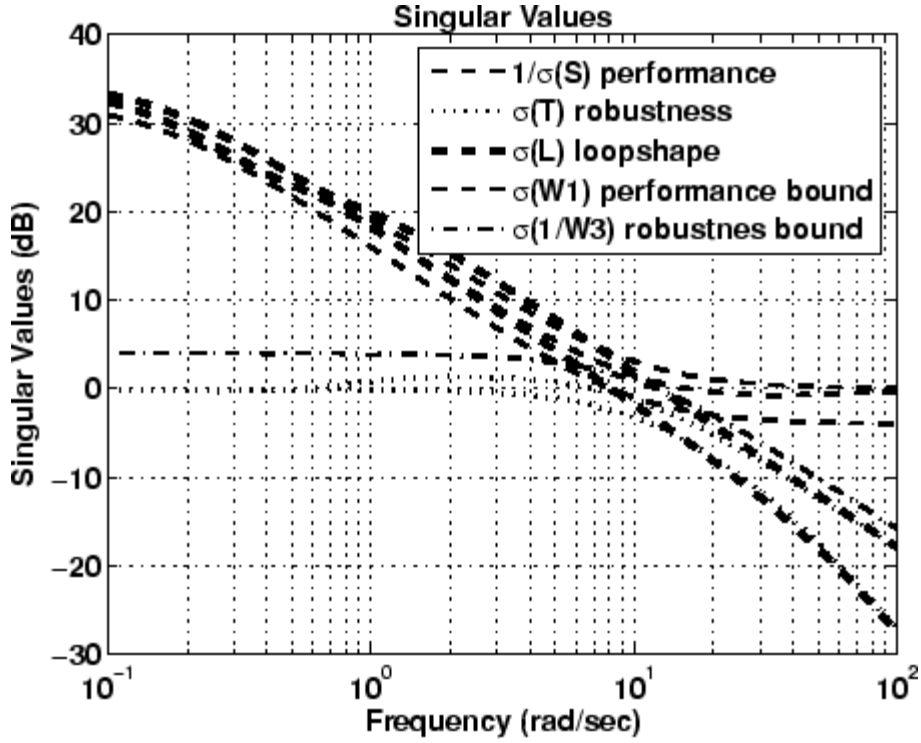
Example: NASA HiMAT Design Using MIXSYN

To do a `mixsyn` H_∞ mixed-sensitivity synthesis design on the HiMAT model, start with the plant model `G` created in "Example: NASA HiMAT Design Using MIXSYN" on page 2-22 and type the following commands:

```
% Set up the performance and robustness bounds W1 & W3
s=zpk('s'); % Laplace variable s
MS=2;AS=.03;WS=5;
W1=(s/MS+WS)/(s+AS*WS);
MT=2;AT=.05;WT=20;
W3=(s+WT/MT)/(AT*s+WT);
% Compute the H-infinity mixed-sensitivity optimal sontroller K1
[K1,CL1,GAM1]=mixsyn(G,W1,[],W3);
% Next compute and plot the closed-loop system.
% Compute the loop L1, sensitivity S1, and comp sensitivity T1:
L1=G*K1;
I=eye(size(L1));
S1=feedback(I,L1); % S=inv(I+L1);
T1=I-S1;
% Plot the results:
% step response plots
```

```
step(T1,1.5);
title('\alpha and \theta command step responses');
% frequency response plots
figure;
sigma(I+L1,'--',T1,':',L1,'r--',... W1/GAM1,'k--',GAM1/W3,'k-.',{.1,100});grid
legend('1/\sigma(S) performance',...
'\sigma(T) robustness',...
'\sigma(L) loopshape',...
'\sigma(W1) performance bound',...
'\sigma(1/W3) robustness bound');
```

The resulting `mixsyn` singular value plots for the NASA HiMAT model are shown below.



MIXSYN Design Results for NASA HiMAT

Loop-Shaping Commands

Robust Control Toolbox software gives you several choices for shaping the frequency response properties of multiinput/multioutput (MIMO) feedback control loops. Some of the main commands that you are likely to use for loop-shaping design, and associated utility functions, are listed below:

MIMO Loop-Shaping Commands	
`loopsyn`	H_∞ loop-shaping controller synthesis
`ltrsyn`	LQG loop-transfer recovery
`mixsyn`	H_∞ mixed-sensitivity controller synthesis
`ncfsyn`	Glover-McFarlane H_∞ normalized coprime factor loop-shaping controller synthesis

MIMO Loop-Shaping Utility Functions	
`augw`	Augmented plant for weighted H_2 and H_∞ mixed-sensitivity control synthesis
`makeweight`	Weights for H_∞ mixed sensitivity (`mixsyn`, `augw`)
`sigma`	Singular value plots of LTI feedback loops

3

Model Reduction for Robust Control

Introduction

In the design of robust controllers for complicated systems, model reduction fits several goals:

1. To simplify the best available model in light of the purpose for which the model is to be used—namely, to design a control system to meet certain specifications.

2. To speed up the simulation process in the design validation stage, using a smaller size model with most of the important system dynamics preserved.

3. Finally, if a modern control method such as LQG or H_∞ is used for which the complexity of the control law is not explicitly constrained, the order of the resultant controller is likely to be considerably greater than is truly needed. A good model reduction algorithm applied to the control law can sometimes significantly reduce control law complexity with little change in control system performance.

Model reduction routines in this toolbox can be put into two categories:

- **Additive error method** — The reduced-order model has an additive error bounded by an error criterion.
- **Multiplicative error method** — The reduced-order model has a multiplicative or relative error bounded by an error criterion.

The error is measured in terms of peak gain across frequency (H_∞ norm), and the error bounds are a function of the neglected Hankel singular values.

Hankel Singular Values

In control theory, eigenvalues define a system stability, whereas *Hankel singular values* define the "energy" of each state in the system. Keeping larger energy states of a system preserves most of its characteristics in terms of stability, frequency, and time responses. Model reduction techniques presented here are all based on the Hankel singular values of a system. They can achieve a reduced-order model that preserves the majority of the system characteristics.

Mathematically, given a *stable* state-space system (*A,B,C,D*), its Hankel singular values are defined as [1]

$$\sigma_H = \sqrt{\lambda_i(PQ)}$$

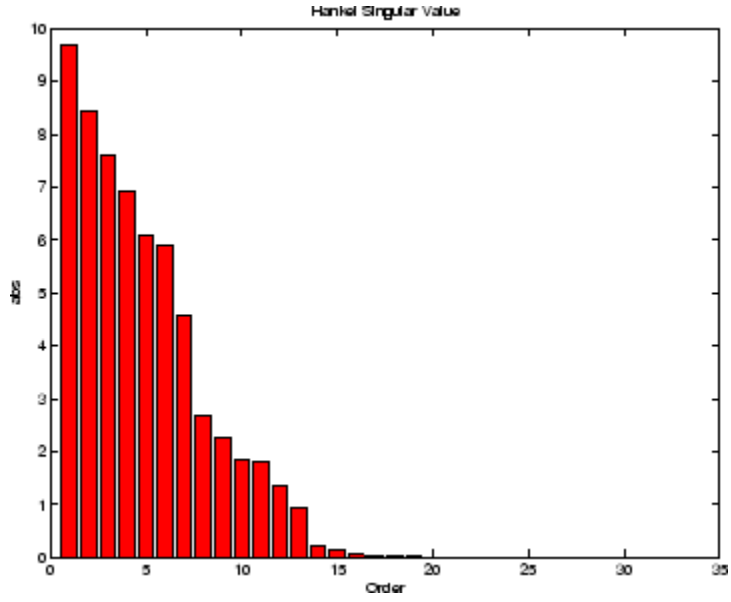
where P and Q are *controllability* and *observability grammians* satisfying

$$AP + PA^T = -BB^T$$
$$A^TQ + QA = -C^TC.$$

For example,

```
rand('state',1234); randn('state',5678);
G = rss(30,4,3);
hankelsv(G)
```

returns a Hankel singular value plot as follows:



which shows that system G has most of its "energy" stored in states 1 through 15 or so. Later, you will see how to use model reduction routines to keep a 15-state reduced model that preserves most of its dynamic response.

Overview of Model Reduction Techniques

Robust Control Toolbox software offers several algorithms for model approximation and order reduction. These algorithms let you control the absolute or relative approximation error, and are all based on the Hankel singular values of the system.

As discussed in previous sections, robust control theory quantifies a system uncertainty as either *additive* or *multiplicative* types. These model reduction routines are also categorized into two groups: *additive error* and *multiplicative error* types. In other words, some model reduction routines produce a reduced-order model `Gred` of the original model `G` with a bound on the error $\|G - Gred\|_\infty$, the peak gain across frequency. Others produce a reduced-order model with a bound on the relative error $\left\|G^{-1}(G - Gred)\right\|_\infty$.

These theoretical bounds are based on the "tails" of the Hankel singular values of the model, i.e.,

Additive Error Bound

$$\|G - Gred\|_\infty \le 2\sum_{k+1}^{n} \sigma_i \tag{3-1}$$

where σ_i are denoted the ith Hankel singular value of the original system `G`.

Multiplicative (Relative) Error Bound

$$\left\|G^{-1}(G - Gred)\right\|_\infty \le \prod_{k+1}^{n}\left(1 + 2\sigma_i\left(\sqrt{1+\sigma_i^2} + \sigma_i\right)\right) - 1 \tag{3-2}$$

where σ_i are denoted the ith Hankel singular value of the phase matrix of the model `G` (see the `bstmr` reference page).

Top-Level Model Reduction Command

Method	Description
`reduce`	Main interface to model approximation algorithms

Normalized Coprime Balanced Model Reduction Command

Method	Description
`ncfmr`	Normalized coprime balanced truncation

Additive Error Model Reduction Commands

Method	Description
`balancmr`	Square-root balanced model truncation
`schurmr`	Schur balanced model truncation
`hankelmr`	Hankel minimum degree approximation

Multiplicative Error Model Reduction Command

Method	Description
`bstmr`	Balanced stochastic truncation

Additional Model Reduction Tools

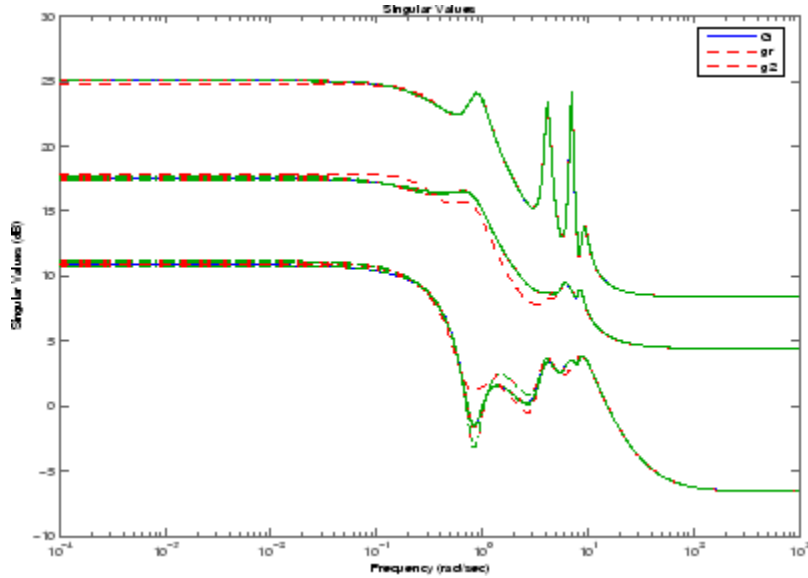
Method	Description
`modreal`	Modal realization and truncation
`slowfast`	Slow and fast state decomposition
`stabsep`	Stable and antistable state projection

Approximating Plant Models — Additive Error Methods

Given a system in LTI form, the following commands reduce the system to any desired order you specify. The judgment call is based on its Hankel singular values as shown in the previous paragraph.

```
rand('state',1234); randn('state',5678);
G = rss(30,4,3);
% balanced truncation to models with sizes 12:16
[g1,info1] = balancmr(G,12:16); % or use reduce
% Schur balanced truncation by specifying `MaxError'
[g2,info2] = schurmr(G,'MaxError',[1,0.8,0.5,0.2]);
sigma(G,'b-',g1,'r--',g2,'g-.')
```

shows a comparison plot of the original model `G` and reduced models `g1` and `g2`.

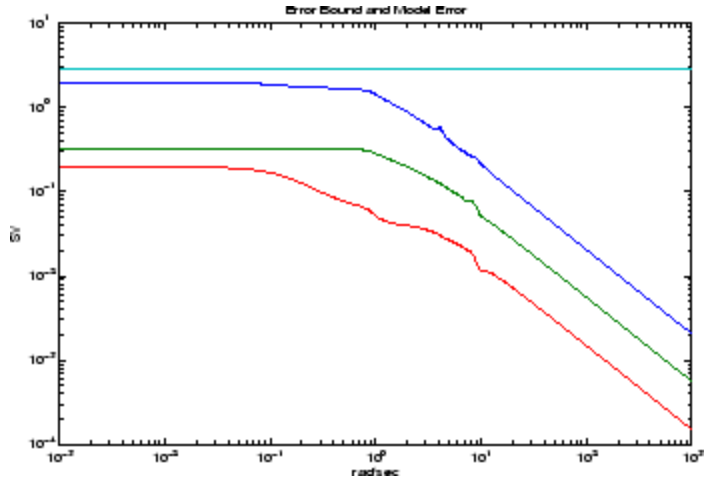


To determine whether the theoretical error bound is satisfied,

```
norm(G-g1(:,:,1),'inf') % 2.0123
info1.ErrorBound(1) % 2.8529
```

or plot the model error vs. error bound via the following commands:

```
[sv,w] = sigma(G-g1(:,:,1));
loglog(w,sv,w,info1.ErrorBound(1)*ones(size(w)))
xlabel('rad/sec');ylabel('SV');
title('Error Bound and Model Error')
```

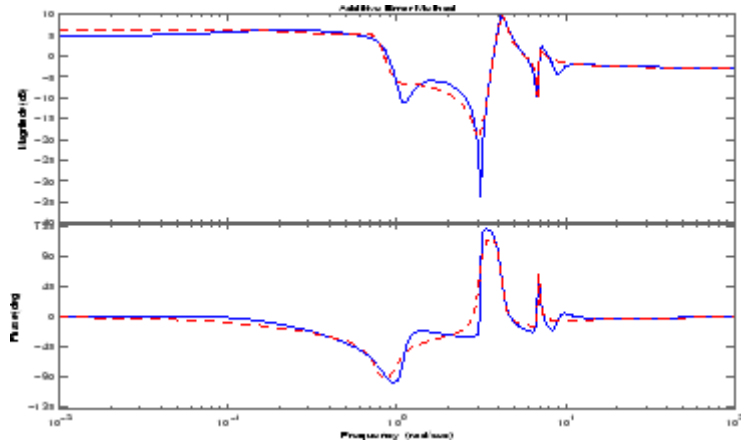


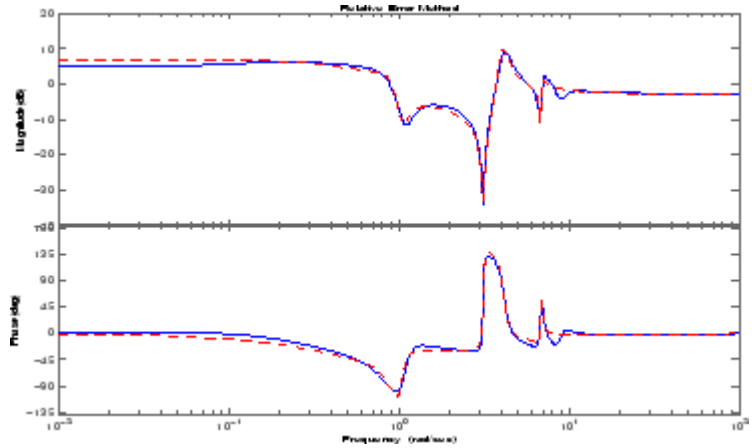
Approximating Plant Models — Multiplicative Error Method

In most cases, *multiplicative error* model reduction method `bstmr` tends to bound the relative error between the original and reduced-order models across the frequency range of interest, hence producing a more accurate reduced-order model than the *additive error* methods. This characteristic is obvious in system models with low damped poles.

The following commands illustrate the significance of a multiplicative error model reduction method as compared to any additive error type. Clearly, the phase-matching algorithm using `bstmr` provides a better fit in the Bode plot.

```
rand('state',1234); randn('state',5678); G = rss(30,1,1);
[gr,infor] = reduce(G,'algo','balance','order',7);
[gs,infos] = reduce(G,'algo','bst','order',7);
figure(1);bode(G,'b-',gr,'r--');
title('Additive Error Method')
figure(2);bode(G,'b-',gs,'r--');
title('Relative Error Method')
```





Therefore, for some systems with low damped poles/zeros, the balanced stochastic method (`bstmr`) produces a better reduced-order model fit in those frequency ranges to make multiplicative error small. Whereas additive error methods such as `balancmr`, `schurmr`, or `hankelmr` only care about minimizing the overall "absolute" peak error, they can produce a reduced-order model missing those low damped poles/zeros frequency regions.

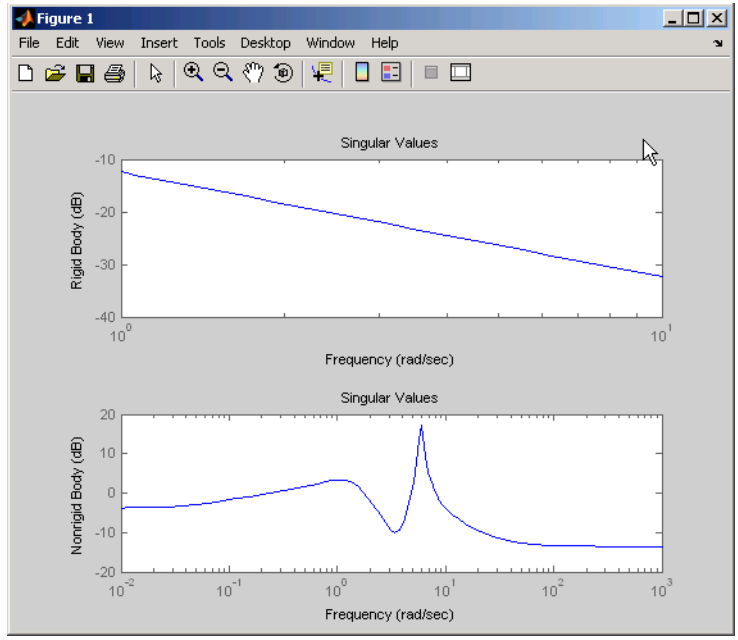
Using Modal Algorithms

Rigid Body Dynamics

In many cases, a model's jω-axis poles are important to keep after model reduction, e.g., rigid body dynamics of a flexible structure plant or integrators of a controller. A unique routine, `modreal`, serves the purpose nicely.

`modreal` puts a system into its *modal form*, with eigenvalues appearing on the diagonal of its A-matrix. Real eigenvalues appear in 1-by-1 blocks, and complex eigenvalues appear in 2-by-2 real blocks. All the blocks are ordered in ascending order, based on their eigenvalue magnitudes, by default, or descending order, based on their real parts. Therefore, specifying the number of jω-axis poles splits the model into two systems with one containing only jω-axis dynamics, the other containing the non-jω axis dynamics.

```
rand('state',5678); randn('state',1234); G = rss(30,1,1);
[Gjw,G2] = modreal(G,1); % only one rigid body dynamics
G2.d = Gjw.d; Gjw.d = 0; % put DC gain of G into G2
subplot(211);sigma(Gjw);ylabel('Rigid Body')
subplot(212);sigma(G2);ylabel('Nonrigid Body')
```



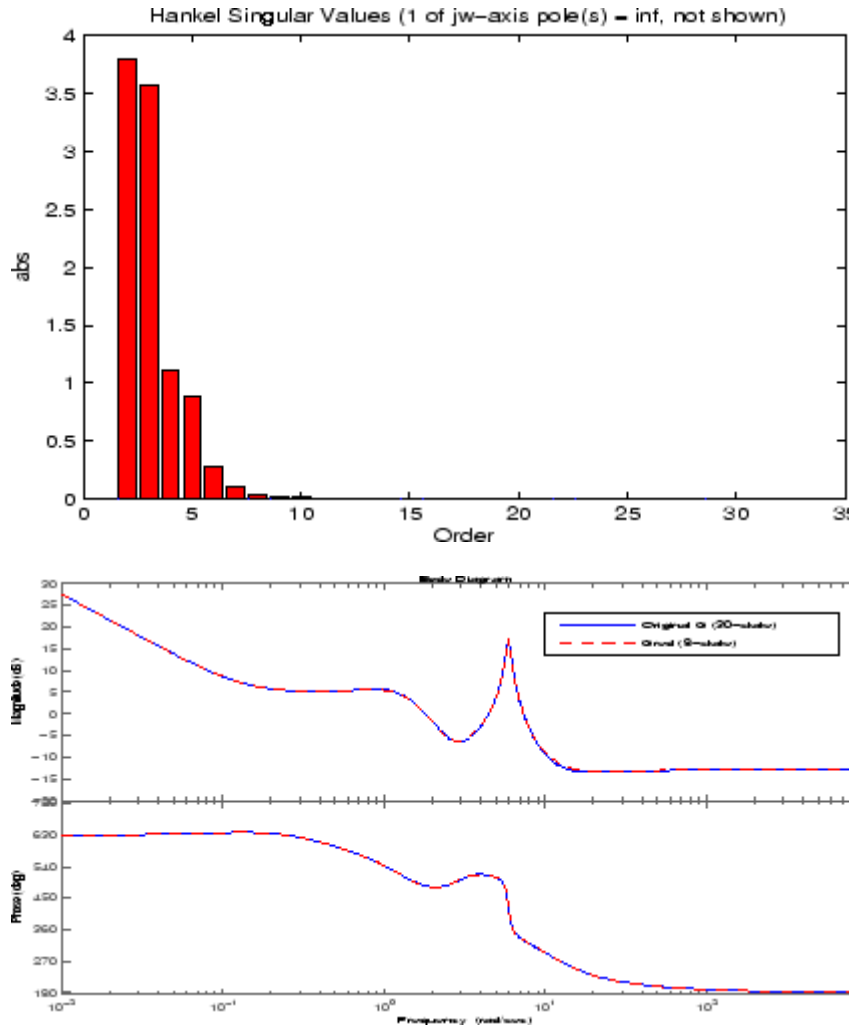
Further model reduction can be done on `G2` without any numerical difficulty. After `G2` is further reduced to `Gred`, the final approximation of the model is simply `Gjw+Gred`.

This process of splitting jω-axis poles has been built in and automated in all the model reduction routines (`balancmr`, `schurmr`, `hankelmr`, `bstmr`, `hankelsv`) so that users need not worry about splitting the model.

The following single command creates a size 8 reduced-order model from its original 30-state model:

```
rand('state',5678); randn('state',1234); G = rss(30,1,1);
[gr,info] = reduce(G); % choose a size of 8 at prompt
bode(G,'b-',gr,'r--')
```

Without specifying the size of the reduced-order model, a Hankel singular value plot is shown below.



The default algorithm `balancmr` of reduce has done a great job of approximating a 30-state model with just eight states. Again, the rigid body dynamics are preserved for further controller design.

Reducing Large-Scale Models

For some really large size problems (states > 200), `modreal` turns out to be the only way to start the model reduction process. Because of the size and numerical properties associated with those large size, and low damped dynamics, most Hankel based routines can fail to produce a good reduced-order model.

`modreal` puts the large size dynamics into the modal form, then truncates the dynamic model to an intermediate stage model with a comfortable size of 50 or so states. From this point on, those more sophisticated Hankel singular value based routines can further reduce this intermediate stage model, in a much more accurate fashion, to a smaller size for final controller design.

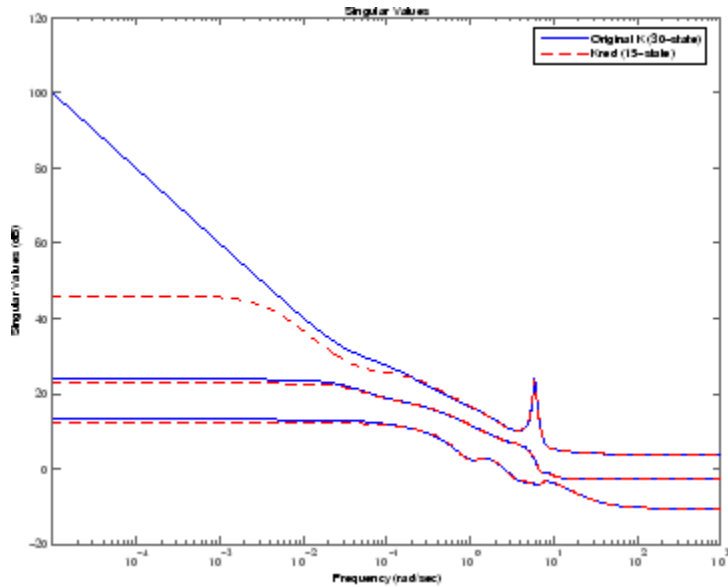
For a typical 240-state flexible spacecraft model in the spacecraft industry, applying `modreal` and `bstmr` (or any other additive routines) in sequence can reduce the original 240-state plant dynamics to a seven-state three-axis model including rigid body dynamics. Any modern robust control design technique mentioned in this toolbox can then be easily applied to this smaller size plant for a controller design.

Using Normalized Coprime Factor Methods

A special model reduction routine `ncfmr` produces a reduced-order model by truncating a balanced coprime set of a given model. It can directly simplify a modern controller with integrators to a smaller size by balanced truncation of the normalized coprime factors. It does not need `modreal` for pre-/postprocessing as the other routines do. However, the integrators will not be preserved.

```
rand('state',5678); randn('state',1234);
K= rss(30,4,3); % The same model G used in the 1st example
[Kred,info2] = ncfmr(K);
sigma(K,Kred)
```

Again, without specifying the size of the reduced-order model, any model reduction routine presented here will plot a Hankel singular value bar chart and prompt you for a reduced model size. In this case, enter `15`.



If integral control is important, previously mentioned methods (except `ncfmr`) can nicely preserve the original integrator(s) in the model.

Bibliography

[1] Glover, K., "All Optimal Hankel Norm Approximation of Linear Multivariable Systems, and Their L^∞ - Error Bounds," *Int. J. Control*, Vol. 39, No. 6, 1984, pp. 1145-1193.

[2] Zhou, K., Doyle, J.C., and Glover, K., *Robust and Optimal Control*, Englewood Cliffs, NJ, Prentice Hall, 1996.

[3] Safonov, M.G., and Chiang, R.Y., "A Schur Method for Balanced Model Reduction," *IEEE Trans. on Automat. Contr.*, Vol. 34, No. 7, July 1989, pp. 729-733.

[4] Safonov, M.G., Chiang, R.Y., and Limebeer, D.J.N., "Optimal Hankel Model Reduction for Nonminimal Systems," *IEEE Trans. on Automat. Contr.*, Vol. 35, No. 4, April 1990, pp. 496-502.

[5] Safonov, M.G., and Chiang, R.Y., "Model Reduction for Robust Control: A Schur Relative Error Method," *International J. of Adaptive Control and Signal Processing*, Vol. 2, 1988, pp. 259-272.

[6] Obinata, G., and Anderson, B.D.O., *Model Reduction for Control System Design*, London, Springer-Verlag, 2001.

4

Robustness Analysis

Uncertainty Modeling

Dealing with and understanding the effects of uncertainty are important tasks for the control engineer. Reducing the effects of some forms of uncertainty (initial conditions, low-frequency disturbances) without catastrophically increasing the effects of other dominant forms (sensor noise, model uncertainty) is the primary job of the feedback control system.

Closed-loop stability is the way to deal with the (always present) uncertainty in initial conditions or arbitrarily small disturbances.

High-gain feedback in low-frequency ranges is a way to deal with the effects of unknown biases and disturbances acting on the process output. In this case, you are forced to use roll-off filters in high-frequency ranges to deal with high-frequency sensor noise in a feedback system.

Finally, notions such as gain and phase margins (and their generalizations) help quantify the sensitivity of stability and performance in the face of *model uncertainty*, which is the imprecise knowledge of how the control input directly affects the feedback variables.

Robust Control Toolbox software has built-in features allowing you to specify model uncertainty simply and naturally. The primary building blocks, called *uncertain elements* or *atoms*, are uncertain real parameters and uncertain linear, time-invariant objects. These can be used to create coarse and simple or detailed and complex descriptions of the model uncertainty present within your process models.

Once formulated, high-level system robustness tools can help you analyze the potential degradation of stability and performance of the closed-loop system brought on by the system model uncertainty.

Creating Uncertain Models of Dynamic Systems

The two dominant forms of model uncertainty are as follows:

- Uncertainty in parameters of the underlying differential equation models

- Frequency-domain uncertainty, which often quantifies model uncertainty by describing absolute or relative uncertainty in the process's frequency response

Using these two basic building blocks, along with conventional system creation commands (such as `ss` and `tf`), you can easily create uncertain system models.

Creating Uncertain Parameters

An uncertain parameter has a name (used to identify it within an uncertain system with many uncertain parameters) and a nominal value. Being uncertain, it also has variability, described in one of the following ways:

- An additive deviation from the nominal
- A range about the nominal
- A percentage deviation from the nominal

Create a real parameter, with name 'bw', nominal value 5, and a percentage uncertainty of 10%.

```
bw = ureal('bw',5,'Percentage',10)
```

This creates a `ureal` object. View its properties using the `get` command.

```
Uncertain Real Parameter: Name bw, NominalValue 5, variability = [-10  10]%
get(bw)
            Name: 'bw'
    NominalValue: 5
            Mode: 'Percentage'
           Range: [4.5000 5.5000]
       PlusMinus: [-0.5000 0.5000]
      Percentage: [-10 10]
    AutoSimplify: 'basic'
```

Note that the range of variation (`Range` property) and the additive deviation from nominal (the `PlusMinus` property) are consistent with the `Percentage` property value.

You can create state-space and transfer function models with uncertain real coefficients using `ureal` objects. The result is an *uncertain state-space object*, or `uss`. As an example, use the uncertain real parameter `bw` to model a first-order system whose bandwidth is between 4.5 and 5.5 rad/s.

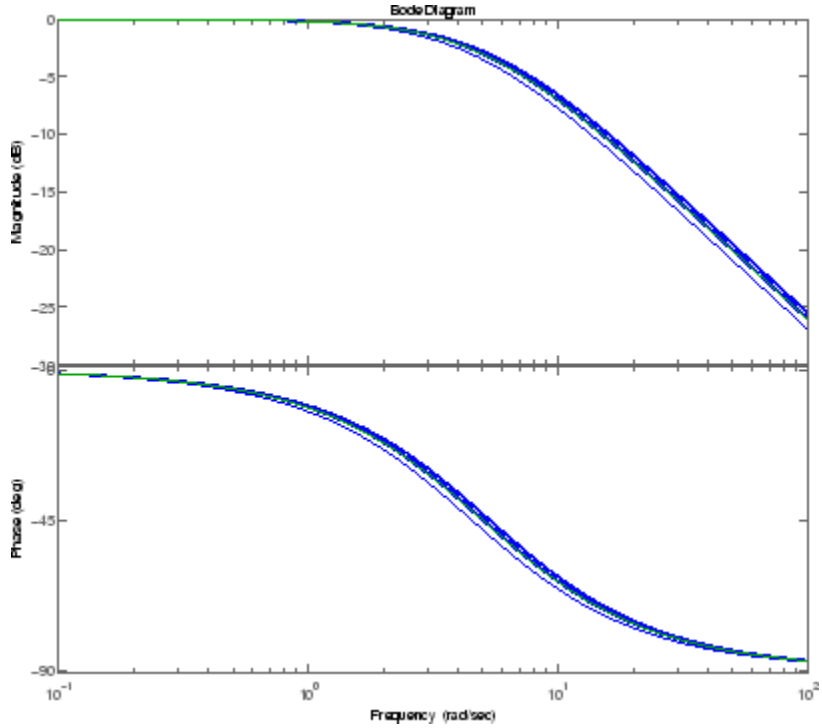
```
H = tf(1,[1/bw 1])
USS: 1 State, 1 Output, 1 Input, Continuous System
  bw: real, nominal = 5, variability = [-10  10]%, 1 occurrence
```

Note that the result `H` is an uncertain system, called a `uss` object. The nominal value of `H` is a state-space object. Verify that the pole is at –5.

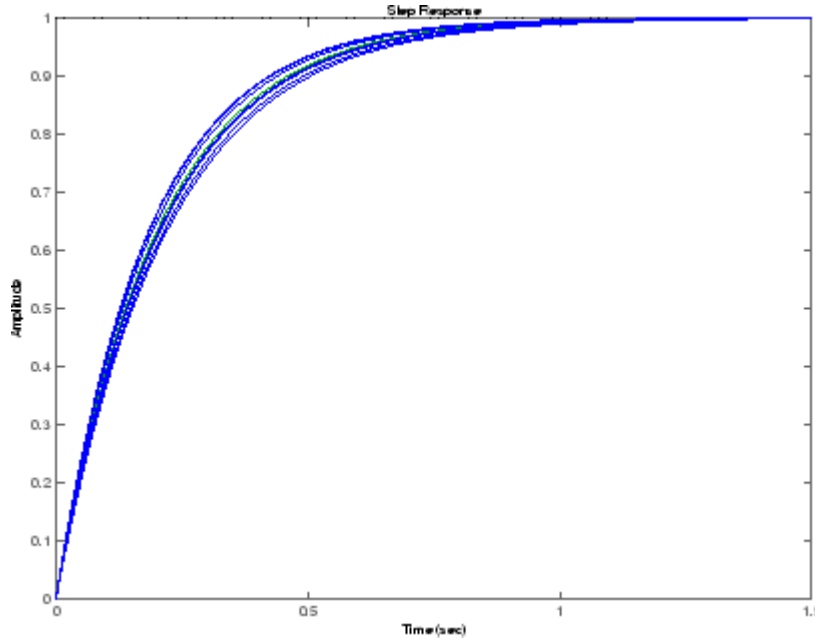
```
pole(H.NominalValue)
ans =
    -5
```

Next, use `bode` and `step` to examine the behavior of `H`.

```
bode(H,{1e-1 1e2});
```


Bode Diagram
Magnitude (dB)
0
-5
-10
-15
-20
-25
-30
Phase (deg)
-45
-90
10^{-1}
10^{0}
10^{1}
10^{2}
Frequency (rad/sec)

```
step(H)
```



While there are variations in the bandwidth and time constant of `H`, the high-frequency rolls off at 20 dB/decade regardless of the value of `bw`. You can capture the more complicated uncertain behavior that typically occurs at high frequencies using the `ultidyn` uncertain element, which is described next.

Quantifying Unmodeled Dynamics

An informal way to describe the difference between the model of a process and the actual process behavior is in terms of bandwidth. It is common to hear "The model is good out to 8 radians/second." The precise meaning is not clear, but it is reasonable to believe that for frequencies lower than, say, 5 rad/s, the model is accurate, and for frequencies beyond, say, 30 rad/s, the model is not necessarily representative of the process behavior. In the frequency range between 5 and 30, the guaranteed accuracy of the model degrades.

The uncertain linear, time-invariant dynamics object `ultidyn` can be used to model this type of knowledge. An `ultidyn` object represents an unknown linear system whose only known attribute is a uniform magnitude bound on its frequency response. When coupled with a nominal model and a

frequency-shaping filter, `ultidyn` objects can be used to capture uncertainty associated with the model dynamics.

Suppose that the behavior of the system modeled by `H` significantly deviates from its first-order behavior beyond 9 rad/s, for example, about 5% potential relative error at low frequency, increasing to 1000% at high frequency where `H` rolls off. In order to model frequency domain uncertainty as described above using `ultidyn` objects, follow these steps:

1. Create the nominal system `Gnom`, using `tf`, `ss`, or `zpk`. `Gnom` itself might already have parameter uncertainty. In this case `Gnom` is `H`, the first-order system with an uncertain time constant.

2. Create a filter `W`, called the "weight," whose magnitude represents the relative uncertainty at each frequency. The utility `makeweight` is useful for creating first-order weights with specific low- and high-frequency gains, and specified gain crossover frequency.

3. Create an `ultidyn` object `Delta` with magnitude bound equal to 1.

The uncertain model `G` is formed by `G = Gnom*(1+W*Delta)`.

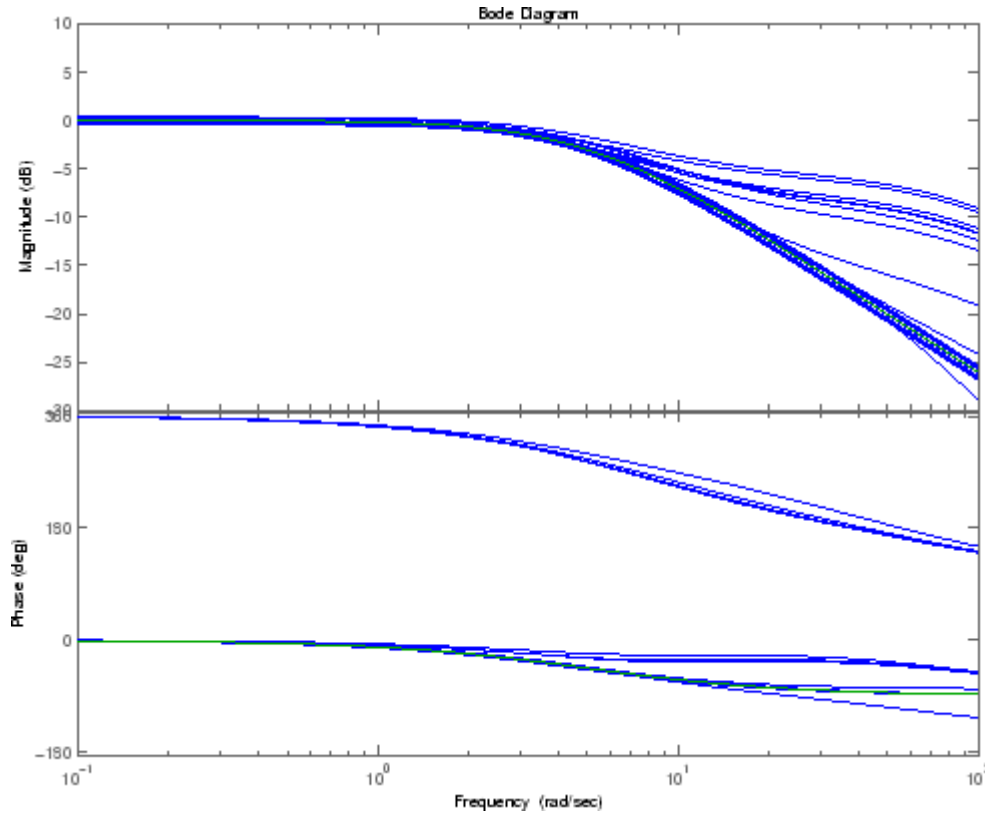
If the magnitude of `W` represents an absolute (rather than relative) uncertainty, use the formula `G = Gnom + W*Delta` instead.

The following commands carry out these steps:

```
Gnom = H;
W = makeweight(.05,9,10);
Delta = ultidyn('Delta',[1 1]);
G = Gnom*(1+W*Delta)
USS: 2 States, 1 Output, 1 Input, Continuous System
  Delta: 1x1 LTI, max. gain = 1, 1 occurrence
     bw: real, nominal = 5, variability = [-10  10]%, 1 occurrence
```

Note that the result `G` is also an uncertain system, with dependence on both `Delta` and `bw`. You can use `bode` to make a Bode plot of 20 random samples of `G`'s behavior over the frequency range [0.1 100] rad/s.

```
bode(G,{1e-1 1e2},25)
```



In the next section, you design and compare two feedback controllers for `G`.

Robustness Analysis

Next, design a feedback controller for `G`. The goals of this design are the usual ones: good steady-state tracking and disturbance rejection properties. Because the plant model is nominally a first-order lag, choose a PI control architecture. Given the desired closed-loop damping ratio ξ and natural frequency ω_n, the design equations for K_I and K_P (based on the nominal open-loop time constant of 0.2) are

$$K_I = \frac{\omega_n^2}{5},\ K_P = \frac{2\xi\omega_n}{5} - 1.$$

Follow these steps to design the controller:

1 In order to study how the uncertain behavior of `G` affects the achievable closed-loop bandwidth, design two controllers, both achieving ξ=0.707, with different ω_n: 3 and 7.5 respectively.

```
xi = 0.707;
wn = 3;
K1 = tf([(2*xi*wn/5-1) wn*wn/5],[1 0]);
wn = 7.5;
K2 = tf([(2*xi*wn/5-1) wn*wn/5],[1 0]);
```

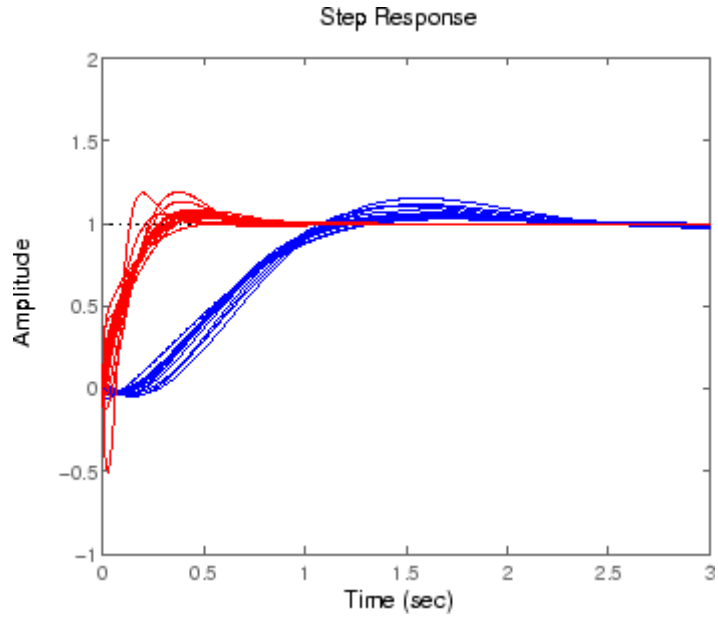
Note that the nominal closed-loop bandwidth achieved by `K2` is in a region where `G` has significant model uncertainty. It will not be surprising if the model variations lead to significant degradations in the closed-loop performance.

2 Form the closed-loop systems using `feedback`.

```
T1 = feedback(G*K1,1);
T2 = feedback(G*K2,1);
```

3 Plot the step responses of 20 samples of each closed-loop system.

```
tfinal = 3;
step(T1,'b',T2,'r',tfinal,20)
```



The step responses for `T2` exhibit a faster rise time because `K2` sets a higher closed loop bandwidth. However, the model variations have a greater effect.

You can use `robuststab` to check the robustness of stability to the model variations.

```
[stabmarg1,destabu1,report1] = robuststab(T1);
stabmarg1
stabmarg1 =
        ubound: 4.0241
        lbound: 4.0241
    destabfreq: 3.4959
[stabmarg2,destabu2,report2] = robuststab(T2);
stabmarg2
stabmarg2 =
        ubound: 1.2545
        lbound: 1.2544
    destabfreq: 10.5249
```

The `stabmarg` variable gives lower and upper bounds on the stability margin. A stability margin greater than 1 means the system is stable for all values of the modeled uncertainty. A stability margin less than 1 means there are allowable values of the uncertain elements that make the system unstable. The `report` variable briefly summarizes the analysis.

```
report1
report1 =
Uncertain System is robustly stable to modeled uncertainty.
 -- It can tolerate up to 402% of modeled uncertainty.
 -- A destabilizing combination of 402% the modeled uncertainty exists, causing an instability at
3.5 rad/s.
report2
report2 =
Uncertain System is robustly stable to modeled uncertainty.
 -- It can tolerate up to 125% of modeled uncertainty.
 -- A destabilizing combination of 125% the modeled uncertainty exists, causing an instability at
10.5 rad/s.
```

While both systems are stable for all variations, their performance is clearly affected to different degrees. To determine how the uncertainty affects closed-loop performance, you can use `wcgain` to compute the *worst-case* effect of the uncertainty on the peak magnitude of the closed-loop sensitivity (S=1/(1+GK)) function. This peak gain is typically correlated with the amount of overshoot in a step response.

To do this, form the closed-loop sensitivity functions and call `wcgain`.

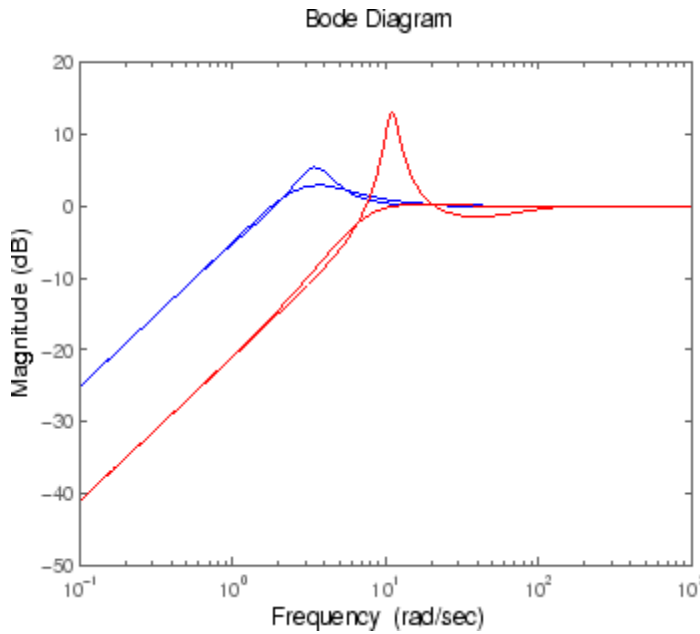
```
S1 = feedback(1,G*K1);
S2 = feedback(1,G*K2);
[maxgain1,wcu1] = wcgain(S1);
maxgain1
maxgain1 =
      lbound: 1.8684
      ubound: 1.9025
    critfreq: 3.5152
[maxgain2,wcu2] = wcgain(S2);
maxgain2
maxgain2 =
      lbound: 4.6031
```

```
    ubound: 4.6671
critfreq: 11.0231
```

The `maxgain` variable gives lower and upper bounds on the worst-case peak gain of the sensitivity transfer function, as well as the specific frequency where the maximum gain occurs. The `wcu` variable contains specific values of the uncertain elements that achieve this worst-case behavior.

You can use `usubs` to substitute these worst-case values for uncertain elements, and compare the nominal and worst-case behavior. Use `bodemag` and `step` to make the comparison.

```
bodemag(S1.NominalValue,'b',usubs(S1,wcu1),'b');
hold on
bodemag(S2.NominalValue,'r',usubs(S2,wcu2),'r');
hold off
```



Clearly, while `K2` achieves better nominal sensitivity than `K1`, the nominal closed-loop bandwidth extends too far into the frequency range where the

process uncertainty is very large. Hence the worst-case performance of K2 is inferior to K1 for this particular uncertain model.

The next section explores these robustness analysis tools further on a multiinput, multioutput system.

Multiinput, Multioutput Robustness Analysis

The previous sections focused on simple uncertainty models of single-input and single-output systems, predominantly from a transfer function perspective. You can also create uncertain state-space models made up of uncertain state-space matrices. Moreover, all the analysis tools covered thus far can be applied to these systems as well.

Consider, for example, a two-input, two-output, two-state system whose model has parametric uncertainty in the state-space matrices. First create an uncertain parameter `p`. Using the parameter, make uncertain `A` and C matrices. The B matrix happens to be not-uncertain, although you will add frequency domain input uncertainty to the model in "Adding Independent Input Uncertainty to Each Channel" on page 4-15.

```
p = ureal('p',10,'Percentage',10);
A = [0 p;-p 0];
B = eye(2);
C = [1 p;-p 1];
H = ss(A,B,C,[0 0;0 0]);
```

You can view the properties of the uncertain system `H` using the `get` command.

```
get(H)
               a: [2x2 umat]
               b: [2x2 double]
               c: [2x2 umat]
               d: [2x2 double]
       StateName: {2x1 cell}
              Ts: 0
       InputName: {2x1 cell}
      OutputName: {2x1 cell}
      InputGroup: [1x1 struct]
     OutputGroup: [1x1 struct]
    NominalValue: [2x2 ss]
     Uncertainty: [1x1 atomlist]
           Notes: {}
        UserData: []
```

The properties a, b, c, d, and StateName behave in exactly the same manner as ss objects. The properties InputName, OutputName, InputGroup, and OutputGroup behave in exactly the same manner as all the system objects (ss, zpk, tf, and frd). The NominalValue is an ss object.

Adding Independent Input Uncertainty to Each Channel

The model for H does not include actuator dynamics. Said differently, the actuator models are unity-gain for all frequencies.

Nevertheless, the behavior of the actuator for channel 1 is modestly uncertain (say 10%) at low frequencies, and the high-frequency behavior beyond 20 rad/s is not accurately modeled. Similar statements hold for the actuator in channel 2, with larger modest uncertainty at low frequency (say 20%) but accuracy out to 45 rad/s.

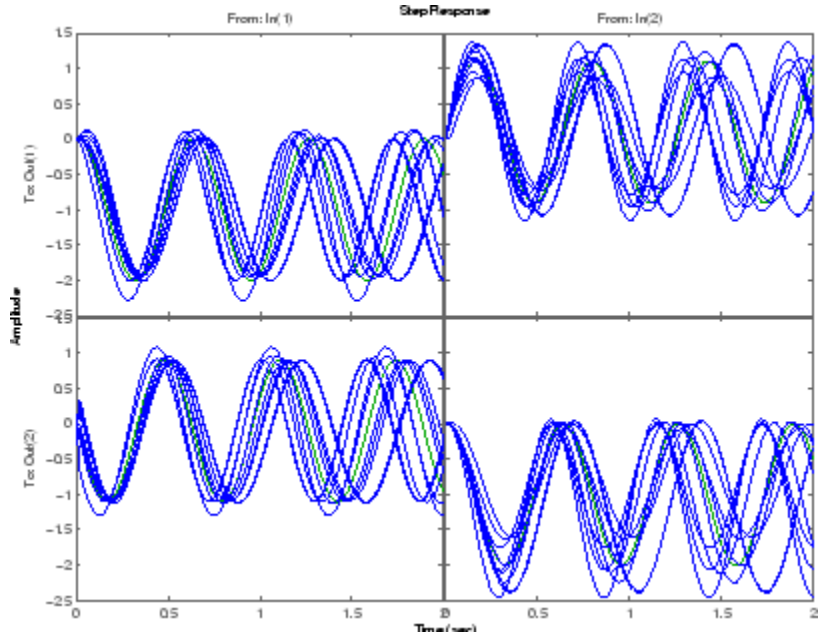
Use ultidyn objects Delta1 and Delta2 along with shaping filters W1 and W2 to add this form of frequency domain uncertainty to the model.

```
W1 = makeweight(.1,20,50);
W2 = makeweight(.2,45,50);
Delta1 = ultidyn('Delta1',[1 1]);
Delta2 = ultidyn('Delta2',[1 1]);
G = H*blkdiag(1+W1*Delta1,1+W2*Delta2)
USS: 4 States, 2 Outputs, 2 Inputs, Continuous System
  Delta1: 1x1 LTI, max. gain = 1, 1 occurrence
  Delta2: 1x1 LTI, max. gain = 1, 1 occurrence
       p: real, nominal = 10, variability = [-10  10]%, 2 occurrences
```

Note that G is a two-input, two-output uncertain system, with dependence on three uncertain elements, Delta1, Delta2, and p. It has four states, two from H and one each from the shaping filters W1 and W2, which are embedded in G.

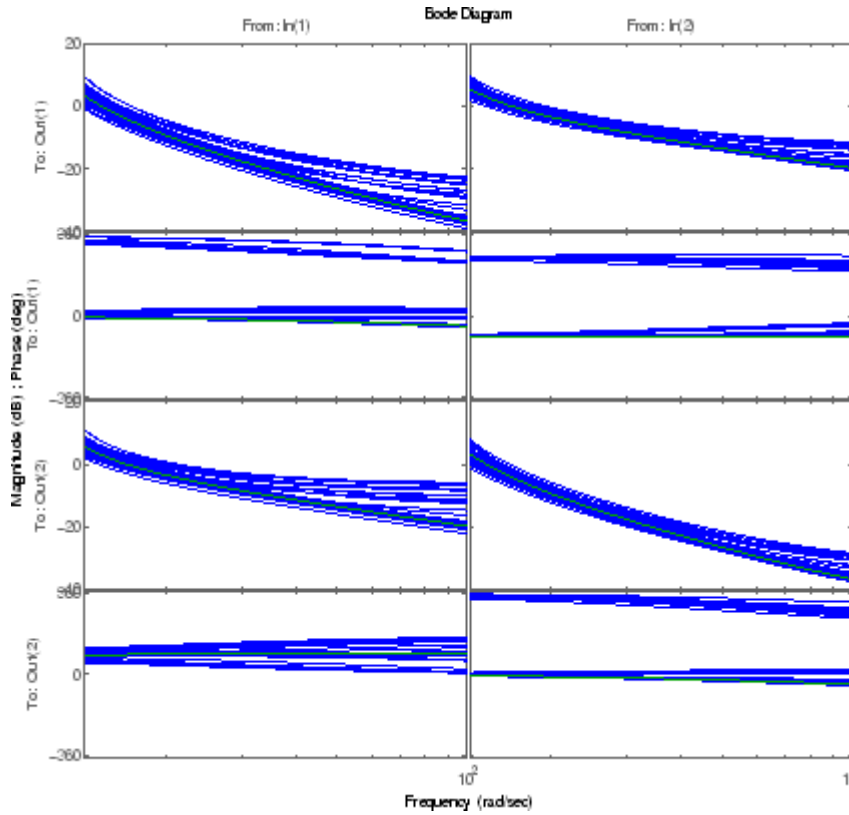
You can plot a 2-second step response of several samples of G. The 10% uncertainty in the natural frequency is obvious.

```
step(G,2)
```



You can create a Bode plot of 50 samples of `G`. The high-frequency uncertainty in the model is also obvious. For clarity, start the Bode plot beyond the resonance.

```
bode(G,{13 100},50)
```



Closed-Loop Robustness Analysis

You need to load the controller and verify that it is two-input and two-output.

```
load mimoKexample
size(K)
State-space model with 2 outputs, 2 inputs, and 9 states.
```

You can use the command `loopsens` to form all the standard plant/controller feedback configurations, including sensitivity and complementary sensitivity at both the input and output. Because `G` is uncertain, all the closed-loop systems are uncertain as well.

```
F = loopsens(G,K)
```

```
F =
     Poles: [13x1 double]
    Stable: 1
        Si: [2x2 uss]
        Ti: [2x2 uss]
        Li: [2x2 uss]
        So: [2x2 uss]
        To: [2x2 uss]
        Lo: [2x2 uss]
       PSi: [2x2 uss]
       CSo: [2x2 uss]
```

`F` is a structure with many fields. The poles of the nominal closed-loop system are in `F.Poles`, and `F.Stable` is 1 if the nominal closed-loop system is stable. In the remaining 10 fields, `S` stands for sensitivity, `T` for complementary sensitivity, and `L` for open-loop gain. The suffixes `i` and `o` refer to the input and output of the plant (`G`). Finally, `P` and `C` refer to the "plant" and "controller."

Hence `Ti` is mathematically the same as

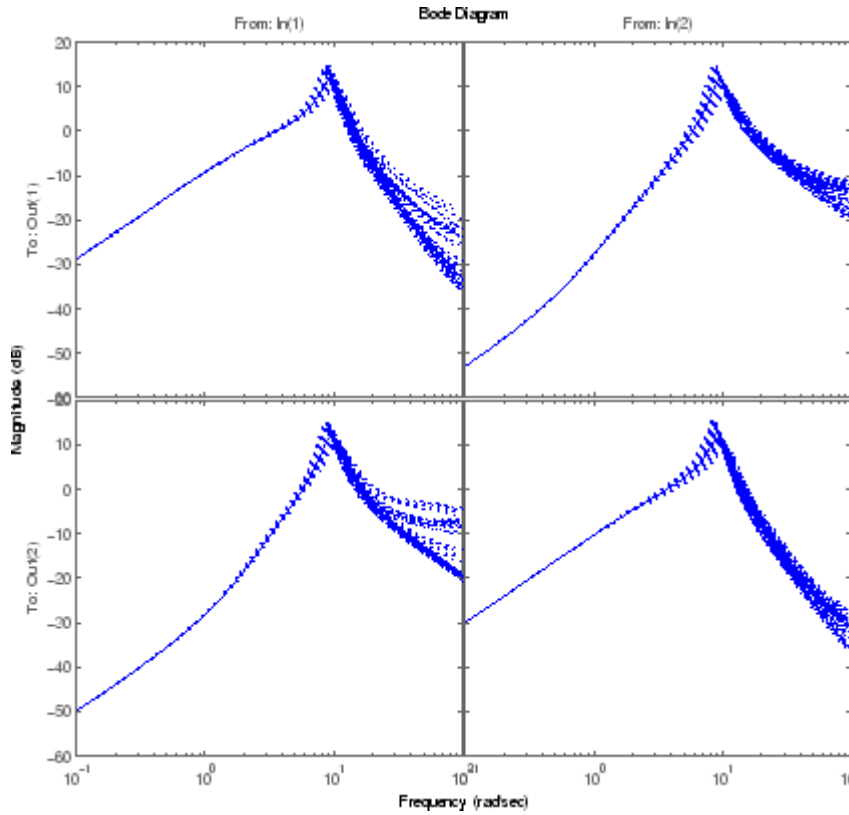
$$K(I + GK)^{-1}G$$

while `Lo` is `G*K`, and `CSo` is mathematically the same as

$$K(I + GK)^{-1}$$

You can examine the transmission of disturbances at the plant input to the plant output using `bodemag` on `F.PSi`. Graph 50 samples along with the nominal.

```
bodemag(F.PSi,':/-',{1e-1 100},50)
```



Nominal Stability Margins

You can use `loopmargin` to investigate loop-at-a-time gain and phase margins, loop-at-a-time disk margins, and simultaneous multivariable margins. They are computed for the nominal system and do not reflect the uncertainty models within `G`.

Explore the simultaneous margins individually at the plant input, individually at the plant output, and simultaneously at both input and output.

```
[I,DI,SimI,O,DO,SimO,Sim] = loopmargin(G,K);
```

The third output argument is the *simultaneous* gain and phase variations allowed in all input channels to the plant.

```
SimI
SimI =
     GainMargin: [0.1180 8.4769]
    PhaseMargin: [-76.5441 76.5441]
      Frequency: 6.2287
```

This information implies that the gain at the plant input can vary in both channels independently by factors between (approximately) `1/8` and `8`, *as well as* phase variations up to 76 degrees.

The sixth output argument is the simultaneous gain and phase variations allowed in all output channels to the plant.

```
SimO
SimO =
     GainMargin: [0.1193 8.3836]
    PhaseMargin: [-76.3957 76.3957]
      Frequency: 18.3522
```

Note that the simultaneous margins at the plant output are similar to those at the input. This is not always the case in multiloop feedback systems.

The last output argument is the simultaneous gain and phase variations allowed in all input *and* output channels to the plant. As expected, when you consider all such variations simultaneously, the margins are somewhat smaller than those at the input or output alone.

```
Sim
Sim =
     GainMargin: [0.5671 1.7635]
    PhaseMargin: [-30.8882 30.8882]
      Frequency: 18.3522
```

Nevertheless, these numbers indicate a generally robust closed-loop system, able to tolerate significant gain (more than +/-50% in each channel) and 30 degree phase variations simultaneously in all input and output channels of the plant.

Robustness of Stability Model Uncertainty

With `loopmargin`, you determined various margins of the nominal, multiloop system. These margins are computed only for the nominal system, and do not reflect the uncertainty explicitly modeled by the `ureal` and `ultidyn` objects. When you work with detailed, complex uncertain system models, the conventional margins computed by `loopmargin` might not always be indicative of the actual stability margins associated with the uncertain elements. You can use `robuststab` to check the stability margin of the system to these specific modeled variations.

In this example, use `robuststab` to compute the stability margin of the closed-loop system represented by `Delta1`, `Delta2`, and `p`.

Use any of the closed-loop systems within `F = loopsens(G,K)`. All of them, `F.Si, F.To`, etc., have the same internal dynamics, and hence the stability properties are the same.

```
[stabmarg,desgtabu,report] = robuststab(F.So);
stabmarg
stabmarg =
        ubound: 2.2175
        lbound: 2.2175
    destabfreq: 13.7576
report
report =
Uncertain System is robustly stable to modeled uncertainty.
 -- It can tolerate up to 222% of modeled uncertainty.
 -- A destabilizing combination of 222% the modeled uncertainty exists,causing an instability at
13.8 rad/s.
```

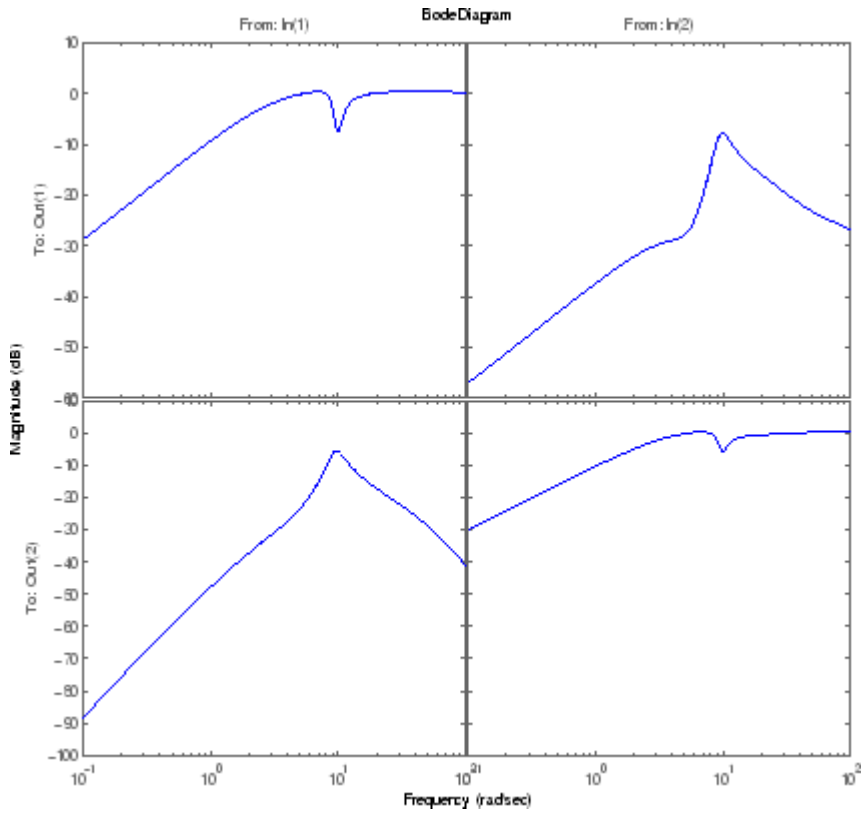
This analysis confirms what the `loopmargin` analysis suggested. The closed-loop system is quite robust, *in terms of stability*, to the variations modeled by the uncertain parameters `Delta1`, `Delta2`, and `p`. In fact, the system can tolerate more than twice the modeled uncertainty without losing closed-loop stability.

The next section studies the effects of these variations on the closed-loop output sensitivity function.

Worst-Case Gain Analysis

You can plot the Bode magnitude of the nominal output sensitivity function. It clearly shows decent disturbance rejection in all channels at low frequency.

```
bodemag(F.So.NominalValue,{1e-1 100})
```



You can compute the peak value of the maximum singular value of the frequency response matrix using `norm`.

```
[PeakNom,freq] = norm(F.So.NominalValue,'inf')
PeakNom =
    1.1317
freq =
```

```
    7.0483
```

The peak is about 1.13, occurring at a frequency of 36 rad/s.

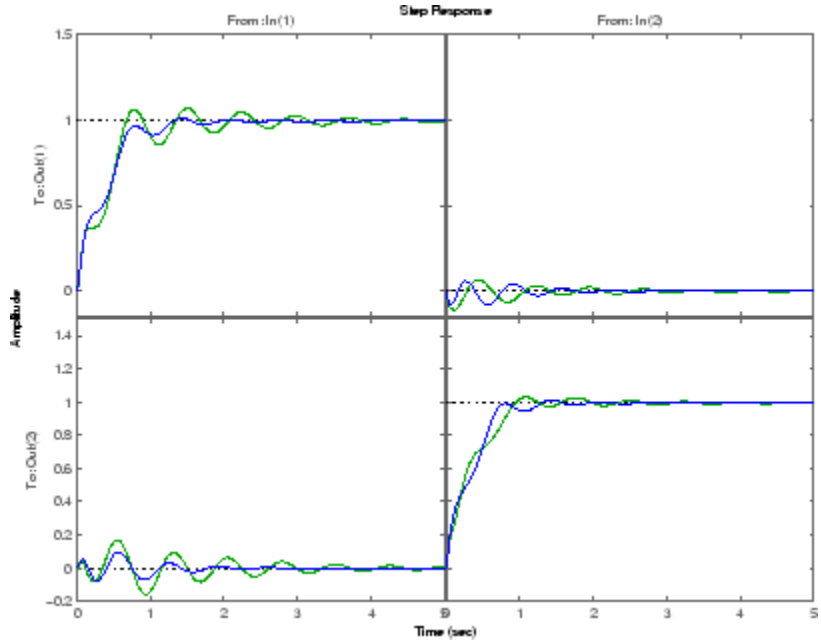
What is the maximum output sensitivity gain that is achieved when the uncertain elements `Delta1`, `Delta2`, and `p` vary over their ranges? You can use `wcgain` to answer this.

```
[maxgain,wcu] = wcgain(F.So);
maxgain
maxgain =
      lbound: 2.1017
      ubound: 2.1835
    critfreq: 8.5546
```

The analysis indicates that the worst-case gain is somewhere between 2.1 and 2.2. The frequency where the peak is achieved is about 8.5.

You can replace the values for `Delta1`, `Delta2`, and `p` that achieve the gain of 2.1, using `usubs`. Make the substitution in the output complementary sensitivity, and do a step response.

```
step(F.To.NominalValue,usubs(F.To,wcu),5)
```



The perturbed response, which is the worst combination of uncertain values in terms of output sensitivity amplification, does not show significant degradation of the command response. The settling time is increased by about 50%, from 2 to 4, and the off-diagonal coupling is increased by about a factor of about 2, but is still quite small.

Summary of Robustness Analysis Tools

Function	Description
`ureal`	Create uncertain real parameter.
`ultidyn`	Create uncertain, linear, time-invariant dynamics.
`uss`	Create uncertain state-space object from uncertain state-space matrices.
`ufrd`	Create uncertain frequency response object.
`loopsens`	Compute all relevant open and closed-loop quantities for a MIMO feedback connection.
`loopmargin`	Compute loop-at-a-time as well as MIMO gain and phase margins for a multiloop system, including the simultaneous gain/phase margins.
`robustperf`	Robustness performance of uncertain systems.
`robuststab`	Compute the robust stability margin of a nominally stable uncertain system.
`wcgain`	Compute the worst-case gain of a nominally stable uncertain system.
`wcmargin`	Compute worst-case (over uncertainty) loop-at-a-time disk-based gain and phase margins.
`wcsens`	Compute worst-case (over uncertainty) sensitivity of plant-controller feedback loop.

5

H-Infinity and Mu Synthesis

Interpretation of H-Infinity Norm

Norms of Signals and Systems

There are several ways of defining norms of a scalar signal $e(t)$ in the time domain. We will often use the 2-norm, (L_2-norm), for mathematical convenience, which is defined as

$$\|e\|_2 := \left(\int_{-\infty}^{\infty} e(t)^2 \, dt \right)^{\frac{1}{2}}.$$

If this integral is finite, then the signal e is *square integrable*, denoted as $e \in L_2$. For vector-valued signals

$$e(t) = \begin{bmatrix} e_1(t) \\ e_2(t) \\ \vdots \\ e_n(t) \end{bmatrix},$$

the 2-norm is defined as

$$\begin{aligned} \|e\|_2 &:= \left(\int_{-\infty}^{\infty} \|e(t)\|_2^2 \, dt \right)^{\frac{1}{2}} \\ &= \left(\int_{-\infty}^{\infty} e^T(t) e(t) dt \right)^{\frac{1}{2}}. \end{aligned}$$

In μ-tools the dynamic systems we deal with are exclusively linear, with state-space model

$$\begin{bmatrix} \dot{x} \\ e \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} x \\ d \end{bmatrix},$$

or, in the transfer function form,

$e(s) = T(s)d(s),\ T(s) := C(sI - A)^{-1}B + D$

Two mathematically convenient measures of the transfer matrix $T(s)$ in the frequency domain are the matrix H_2 and H_∞ norms,

$$\|T\|_2 := \left[\frac{1}{2\pi}\int_{-\infty}^{\infty}\|T(j\omega)\|_F^2\, d\omega\right]^{\frac{1}{2}}$$
$$\|T\|_\infty := \max_{\omega\in R}\bar{\sigma}\left[T(j\omega)\right],$$

where the Frobenius norm (see the MATLAB `norm` command) of a complex matrix M is

$$\|M\|_F := \sqrt{\mathrm{Trace}\left(M^* M\right)}.$$

Both of these transfer function norms have input/output time-domain interpretations. If, starting from initial condition $x(0) = 0$, two signals d and e are related by

$$\begin{bmatrix}\dot{x}\\ e\end{bmatrix} = \begin{bmatrix}A & B\\ C & D\end{bmatrix}\begin{bmatrix}x\\ d\end{bmatrix},$$

then

- For d, a unit intensity, white noise process, the steady-state variance of e is $\|T\|_2$.
- The L_2 (or RMS) gain from $d \to e$,

  $$\max_{d\neq 0}\frac{\|e\|_2}{\|d\|_2}$$

  is equal to $\|T\|_\infty$. This is discussed in greater detail in the next section.

Using Weighted Norms to Characterize Performance

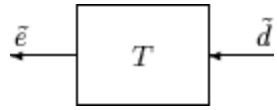
Any performance criterion must also account for

- Relative magnitude of outside influences
- Frequency dependence of signals
- Relative importance of the magnitudes of regulated variables

So, if the performance objective is in the form of a matrix norm, it should actually be a *weighted norm*

$$\|W_L T W_R\|$$

where the weighting function matrices W_L and W_R are frequency dependent, to account for bandwidth constraints and spectral content of exogenous signals. The most natural (mathematical) manner to characterize acceptable performance is in terms of the MIMO $\|\cdot\|_\infty$ (H_∞) norm. For this reason, this section now discusses some interpretations of the H_∞ norm.

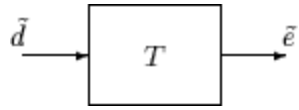


Unweighted MIMO System

Suppose T is a MIMO stable linear system, with transfer function matrix $T(s)$.

For a given driving signal $\tilde{d}(t)$, define $\tilde{e}$ as the output, as shown below.

Note that it is more traditional to write the diagram in Unweighted MIMO System: Vectors from Left to Right on page 5-4 with the arrows going from left to right as in Weighted MIMO System on page 5-6.



Unweighted MIMO System: Vectors from Left to Right

The two diagrams shown above represent the exact same system. We prefer to write these block diagrams with the arrows going right to left to be consistent with matrix and operator composition.

Assume that the dimensions of T are $n_e \times n_d$. Let ß > 0 be defined as

$$\beta := \|T\|_\infty := \max_{\omega \in R} \bar{\sigma}\left[T(j\omega)\right].$$

Now consider a response, starting from initial condition equal to 0. In that case, Parseval's theorem gives that

$$\frac{\|\tilde{e}\|_2}{\|\tilde{d}\|_2} = \frac{\left[\int_0^\infty \tilde{e}^T(t)\tilde{e}(t)dt\right]^{\frac{1}{2}}}{\left[\int_0^\infty \tilde{d}^T(t)\tilde{d}(t)dt\right]^{\frac{1}{2}}} \le \beta.$$

Moreover, there are specific disturbances d that result in the ratio $\|\tilde{e}\|_2 / \|\tilde{d}\|_2$ arbitrarily close to ß. Because of this, $\|T\|_\infty$ is referred to as the L_2 (or RMS) gain of the system.

As you would expect, a sinusoidal, steady-state interpretation of $\|T\|_\infty$ is also possible: For any frequency $\bar{\omega} \in R$, any vector of amplitudes $a \in R_{n_d}$, and any vector of phases $\phi \in R^{n_d}$, with $\|a\|_2 \le 1$, define a time signal

$$\tilde{d}(t) = \begin{bmatrix} a_1 \sin(\bar{\omega}t + \phi_1) \\ \vdots \\ a_{n_d} \sin\left(\bar{\omega}t + \phi_{n_d}\right) \end{bmatrix}.$$

Applying this input to the system T results in a steady-state response $\tilde{e}_{ss}$ of the form

$$\tilde{e}_{ss}(t) = \begin{bmatrix} b_1 \sin(\bar{\omega}t + \phi_1) \\ \vdots \\ b_{n_e} \sin\left(\bar{\omega}t + \phi_{n_e}\right) \end{bmatrix}.$$

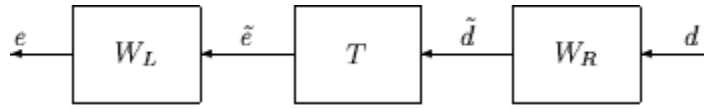
The vector $b \in R^{n_e}$ will satisfy $\|b\|_2 \le ß$. Moreover, ß, as defined in Weighted MIMO System on page 5-6, is the smallest number such that this is true for every $\|a\|_2 \le 1$, $\bar{\omega}$, and φ.

Note that in this interpretation, the vectors of the sinusoidal magnitude responses are unweighted, and measured in Euclidean norm. If realistic multivariable performance objectives are to be represented by a single MIMO $\|\cdot\|_\infty$ objective on a closed-loop transfer function, additional scalings are necessary. Because many different objectives are being lumped into one matrix and the associated cost is the norm of the matrix, it is important to use frequency-dependent weighting functions, so that different requirements can be meaningfully combined into a single cost function. Diagonal weights are most easily interpreted.

Consider the diagram of Weighted MIMO System on page 5-6, along with Unweighted MIMO System: Vectors from Left to Right on page 5-4.

Assume that W_L and W_R are diagonal, stable transfer function matrices, with diagonal entries denoted L_i and R_i.

$$W_L = \begin{bmatrix} L_1 & 0 & \dots & 0 \\ 0 & L_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & L_{n_e} \end{bmatrix}, \quad W_R = \begin{bmatrix} R_1 & 0 & \dots & 0 \\ 0 & R_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & R_{n_d} \end{bmatrix}.$$



$$e = W_L\tilde{e} = W_L T\tilde{d} = W_L T W_R d$$

Weighted MIMO System

Bounds on the quantity $\|W_L T W_R\|_\infty$ will imply bounds about the sinusoidal steady-state behavior of the signals $\tilde{d}$ and $\tilde{e}\left(= T\tilde{d}\right)$ in the diagram of Unweighted MIMO System: Vectors from Left to Right on page 5-4. Specifically, for sinusoidal signal $\tilde{d}$, the steady-state relationship between

$\tilde{e}\left(=T\tilde{d}\right)$, $\tilde{d}$ and $\|W_L T W_R\|_\infty$ is as follows. The steady-state solution $\tilde{e}_{ss}$, denoted as

$$\tilde{e}_{ss}(t)=\begin{bmatrix}\tilde{e}_1\sin(\bar{\omega}t+\phi_1)\\ \vdots \\ \tilde{e}_{n_e}\sin\left(\bar{\omega}t+\phi_{n_d}\right)\end{bmatrix} \tag{5-1}$$

satisfies

$$\sum_{i=1}^{n_e}\left|W_{L_i}(j\bar{\omega})\tilde{e}_i\right|^2\le 1$$

for all sinusoidal input signals $\tilde{d}$ of the form

$$\tilde{d}(t)=\begin{bmatrix}\tilde{d}_1\sin(\bar{\omega}t+\phi_1)\\ \vdots \\ \tilde{d}_{n_e}\sin\left(\bar{\omega}t+\phi_{n_d}\right)\end{bmatrix} \tag{5-2}$$

satisfying

$$\sum_{i=1}^{n_d}\frac{\left|\tilde{d}_i\right|^2}{\left|W_{R_i}(j\bar{\omega})\right|^2}\le 1$$

if and only if $\|W_L T W_R\|_\infty \le 1$.

This approximately (*very* approximately — the next statement is not actually correct) implies that $\|W_L T W_R\|_\infty \le 1$ if and only if for every fixed frequency $\bar{\omega}$, and all sinusoidal disturbances $\tilde{d}$ of the form Equation 5-2 satisfying

$$\left|\tilde{d}_i\right|\le\left|W_{R_i}(j\bar{\omega})\right|$$

the steady-state error components will satisfy

$$|\tilde{e}_i| \le \frac{1}{\left|W_{L_i}(j\bar{\omega})\right|}.$$

This shows how one could pick performance weights to reflect the desired frequency-dependent performance objective. Use W_R to represent the relative magnitude of sinusoids disturbances that might be present, and use $1/W_L$ to represent the desired upper bound on the subsequent errors that are produced.

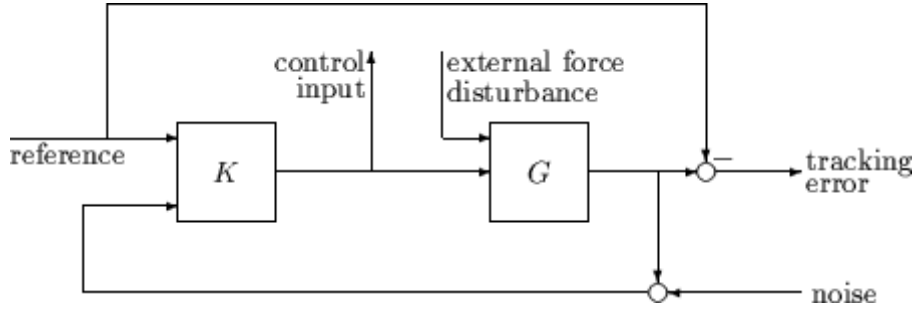
Remember, however, that the weighted H_∞ norm does *not* actually give element-by-element bounds on the components of $\tilde{e}$ based on element-by-element bounds on the components of $\tilde{d}$. The precise bound it gives is in terms of Euclidean norms of the components of $\tilde{e}$ and $\tilde{d}$ (weighted appropriately by $W_L(j\bar{\omega})$ and $W_R(j\bar{\omega})$).

H-Infinity Performance

Performance as Generalized Disturbance Rejection

The modern approach to characterizing closed-loop performance objectives is to measure the size of certain closed-loop transfer function matrices using various matrix norms. Matrix norms provide a measure of how large output signals can get for certain classes of input signals. Optimizing these types of performance objectives over the set of stabilizing controllers is the main thrust of recent optimal control theory, such as L_1, H_2, H_∞, and optimal control. Hence, it is important to understand how many types of control objectives can be posed as a minimization of closed-loop transfer functions.

Consider a tracking problem, with disturbance rejection, measurement noise, and control input signal limitations, as shown in Generalized and Weighted Performance Block Diagram on page 5-11. K is some controller to be designed and G is the system you want to control.



Typical Closed-Loop Performance Objective

A reasonable, though not precise, design objective would be to design K to keep tracking errors and control input signal small for all reasonable reference commands, sensor noises, and external force disturbances.

Hence, a natural performance objective is the closed-loop gain from exogenous influences (reference commands, sensor noise, and external force disturbances) to regulated variables (tracking errors and control input signal). Specifically, let T denote the closed-loop mapping from the outside influences to the regulated variables:

$$\underbrace{\begin{bmatrix}\text{tracking error}\\ \text{control input}\end{bmatrix}}_{\text{regulated variables}} = T \underbrace{\begin{bmatrix}\text{reference}\\ \text{external force}\\ \text{noise}\end{bmatrix}}_{\text{outside influences}}$$

You can assess performance by measuring the gain from *outside influences to regulated variables*. In other words, good performance is associated with T being small. Because the closed-loop system is a multiinput, multioutput (MIMO) dynamic system, there are two different aspects to the gain of T:

- Spatial (*vector* disturbances and *vector* errors)
- Temporal (dynamic relationship between input/output signals)

Hence the performance criterion must account for

- Relative magnitude of outside influences
- Frequency dependence of signals
- Relative importance of the magnitudes of regulated variables

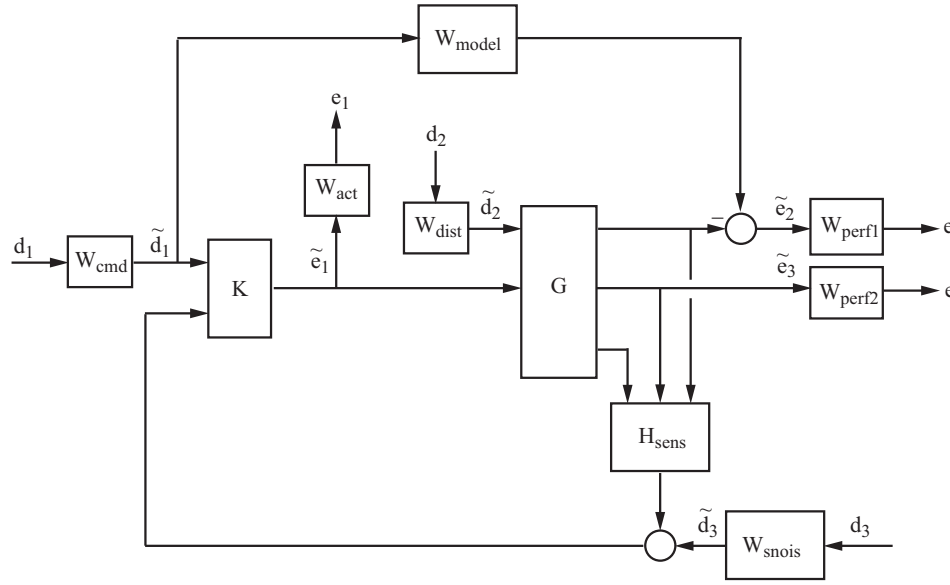
So if the performance objective is in the form of a matrix norm, it should actually be a *weighted norm*

$$\|W_L T W_R\|$$

where the weighting function matrices W_L and W_R are frequency dependent, to account for bandwidth constraints and spectral content of exogenous signals. A natural (mathematical) manner to characterize acceptable performance is in terms of the MIMO $\|\cdot\|_\infty$ (H_∞) norm. See “Interpretation of H-Infinity Norm” on page 5-2 for an interpretation of the H_∞ norm and signals.

Interconnection with Typical MIMO Performance Objectives

The closed-loop performance objectives are formulated as weighted closed-loop transfer functions that are to be made small through feedback. A generic example, which includes many relevant terms, is shown in block diagram form in Generalized and Weighted Performance Block Diagram on page 5-11. In the diagram, G denotes the plant model and K is the feedback controller.



Generalized and Weighted Performance Block Diagram

The blocks in this figure might be scalar (SISO) and/or multivariable (MIMO), depending on the specific example. The mathematical objective of H_∞ control is to make the closed-loop MIMO transfer function T_{ed} satisfy $\|T_{\mathrm{ed}}\|_\infty < 1$. The weighting functions are used to scale the input/output transfer functions such that when $\|T_{\mathrm{ed}}\|_\infty < 1$, the relationship between $\tilde{d}$ and $\tilde{e}$ is suitable.

Performance requirements on the closed-loop system are transformed into the H_∞ framework with the help of *weighting* or *scaling* functions. Weights are selected to account for the relative magnitude of signals, their frequency dependence, and their relative importance. This is captured in the figure above, where the weights or scalings [W_{cmd}, W_{dist},W_{snois}] are used to transform and scale the normalized input signals [d_1,d_2,d_3] into physical units defined as [d_1, d_2, d_3]. Similarly weights or scalings [W_{act}, W_{perf1},W_{perf2}] transform and scale physical units into normalized output signals [e_1, e_2, e_3]. An interpretation of the signals, weighting functions, and models follows.

Signal	Meaning
d_1	Normalized reference command
$\tilde{d}_1$	Typical reference command in physical units
d_2	Normalized exogenous disturbances
$\tilde{d}_2$	Typical exogenous disturbances in physical units
d_3	Normalized sensor noise
$\tilde{d}_3$	Typical sensor noise in physical units
e_1	Weighted control signals
$\tilde{e}_1$	Actual control signals in physical units
e_2	Weighted tracking errors
$\tilde{e}_2$	Actual tracking errors in physical units
e_3	Weighted plant errors
$\tilde{e}_3$	Actual plant errors in physical units

$\mathbf{W_{cmd}}$

W_{cmd} is included in H_∞ control problems that require tracking of a reference command. W_{cmd} shapes the normalized reference command signals (magnitude and frequency) into the actual (or typical) reference signals that you expect to occur. It describes the magnitude and the frequency dependence of the reference commands generated by the normalized reference signal. Normally W_{cmd} is flat at low frequency and rolls off at high frequency. For example, in a flight control problem, fighter pilots generate stick input reference commands up to a bandwidth of about 2 Hz. Suppose that the stick has a maximum travel of three inches. Pilot commands could be modeled as normalized signals passed through a first-order filter:

$$W_{cmd} = \frac{3}{\frac{1}{2 \cdot 2\pi} s + 1}.$$

W_{model}

W_{model} represents a desired ideal model for the closed-looped system and is often included in problem formulations with tracking requirements. Inclusion of an ideal model for tracking is often called a *model matching* problem, i.e., the objective of the closed-loop system is to match the defined model. For good command tracking response, you might want the closed-loop system to respond like a well-damped second-order system. The ideal model would then be

$$W_{model} = \frac{\omega^2}{s^2 + 2\zeta\omega + \omega^2}$$

for specific desired natural frequency ω and desired damping ratio ζ. Unit conversions might be necessary to ensure exact correlation between the ideal model and the closed-loop system. In the fighter pilot example, suppose that roll-rate is being commanded and 10°/second response is desired for each inch of stick motion. Then, in these units, the appropriate model is:

$$W_{model} = 10\frac{\omega^2}{s^2 + 2\zeta\omega + \omega^2}.$$

W_{dist}

W_{dist} shapes the frequency content and magnitude of the exogenous disturbances affecting the plant. For example, consider an electron microscope as the plant. The dominant performance objective is to mechanically isolate the microscope from outside mechanical disturbances, such as ground excitations, sound (pressure) waves, and air currents. You can capture the spectrum and relative magnitudes of these disturbances with the transfer function weighting matrix W_{dist}.

W_{perf1}

W_{perf1} weights the difference between the response of the closed-loop system and the ideal model W_{model}. Often you might want accurate matching of the ideal model at low frequency and require less accurate matching at higher frequency, in which case W_{perf1} is flat at low frequency, rolls off at first or

second order, and flattens out at a small, nonzero value at high frequency. The inverse of the weight is related to the allowable size of tracking errors, when dealing with the reference commands and disturbances described by W_{cmd} and W_{dist}.

$\mathbf{W_{perf2}}$

W_{perf2} penalizes variables internal to the process G, such as actuator states that are internal to G or other variables that are not part of the tracking objective.

$\mathbf{W_{act}}$

W_{act} is used to shape the penalty on control signal use. W_{act} is a frequency varying weighting function used to penalize limits on the deflection/position, deflection rate/velocity, etc., response of the control signals, when dealing with the tracking and disturbance rejection objectives defined above. Each control signal is usually penalized independently.

$\mathbf{W_{snois}}$

W_{snois} represents frequency domain models of sensor noise. Each sensor measurement feedback to the controller has some noise, which is often higher in one frequency range than another. The W_{snois} weight tries to capture this information, derived from laboratory experiments or based on manufacturer measurements, in the control problem. For example, medium grade accelerometers have substantial noise at low frequency and high frequency. Therefore the corresponding W_{snois} weight would be larger at low and high frequency and have a smaller magnitude in the mid-frequency range. Displacement or rotation measurement is often quite accurate at low frequency and in steady state, but responds poorly as frequency increases. The weighting function for this sensor would be small at low frequency, gradually increase in magnitude as a first- or second-order system, and level out at high frequency.

$\mathbf{H_{sens}}$

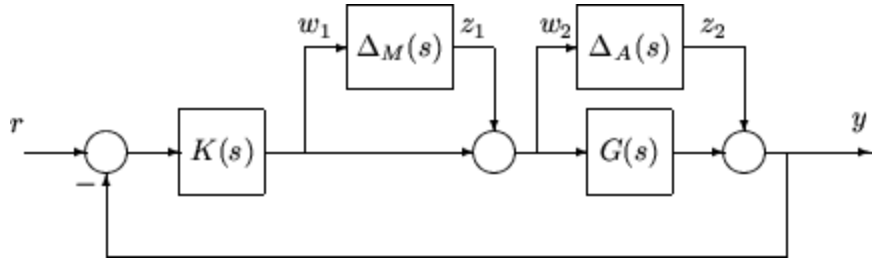
H_{sens} represents a model of the sensor dynamics or an external antialiasing filter. The transfer functions used to describe H_{sens} are based on physical

characteristics of the individual components. These models might also be lumped into the plant model G.

This generic block diagram has tremendous flexibility and many control performance objectives can be formulated in the H_∞ framework using this block diagram description.

Robustness in the H-Infinity Framework

Performance and robustness tradeoffs in control design were discussed in the context of multivariable loop shaping in "Tradeoff Between Performance and Robustness" on page 2-2. In the H_∞ control design framework, you can include robustness objectives as additional disturbance to error transfer functions — disturbances to be kept small. Consider the following figure of a closed-loop feedback system with additive and multiplicative uncertainty models.



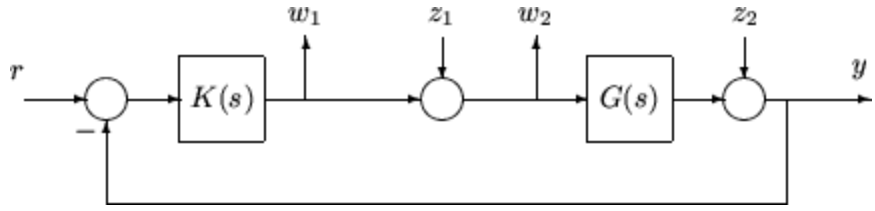
The transfer function matrices are defined as:

$$TF(s)_{z_1 \to w_1} = T_I(s) = KG(I + GK)^{-1}$$

$$TF(s)_{z_2 \to w_2} = KS_O(s) = K(I + GK)^{-1}$$

where $T_I(s)$ denotes the input complementary sensitivity function and $S_O(s)$ denotes the output sensitivity function. Theorems 1 and 2 in Chapter 2, "Multivariable Loop Shaping" give bounds on the size of the transfer function matrices from z_1 to w_1 and z_2 to w_2 to ensure that the closed-loop system is robust to multiplicative uncertainty, $\Delta_M(s)$, at the plant input, and additive uncertainty, $\Delta_A(s)$, around the plant $G(s)$. In the H_∞ control problem formulation, the robustness objectives enter the synthesis procedure as

additional input/output signals to be kept small. The interconnection with the uncertainty blocks removed follows.



The H_∞ control robustness objective is now in the same format as the performance objectives, that is, to minimize the H_∞ norm of the transfer matrix from z, $[z_1,z_2]$, to w, $[w_1,w_2]$.

Weighting or scaling matrices are often introduced to shape the frequency and magnitude content of the sensitivity and complementary sensitivity transfer function matrices. Let W_M correspond to the multiplicative uncertainty and W_A correspond to the additive uncertainty model. $\Delta_M(s)$ and $\Delta_A(s)$ are assumed to be a norm bounded by 1, i.e., $|\Delta_M(s)|<1$ and $|\Delta_A(s)|<1$. Hence as a function of frequency, $|W_M(j\omega)|$ and $|W_A(j\omega)|$ are the respective sizes of the largest anticipated additive and multiplicative plant perturbations.

The multiplicative weighting or scaling W_M represents a percentage error in the model and is often small in magnitude at low frequency, between 0.05 and 0.20 (5% to 20% modeling error), and growing larger in magnitude at high frequency, 2 to 5 ((200% to 500% modeling error). The weight will transition by crossing a magnitude value of 1, which corresponds to 100% uncertainty in the model, at a frequency at least twice the bandwidth of the closed-loop system. A typical multiplicative weight is

$$W_M = 0.10\frac{\frac{1}{5}s+1}{\frac{1}{200}s+1}.$$

By contrast, the additive weight or scaling W_A represents an absolute error that is often small at low frequency and large in magnitude at high frequency. The magnitude of this weight depends directly on the magnitude of the plant model, $G(s)$.

Functions for Control Design

The term control system design refers to the process of synthesizing a feedback control law that meets design specifications in a closed-loop control system. The design methods are iterative, combining parameter selection with analysis, simulation, and insight into the dynamics of the plant. Robust Control Toolbox software provides a set of commands that you can use for a broad range of multivariable control applications, including

- H_2 control design
- H_∞ standard and loop-shaping control design
- H_∞ tuning of controllers with fixed structure
- H_∞ normalized coprime factor control design
- Mixed H_2/H_∞ control design
- μ-synthesis via *D-K* and *D-G-K* iteration
- Sampled-data H_∞ control design

These functions cover both continuous and discrete-time problems. The following table summarizes the H_2 and H_∞ control design commands.

Function	Description
`augw`	Augments plant weights for mixed-sensitivity control design
`h2hinfsyn`	Mixed H_2/H_∞ controller synthesis
`h2syn`	H_2 controller synthesis
`hinfsyn`	H_∞ controller synthesis
`hinfstruct`	H_∞ tuning of fixed control structures
`loopsyn`	H_∞ loop-shaping controller synthesis
`ltrsyn`	Loop-transfer recovery controller synthesis
`mixsyn`	H_∞ mixed-sensitivity controller synthesis
`ncfsyn`	H_∞ normalized coprime factor controller synthesis
`sdhinfsyn`	Sample-data H_∞ controller synthesis

The following table summarizes μ-synthesis control design commands.

Function	Description
`dksyn`	Synthesis of a robust controller via μ-synthesis
`dkitopt`	Create a `dksyn` options object
`drawmag`	Interactive mouse-based sketching and fitting tool
`fitfrd`	Fit scaling frequency response data with LTI model
`fitmagfrd`	Fit scaling magnitude data with stable, minimum-phase model

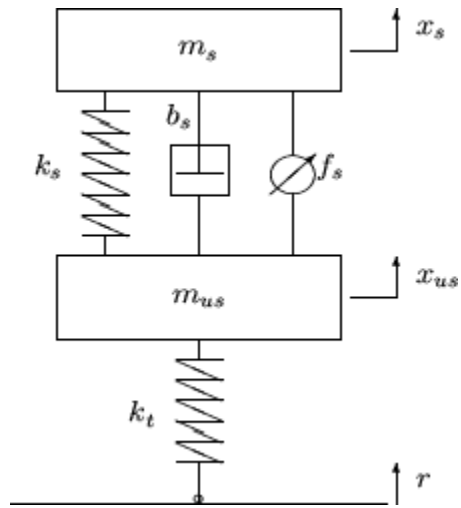
Application of H-Infinity and Mu to Active Suspension Control

Conventional passive suspensions employ a spring and damper between the car body and wheel assembly, representing a tradeoff between conflicting performance metrics such as passenger comfort, road holding, and suspension deflection. Active suspensions allow the designer to balance these objectives using a hydraulic actuator, controlled by feedback, between the chassis and wheel assembly.

In this section, you design an active suspension system for a quarter car body and wheel assembly model using the H_∞ control design technique. You will see the tradeoff between passenger comfort, i.e., minimizing car body travel, versus suspension travel as the performance objective.

Quarter Car Suspension Model

The quarter car model shown is used to design active suspension control laws.



The sprung mass m_s represents the car chassis, while the unsprung mass m_{us} represents the wheel assembly. The spring, k_s, and damper, b_s, represent a passive spring and shock absorber that are placed between the car body and the wheel assembly, while the spring $k_t$serves to model the compressibility of

the pneumatic tire. The variables x_s, x_{us}, and r are the car body travel, the wheel travel, and the road disturbance, respectively. The force f_s, kN, applied between the sprung and unsprung masses, is controlled by feedback and represents the active component of the suspension system. The dynamics of the actuator are ignored in this example, and assume that the control signal is the force f_s. Defining $x_1 := x_s$, $x_2 := \dot{x}_s$, $x_3 := x_{us}$, and $x_4 := \dot{x}_{us}$, the following is the state-space description of the quarter car dynamics.

$$\dot{x}_1 = x_2$$
$$\dot{x}_2 = -\frac{1}{m_s}\left[k_s(x_1 - x_3) + b_s(x_2 - x_4) - f_s\right]$$
$$\dot{x}_3 = x_4$$
$$\dot{x}_1 = \frac{1}{m_{us}}\left[k_s(x_1 - x_3) + b_s(x_2 - x_4) - k_t(x_3 - r) - f_s\right].$$

The following component values are taken from reference [9].

```
ms = 290;     % kg
mus = 59;     % kg
bs = 1000;    % N/m/s
ks = 16182 ; % N/m
kt = 190000; % N/m
```

A linear, time-invariant model of the quarter car model, `qcar`, is constructed from the equations of motion and parameter values. The inputs to the model are the road disturbance and actuator force, respectively, and the outputs are the car body deflection, acceleration, and suspension deflection.

```
A12 = [ 0 1 0 0; [-ks -bs ks bs]/ms ];
A34 = [ 0 0 0 1; [ks bs -ks-kt -bs]/mus];
B12 = [0 0; 0 10000/ms];
B34 = [0 0; [kt -10000]/mus];
C   = [1 0 0 0; A12(2,:); 1 0 -1 0; 0 0 0 0];
D   = [0 0; B12(2,:); 0 0; 0 1];
qcar = ss([A12; A34],[B12; B34],C,D)
```

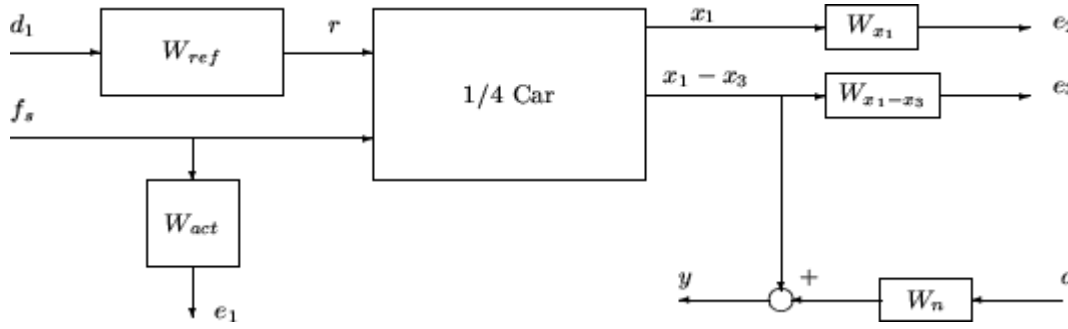
It is well known [8] that the acceleration transfer function has an invariant point at the *tirehop frequency*, 56.7 rad/s. Similarly, the suspension deflection

transfer function has an invariant point at the *rattlespace frequency*, 23.3 rad/s. The tradeoff between passenger comfort and suspension deflection is because it is not possible to simultaneously keep both transfer functions small around the tirehop frequency and in the low frequency range.

Linear H-Infinity Controller Design

The design of linear suspension controllers that emphasize either passenger comfort or suspension deflection. The controllers in this section are designed using linear H_∞ synthesis [5]. As is standard in the H_∞ framework, the performance objectives are achieved via minimizing weighted transfer function norms.

Weighting functions serve two purposes in the H_∞ framework: They allow the direct comparison of different performance objectives with the same norm, and they allow for frequency information to be incorporated into the analysis. For more details on H_∞ control design, refer to [4], [6], [7], [11], and [13]. A block diagram of the H_∞ control design interconnection for the active suspension problem is shown below.



The measured output or feedback signal y is the suspension deflection x_1–x_3. The controller acts on this signal to produce the control input, the hydraulic actuator force f_s. The block W_n serves to model sensor noise in the measurement channel. W_n is set to a sensor noise value of 0.01 m.

```
Wn = 0.01;
```

In a more realistic design, W_n would be frequency dependent and would serve to model the noise associated with the displacement sensor. The weight W_{ref}

is used to scale the magnitude of the road disturbances. Assume that the maximum road disturbance is 7 cm and hence choose `Wref = 0.07`.

```
Wref = 0.07;
```

The magnitude and frequency content of the control force f_s are limited by the weighting function W_{act}. Choose

$$W_{act} = \frac{100}{13}\frac{s+50}{s+500}.$$

The magnitude of the weight increases above 50 rad/s in order to limit the closed-loop bandwidth.

```
Wact = (100/13)*tf([1 50],[1 500]);
```

H-Infinity Control Design 1

The purpose of the weighting functions W_{x_1} and $W_{x_1-x_3}$ is to keep the car deflection and the suspension deflection small over the desired frequency ranges. In the first design, you are designing the controller for passenger comfort, and hence the car body deflection x_1 is penalized.

```
Wx1 = 8*tf(2*pi*5,[1 2*pi*5]);
```

The weight magnitude rolls off above 5×2π rad/s to respect a well-known H_∞ design rule of thumb that requires the performance weights to roll off before an open-loop zero (56.7 rad/s in this case). The suspension deflection weight $W_{x_1-x_3}$ is not included in this control problem formulation.

You can construct the weighted H_∞ plant model for control design, denoted `qcaric1`, using the `sysic` command. The control signal corresponds to the last input to `qcaric1`, `fs`. The car body acceleration, which is noisy, is the measured signal and corresponds to the last output of `qcaric1`.

```
systemnames = 'qcar Wn Wref Wact Wx1';
inputvar = '[ d1; d2; fs ]';
outputvar = '[ Wact; Wx1; qcar(3)+Wn ]';
input_to_qcar = '[ Wref; fs]';
input_to_Wn = '[ d2 ]';
```

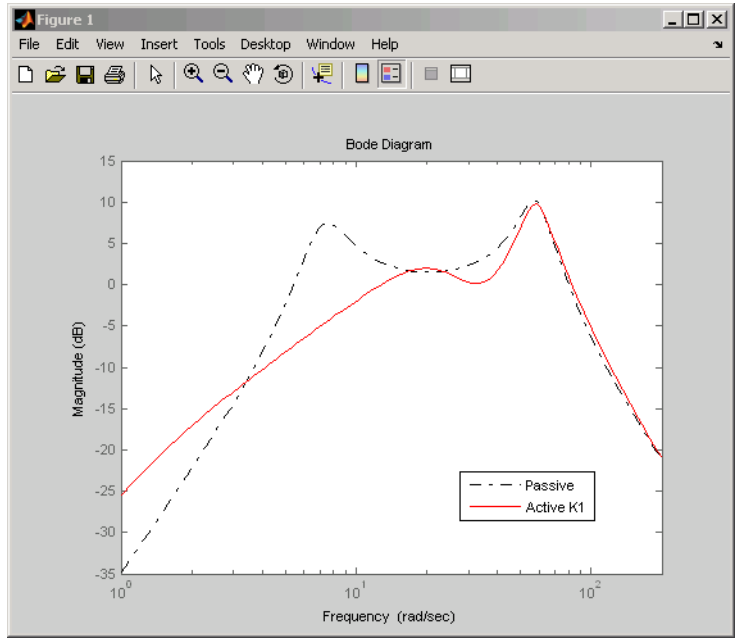
```
input_to_Wref = '[ d1 ]';
input_to_Wact = '[ fs ]';
input_to_Wx1 = '[ qcar(1) ]';
qcaric1 = sysic;
```

An H_∞ controller is synthesized with the `hinfsyn` command. There is one control input, the hydraulic actuator force, and one measurement signal, the car body acceleration.

```
ncont = 1;
nmeas = 1;
[K1,Scl1,gam1] = hinfsyn(qcaric1,nmeas,ncont);
CL1 = lft(qcar([1:4 3],1:2),K1);
sprintf('H-infinity controller K1 achieved a norm of %2.5g',gam1)
ans =
H-infinity controller K1 achieved a norm of 0.61043
```

You can analyze the H_∞ controller by constructing the closed-loop feedback system `CL1`. Bode magnitude plots of the passive suspension and active suspension are shown in the following figure.

```
bodemag(qcar(3,1),'k-.',CL1(3,1),'r-',logspace(0,2.3,140))
```



H-Infinity Control Design 2

In the second design, you are designing the controller to keep the suspension deflection transfer function small. Hence the road disturbance to suspension deflection x_1–x_3 is penalized via the weighting function `Wx1x3`. The `Wx1x3` weight magnitude rolls off above 10 rad/s to roll off before an open-loop zero (23.3 rad/s) in the design.

```
Wx1x3 = 25*tf(1,[1/10 1]);
```

The car deflection weight W_{x_1} is not included in this control problem formulation. You can construct the weighted H_∞ plant model for control design, denoted `qcaric2`, using the `sysic` command. As an alternative, you can create `qcaric2` using `iconnect` objects. The same control and measurements are used as in the first design.

```
M = iconnect;
d = icsignal(2);
fs = icsignal(1);
```

```
ycar = icsignal(size(qcar,1));
M.Equation{1} = equate(ycar,qcar*[Wref*d(1); fs]);
M.Input = [d;fs];
M.Output = [Wact*fs;Wx1x3*ycar(1);ycar(2)+Wn*d(2)];
qcaric2 = M.System;
```

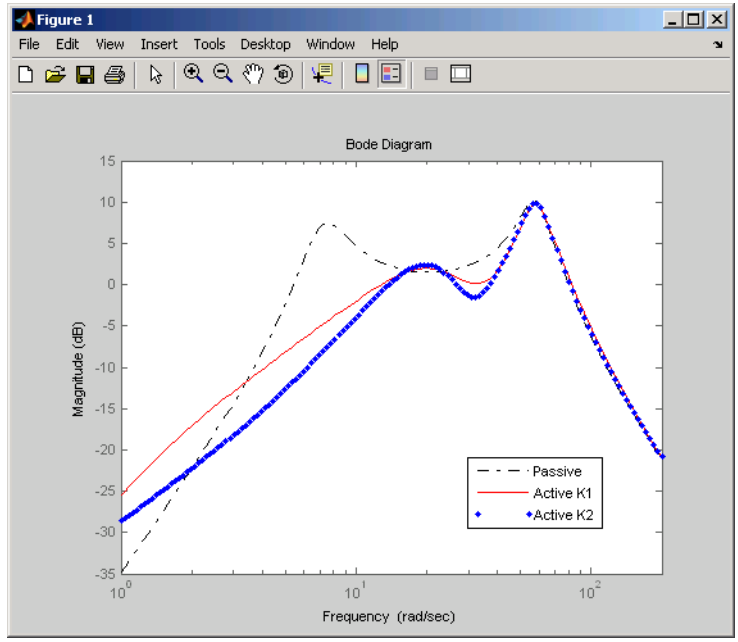
The second H_∞ controller is synthesized with the `hinfsyn` command.

```
[K2,Scl2,gam2] = hinfsyn(qcaric2,nmeas,ncont);
CL2 = lft(qcar([1:4 2],1:2),K2);
sprintf('H-infinity controller K2 achieved a norm of %2.5g',gam2)
ans =
H-infinity controller K2 achieved a norm of 0.89949
```

Recall that this H_∞ control design emphasizes minimization of suspension deflection over passenger comfort, whereas the first H_∞ design focused on passenger comfort.

You can analyze the H_∞ controller by constructing the closed-loop feedback system `CL2`. Bode magnitude plots of the transfer function from road disturbance to suspension deflection for both controllers and the passive suspension system are shown in the following figure.

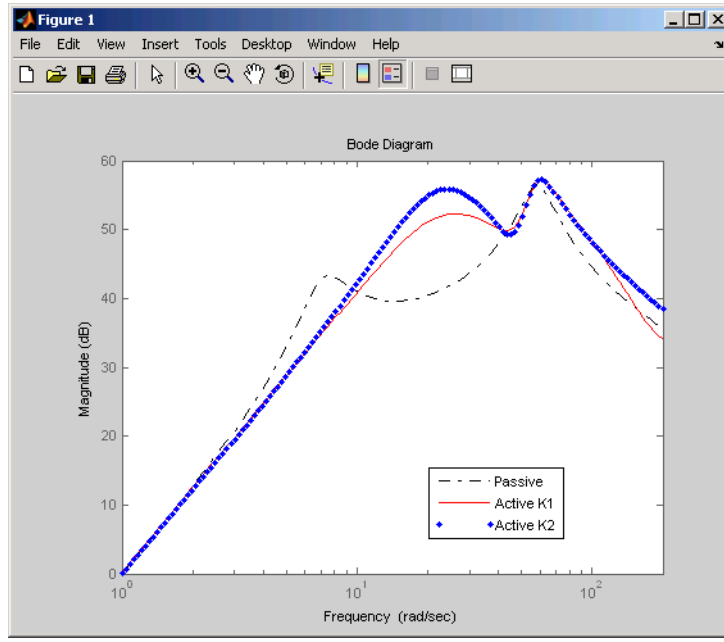
```
bodemag(qcar(3,1),'k-.',CL1(3,1),'r-',CL2(3,1),'b.',...
   logspace(0,2.3,140))
```



The dotted and solid lines in the figure are the closed-loop frequency responses that result from the different performance weighting functions selected. Observe the reduction in suspension deflection in the vicinity of the tirehop frequency, ω_1= 56.7 rad/s, and the corresponding increase in the acceleration frequency response in this vicinity. Also, compared to design 1, a reduction in suspension deflection has been achieved for frequencies below the rattlespace frequency, ω_2= 23.3 rad/s.

The second H_∞ control design attenuates both resonance modes, whereas the first controller focused its efforts on the first mode, the rattlespace frequency at 23.3 rad/s.

```
bodemag(qcar(2,1),'k-.',CL1(2,1),'r-',CL2(2,1),'b.',...
   logspace(0,2.3,140))
```



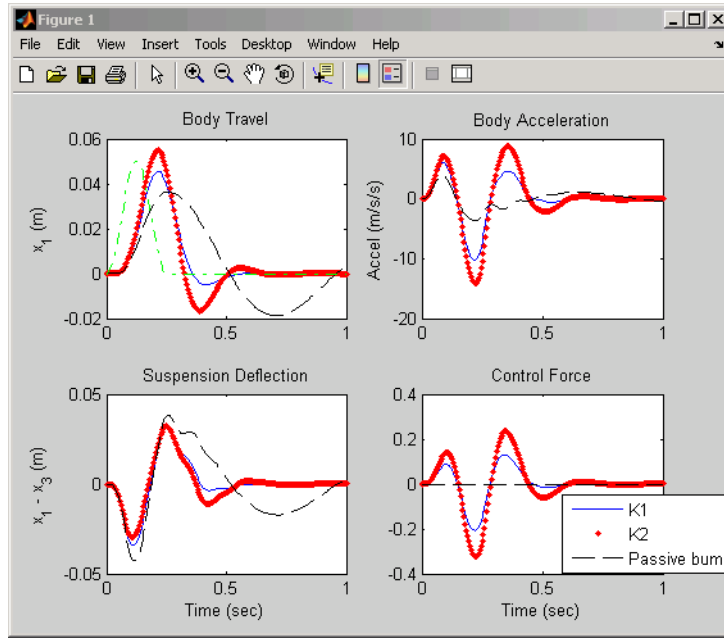
All the analysis till now has been in the frequency domain. Time-domain performance characteristics are critical to the success of the active suspension system on the car. Time response plots of the two H_∞ controllers are shown in following figures. The dashed, solid, and dotted lines correspond to the passive suspension, H_∞ controller 1, and controller 2 respectively. All responses correspond to the road disturbance $r(t)$:

$r(t) = a(1 - \cos 8\pi t)$ for $0 \le t \le 0.25$s

$r(t) = 0$ otherwise.

where a=0.025 corresponds to a road bump of peak magnitude 5 cm. In the following plots, observe that the acceleration response of design 1 to the 5 cm bump is very good; however the suspension deflection is larger than for design 2. This is because suspension deflection was not penalized in this design. The suspension deflection response of design 2 to a 5 cm bump is good. However, the acceleration response to the 5 cm bump is much inferior to design 1. Once again this is because car body displacement and acceleration were not penalized in design 2.

```
time = 0:0.005:1;
roaddist = 0*time;
roaddist(1:51) = 0.025*(1-cos(8*pi*time(1:51)));
[p1,t] = lsim(qcar(1:4,1),roaddist,time);
[y1,t] = lsim(CL1(1:4,1),roaddist,time);
[y2,t] = lsim(CL2(1:4,1),roaddist,time);
subplot(221)
plot(t,y1(:,1),'b-',t,y2(:,1),'r.',t,p1(:,1),'k--',t,...
roaddist,'g-.')
 title('Body Travel')
 ylabel('x_1 (m)')
subplot(222)
 plot(t,y1(:,2),'b-',t,y2(:,2),'r.',t,p1(:,2),'k--')
 title('Body Acceleration')
 ylabel('Accel (m/s/s)')
subplot(223)
 plot(t,y1(:,3),'b-',t,y2(:,3),'r.',t,p1(:,3),'k--')
 title('Suspension Deflection')
 xlabel('Time (sec)')
 ylabel('x_1 - x_3 (m)')
subplot(224)
 plot(t,y1(:,4),'b-',t,y2(:,4),'r.',t,p1(:,4),'k--')
 title('Control Force')
 xlabel('Time (sec)')
 ylabel('fs (10kN)')
```



Designs 1 and 2 represent extreme ends of the performance tradeoff spectrum. This section described synthesis of H_∞ to achieve the performance objectives on the active suspension system. Equally, if not more important, is the design of controllers robust to model error or uncertainty.

The goal of every control design is to achieve the desired performance specifications on the nominal model as well as other plants that are *close* to the nominal model. In other words, you want to achieve the performance objectives in the presence of model error or uncertainty. This is called *robust performance*. In the next section, you will design a controller that achieves robust performance using the μ-synthesis control design methodology. The active suspension system again serves as the example. Instead of assuming a perfect actuator, a nominal actuator model with modeling error is introduced into the control problem.

Control Design via Mu Synthesis

The active suspension H_∞ controllers designed in the previous section ignored the hydraulic actuator dynamics. In this section, you will include a first-order

model of the hydraulic actuator dynamics as well as an uncertainty model to account for differences between the actuator model and the actual actuator dynamics.

The nominal model for the hydraulic actuator is

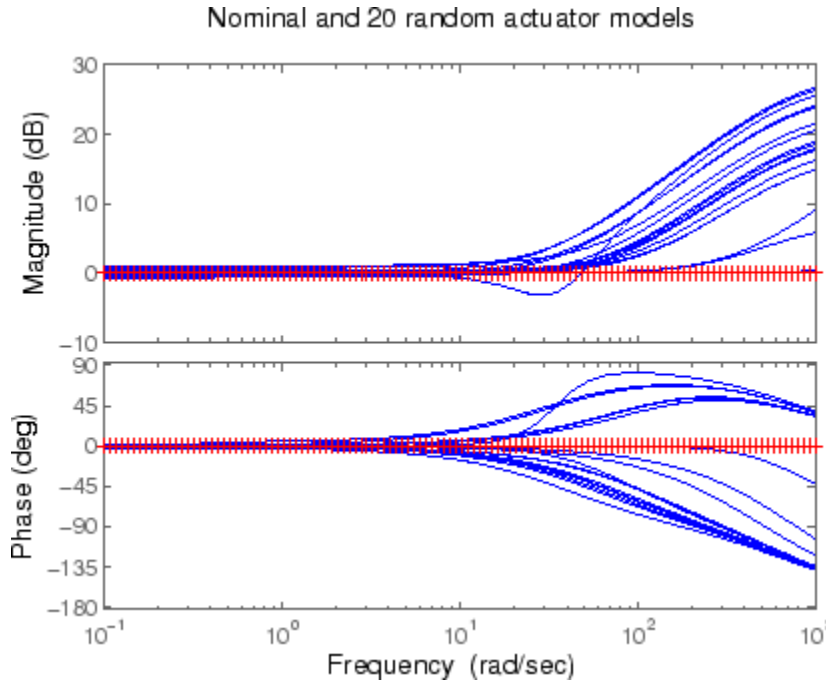
```
actnom = tf(1,[1/60 1]);
```

The actuator model itself is uncertain. You can describe the actuator model error as a set of possible models using a weighting function. At low frequency, below 4 rad/s, it can vary up to 10% from its nominal value. Around 4 rad/s the percentage variation starts to increase and reaches 400% at approximately 800 rad/s. The model uncertainty is represented by the weight W_{unc}, which corresponds to the frequency variation of the model uncertainty and the uncertain LTI dynamic object Δ_{unc} defined as `unc`.

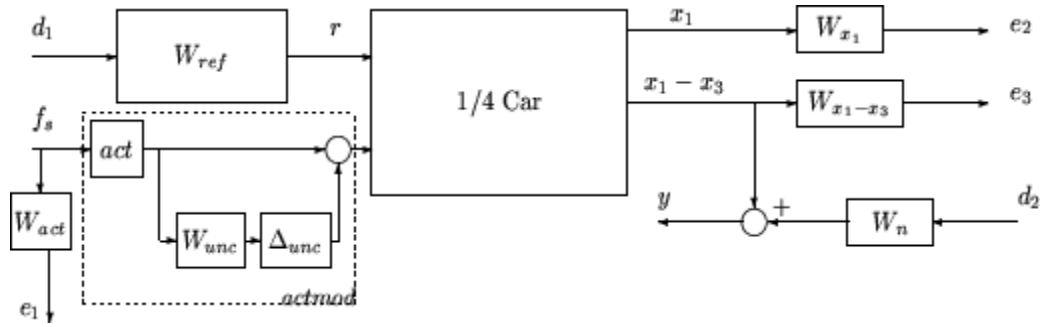
```
Wunc = 0.10*tf([1/4 1],[1/800 1]);
unc = ultidyn('unc',[1 1]);
actmod = actnom*(1+ Wunc*unc)
USS: 2 States, 1 Output, 1 Input, Continuous System
  unc: 1x1 LTI, max. gain = 1, 1 occurrence
```

The actuator model `actmod` is an uncertain state-space system. The following Bode plot shows the nominal actuator model, `actnom`, denoted with a `'+'` symbol, and 20 random actuator models described by `actmod`.

```
bode(actnom,'r+',actmod,'b',logspace(-1,3,120))
```



The uncertain actuator model `actmod` represents the model of the hydraulic actuator used for control. The revised control design interconnection diagram is



You are designing the controller for passenger comfort, as in the first H_∞ control design, hence the car body deflection x_1 is penalized with W_{x1}. The uncertain weighted H_∞ plant model for control design, denoted `qcaricunc`,

is using the sysic command. As previously described, the control signal corresponds to the last input to qcaric1, fs. The car body acceleration, which is noisy, is the measured signal and corresponds to the last output of qcaricunc.

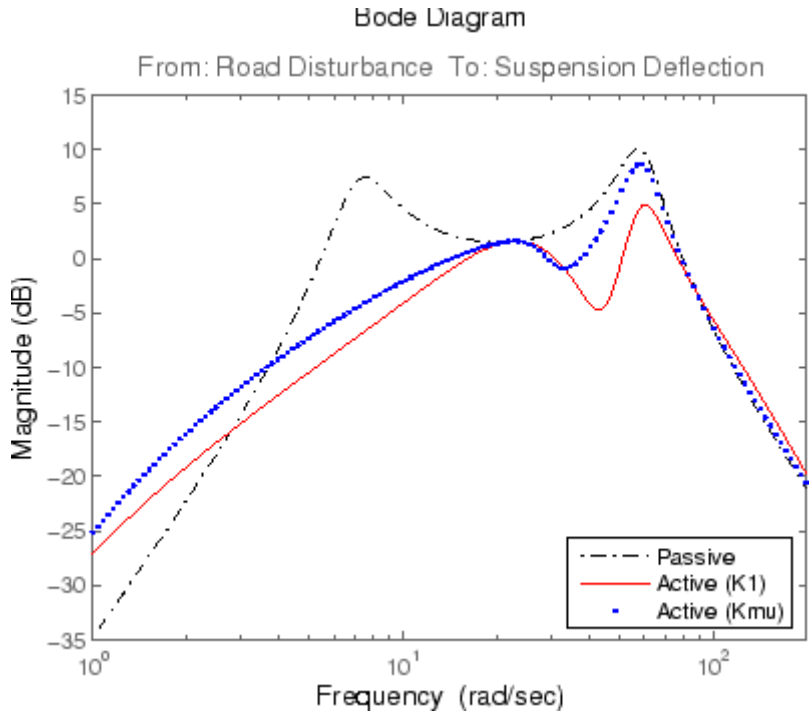
```
systemnames = 'qcar Wn Wref Wact Wx1 actmod';
inputvar = '[ d1; d2; fs ]';
outputvar = '[ Wact; Wx1; qcar(2)+Wn ]';
input_to_actmod = '[ fs ]';
input_to_qcar = '[ Wref; fs]';
input_to_Wn = '[ d2 ]';
input_to_Wref = '[ d1 ]';
input_to_Wact = '[ fs ]';
input_to_Wx1 = '[ qcar(1) ]';
qcaricunc = sysic;
```

A μ-synthesis controller is synthesized using *D-K* iteration with the dksyn command. The *D-K* iteration procedure is an approximation to μ-synthesis that attempts to synthesize a controller that achieves robust performance [1], [10], [11], [12]. There is one control input, the hydraulic actuator force, and one measurement signal, the car body acceleration.

```
[Kdk,CLdk,gdk] = dksyn(qcaricunc,nmeas,ncont);
CLdkunc = lft(qcar([1:4 2],1:2)*blkdiag(1,actmod),Kdk);
sprintf('mu-synthesis controller Kdk achieved a norm of %2.5g',gdk)
ans =
mu-synthesis controller Kdk achieved a norm of 0.53946
```

You can analyze the performance of the μ-synthesis controller by constructing the closed-loop feedback system CLdkunc. Bode magnitude plots of the passive suspension and active suspension systems on the nominal actuator model with H_∞ design 1 and the μ-synthesis controller are shown in the following figure. Note that the μ-synthesis controller better attenuates the first resonant mode at the expense of decreased performance below 3 rad/s.

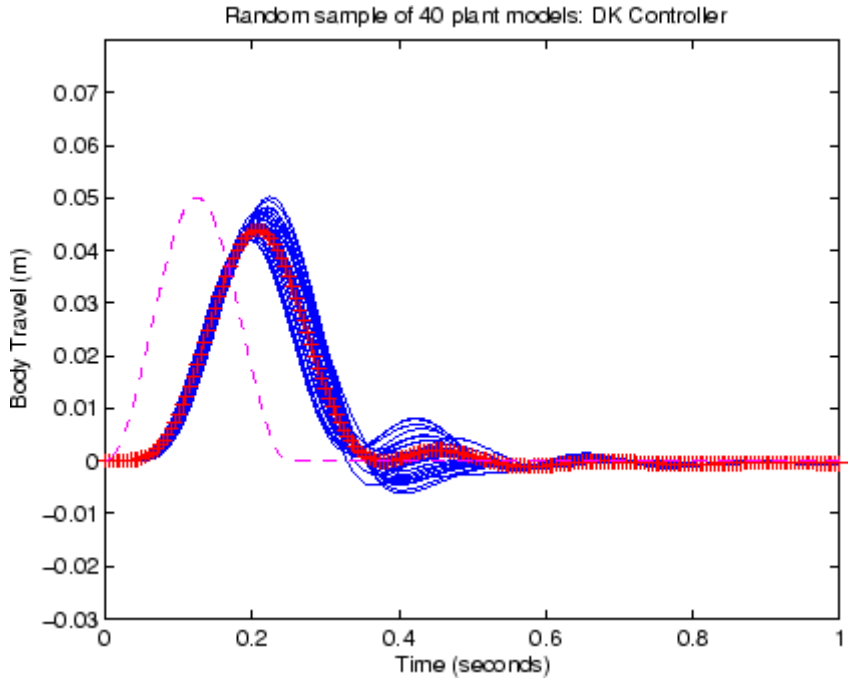
```
bodemag(qcar(3,1),'k-.',CL1(3,1),'r-',CLmuunc.Nominal(3,1),'b.',...
   logspace(0,2.3,140))
```



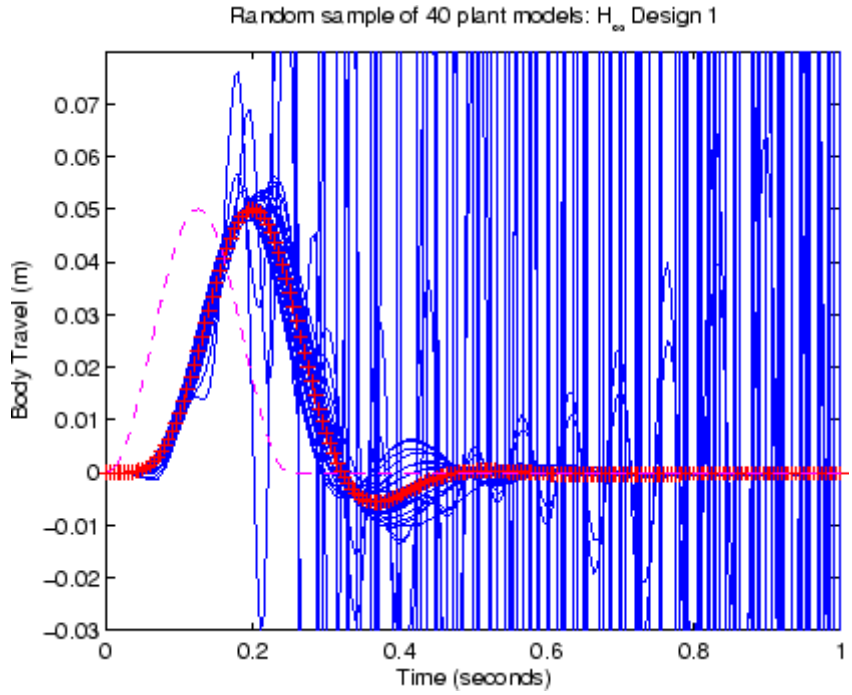
It is important to understand how robust both controllers are in the presence of model error. You can simulate the active suspension system with the H_∞ design 1 and the μ-synthesis controller. The uncertain closed-loop systems, `CL1unc` and `CLdkunc`, are formed with `K1` and `Kdk`, respectively. For each uncertain system, 40 random plant models in the model set are simulated. As you can see, both controllers are robust and perform well in the presence of actuator model error. The μ-synthesis controller `Kdk` achieves slightly better performance than H_∞ design 1.

```
CL1unc = lft(qcar([1:4 2],1:2)*blkdiag(1,actmod),K1);
[CLdkunc40,dksamples] = usample(CLdkunc,40);
CL1unc40 = usubs(CL1unc,dksamples);
nsamp = 40;
for i=1:nsamp
   [ymusamp,t]  = lsim(CLmuunc40(1:4,1,i),roaddist,time);
   plot(t,ymusamp(:,1),'b')
   hold on
```

```
end
[ymusamp,t]  = lsim(CLmuunc.Nominal(1:4,1),roaddist,time);
plot(t,ymusamp(:,1),'r+',t,roaddist,'m--')
```




```
for i=1:nsamp
      [y1samp,t] = lsim(CL1unc40(1:4,1,i),roaddist,time);
      plot(t,y1samp(:,1),'b')
      hold on
   end
   [y1samp,t]  = lsim(CL1unc.Nominal(1:4,1),roaddist,time);
plot(t,y1samp(:,1),'r+',t,roaddist,'m--')
```


Random sample of 40 plant models: H∞ Design 1
Body Travel (m)
Time (seconds)
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
−0.01
−0.02
−0.03
0
0.2
0.4
0.6
0.8
1

Bibliography

[1] Balas, G.J., and A.K. Packard, "The structured singular value μ-framework," CRC Controls Handbook, Section 2.3.6, January, 1996, pp. 671-688.

[2] Ball, J.A., and N. Cohen, "Sensitivity minimization in an H_∞ norm: Parametrization of all suboptimal solutions," *International Journal of Control*, Vol. 46, 1987, pp. 785-816.

[3] Bamieh, B.A., and Pearson, J.B., "A general framework for linear periodic systems with applications to H_∞ sampled-data control," *IEEE Transactions on Automatic Control*, Vol. AC-37, 1992, pp. 418-435.

[4] Doyle, J.C., Glover, K., Khargonekar, P., and Francis, B., "State-space solutions to standard H_2 and H_∞ control problems," *IEEE Transactions on Automatic Control*, Vol. AC-34, No. 8, August 1989, pp. 831-847.

[5] Fialho, I., and Balas, G.J., "Design of nonlinear controllers for active vehicle suspensions using parameter-varying control synthesis," *Vehicle Systems Dynamics*, Vol. 33, No. 5, May 2000, pp. 351-370.

[6] Francis, B.A., *A course in H_∞ control theory*, Lecture Notes in Control and Information Sciences, Vol. 88, Springer-Verlag, Berlin, 1987.

[7] Glover, K., and Doyle, J.C., "State-space formulae for all stabilizing controllers that satisfy an H_∞ norm bound and relations to risk sensitivity," *Systems and Control Letters*, Vol. 11, pp. 167-172, August 1989. *International Journal of Control*, Vol. 39, 1984, pp. 1115-1193.

[8] Hedrick, J.K., and Batsuen, T., "Invariant Properties of Automotive Suspensions," *Proceedings of The Institution of Mechanical Engineers*, 204 (1990), pp. 21-27.

[9] Lin, J., and Kanellakopoulos, I., "Road Adaptive Nonlinear Design of Active Suspensions," *Proceedings of the American Control Conference*, (1997), pp. 714-718.

[10] Packard, A.K., Doyle, J.C., and Balas, G.J., "Linear, multivariable robust control with a μ perspective," *ASME Journal of Dynamics, Measurements and Control: Special Edition on Control,* Vol. 115, No. 2b, June, 1993, pp. 426-438.

[11] Skogestad, S., and Postlethwaite, I., *Multivariable Feedback Control: Analysis & Design*, John Wiley & Sons, 1996.

[12] Stein, G., and Doyle, J., "Beyond singular values and loopshapes," *AIAA Journal of Guidance and Control*, Vol. 14, Num. 1, January, 1991, pp. 5-16.

[13] Zames, G., "Feedback and optimal sensitivity: model reference transformations, multiplicative seminorms, and approximate inverses," *IEEE Transactions on Automatic Control*, Vol. AC-26, 1981, pp. 301-320.

6

Tuning Fixed Control Architectures

Tuning Fixed-Structure Control Systems

In this section...

What is a Fixed-Structure Control System?

Fixed-structure control systems are control systems that have predefined architectures and controller structures. For example,

- A single-loop SISO control architecture where the controller is a fixed-order transfer function, a PID controller, or a PID controller plus a filter.
- A MIMO control architecture where the controller has fixed order and structure. For example, a 2-by-2 decoupling matrix plus two PI controllers is a MIMO controller of fixed order and structure.
- A multiple-loop SISO or MIMO control architecture, including nested or cascading loops, with multiple gains and dynamic components to tune.

You can use `looptune` and `hinfstruct` for frequency-domain tuning of virtually any SISO or MIMO feedback architecture to meet your design requirements. You can use both approaches to tune fixed structure control systems in either MATLAB or Simulink (requires Simulink® Control Design™).

Choosing an Approach to Fixed Control Structure Tuning

Robust Control Toolbox includes two commands for frequency-domain tuning of control systems with fixed control structure: `looptune` (or, for Simulink models, `slTunable.looptune`) and `hinfstruct`. The following table summarizes some differences between the two commands.

Command	Description	Representing Your Design Requirements	More Information
`looptune`	Automatically tunes fixed-structure control systems to meet basic requirements of performance, bandwidth, and robustness. You can also directly specify design requirements such as tracking, gain constraints, or impose a custom loop shape.	Directly express your design requirements such as bandwidth, tracking, maximum gains, and stability margins. `looptune` automatically converts these design requirements into optimization constraints to compute values for the tuned parameters of the control system.	"Tuning a Control System with looptune" on page 6-5
`hinfstruct`	Uses the methods of H_∞ synthesis to tune fixed-structure control systems. Gives you more flexibility than `looptune` in setting up design constraints, but requires some understanding of the theory and methods of H_∞ synthesis.	Using weighting functions, manually convert your design requirements into normalized gain constraints.	"H-Infinity Tuning with hinfstruct" on page 6-19

Difference Between fixed-Structure Tuning and Traditional H-Infinity Synthesis

Both `looptune` and `hinfstruct` tune the controller parameters by optimizing the H_∞ norm across a closed-loop system (see [1]). However, these functions differ in important ways from traditional H_∞ methods.

Traditional H_∞ synthesis (performed using the `hinfsyn` or `loopsyn` commands) designs a full-order, centralized controller. Such controllers often have high-order dynamics. Furthermore, traditional H_∞ synthesis provides no

way to impose structure on the controller. Thus, the results can be difficult to map to your specific real-world control architecture. Additionally, traditional H_∞ synthesis requires you to express all design requirements in terms of a single weighted MIMO transfer function.

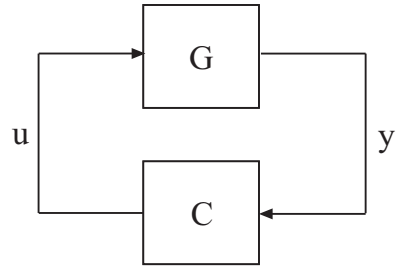
In contrast, structured H_∞ synthesis allows you to describe and tune the specific control system with which you are working. You can specify your control architecture, including the number and configuration of feedback loops, sensor signals, and actuator signals that make up your control system. You can also specify the complexity and structure of each tunable component in your control system, such as PID controllers, gains, and fixed-order transfer functions. Additionally, you can easily combine requirements on separate closed-loop transfer functions.

Tuning a Control System with looptune

In this section...

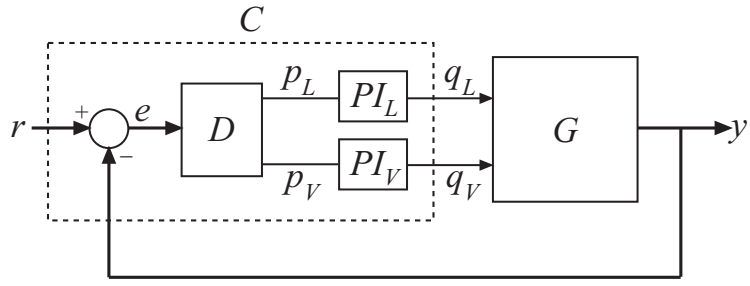
How looptune Sees Your Control System

`looptune` tunes the feedback loop illustrated below to meet default requirements or requirements that you specify.



`C` represents the controller and `G` represents the plant. The sensor outputs `y` (measurements) and actuator outputs `u` (controls) define the boundary between plant and controller. The plant is the portion of your control system whose inputs are controls, and outputs are measurements. Conversely, the controller is the remainder—the portion of your control system that receives measurements as inputs, and produces control signals as outputs.

For example, in the control system of the following illustration, the controller `C` receives the measurement `y`, and the reference signal `r`. The controller produces the controls `qL` and `qV` as outputs.



The controller `C` has a fixed internal structure. `C` includes a gain matrix `D` , the PI controllers `PI_L` and `PI_V`, and a summing junction. The `looptune` command tunes free parameters of `C` such as the gains in `D` and the proportional and integral gains of `PI_L` and `PI_V`. You can also use `looptune` to co-tune free parameters in both `C` and `G`.

Setting Up Your Control System in MATLAB

To set up your control system in MATLAB for tuning with `looptune`:

1. Create a tunable `genss` model representing the controller.
2. Create a Numeric LTI model representing the plant. For co-tuning the plant and controller, represent the plant as a tunable `genss` model.

Related Examples

- "Tune a MIMO Control System for a Specified Bandwidth" on page 6-8
- "Tune a Two-Loop Control System to Meet Specific Design Requirements" on page 6-13

Setting Up Your Control System in Simulink

To set up your control system in Simulink for tuning with `slTunable.looptune` (requires Simulink Control Design software):

1. Use `slTunable` to create an interface to the Simulink model of your control system. When you create the interface, you specify which blocks to tune in your model.

2. Use `slTunable.addControl` and `slTunable.addMeasurement` to specify the control and measurement signals that define the boundaries between plant and controller.

The `slTunable` interface automatically linearizes your Simulink model. The `slTunable` interface also automatically parametrizes the blocks that you specify as tunable blocks. For more information, see the `slTunable` reference page and "Supported Blocks for Tuning in Simulink" on page 6-32.

Related Examples

- "Tune a MIMO Control System in Simulink" on page 6-17

Performance and Robustness Specifications

By default, `looptune` tunes your control system to meet the following default requirements:

- Performance—integral action at low frequency
- Bandwidth—gain crossover for each loop falls in the frequency interval you specify
- Robustness—adequate stability margins and gain roll-off at frequencies above the crossover frequency

You can define custom performance specifications and robustness criteria that further constrain or override the default requirements. The following table summarizes the types of performance specifications and robustness criteria that you can specify.

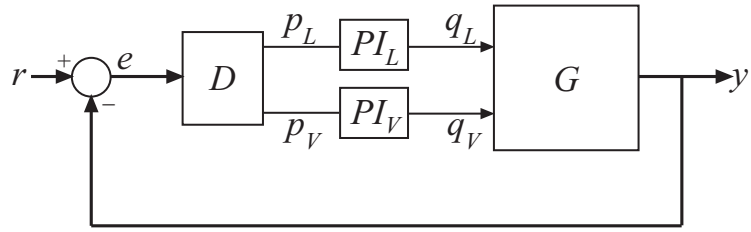
Specification Type	Description	How To Specify
Crossover frequency band	Frequency band into which gain crossover frequencies of all open-loop responses fall	`wc` argument to `looptune` or `slTunable.looptune`
Stability margins	Gain and phase margins, defined as multi-loop disk margins for MIMO systems (see `loopmargin`), Defaults are 7.6 dB gain margin, 45 degrees phase margin	`looptuneOptions` option set passed to `looptune` or `slTunable.looptune`
Custom design requirements	Constraints on the response across particular input/output pairs, such as: • Reference tracking • Maximum gain (for example, for disturbance rejection) • Custom performance or roll-off specification	`TuningGoal` requirement objects passed to `looptune`: • `TuningGoal.Tracking` • `TuningGoal.MaxGain` • `TuningGoal.LoopShape`

Related Examples

- "Tune a MIMO Control System for a Specified Bandwidth" on page 6-8
- "Tune a Two-Loop Control System to Meet Specific Design Requirements" on page 6-13
- "Tune a MIMO Control System in Simulink" on page 6-17

Tune a MIMO Control System for a Specified Bandwidth

This example shows how to tune the control system of the following illustration to achieve crossover between 0.1 and 1 rad/min.



The setpoint signal `r`, the error signal `e`, and the output signal `y` are vector-valued signals of dimension 2. The 2-by-2 plant `G` is represented by:

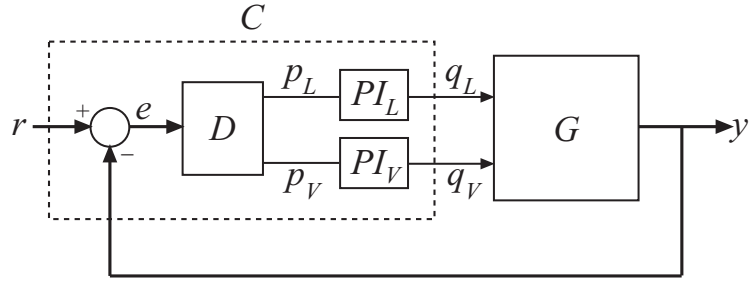
$$G(s) = \frac{1}{75s+1}\begin{bmatrix} 87.8 & -86.4 \\ 108.2 & -109.6 \end{bmatrix}.$$

Create a Numeric LTI model representing the plant.

```
s = tf('s');
G = 1/(75*s+1)*[87.8 -86.4; 108.2 -109.6];
G.InputName = {'qL','qV'};
G.OutputName = 'y';
```

Name the inputs and outputs of your model using the `InputName` and `OutputName` properties. Doing so tells `looptune` how to interconnect your plant and controller.

Define the controller `C`, as the remainder of the control system.



`C` receives the measurement signal `y` and the reference `r` as inputs. `C` produces the control signals `qL` and `qV`.

The controller therefore has a fixed structure that includes four components: the 2-by-2 decoupling matrix `D`, the two PI controllers `PI_L` and `PI_V`, and a summing junction. (For more information about this control system, see the demo Decoupling Controller for a Distillation Column.)

Use Control Design Blocks to represent the components of the controller. Connect the components using model interconnection commands.

```
D = ltiblock.gain('Decoupler',eye(2));
PI_L = ltiblock.pid('PI_L','pi');
PI_L.OutputName = 'qL';
PI_V = ltiblock.pid('PI_V','pi');
PI_V.OutputName = 'qV';

sum = sumblk('e = r - y',2);
C0 = (blkdiag(PI_L,PI_V)*D)*sum;
```

The `genss` model `C0` represents the controller structure. `C0` specifies which controller parameters are tunable, and contains the initial values of those parameters.

Tune the control system.

```
rng('default');
wc = [0.1,1];
options = looptuneOptions('RandomStart',5);
[G,C,gam,Info] = looptune(G,C0,wc,options);
```

These commands produce the result:

```
Final: Peak gain = 1.24, Iterations = 77
Final: Peak gain = 1.24, Iterations = 110
Final: Peak gain = 3.59, Iterations = 35
       Spectral abscissa -1.92e-007 is close to bound -1e-007
Final: Peak gain = 1.24, Iterations = 86
Final: Peak gain = 1.24, Iterations = 134
Final: Peak gain = 1.24, Iterations = 89
```

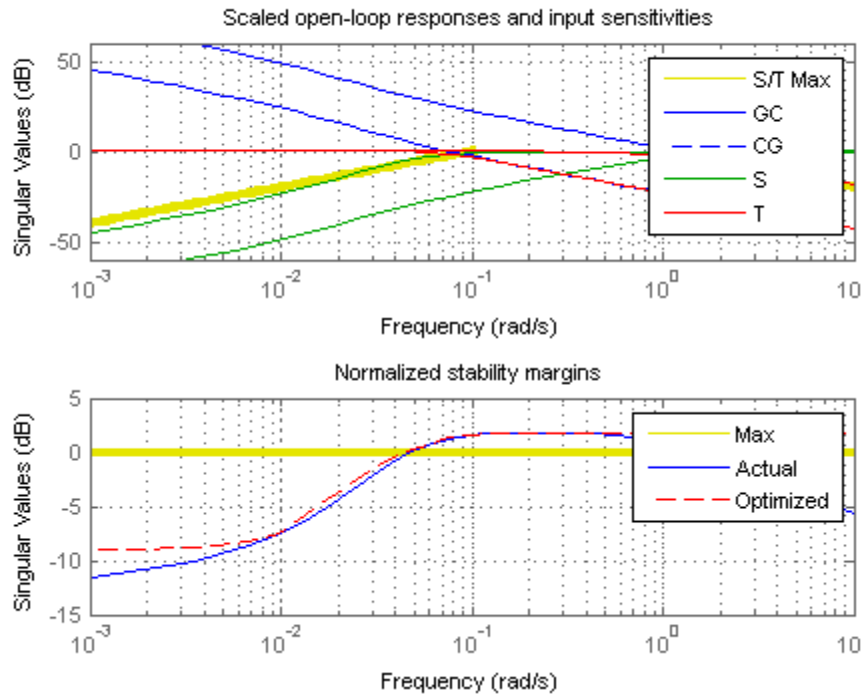
The `looptune` algorithm uses the input and output names of `C0` and `G` to form the closed loop system for tuning. The input `wc` specifies that the crossover frequency of each loop in the system falls between 0.1 and 1 rad/min.

`looptune` returns `C`, the tuned version of `C0`.

The output `gam` is a normalized parameter indicating the degree of success meeting the tuning requirements. `gam <=1` indicates that all requirements are satisfied. A `gam` value much greater than one indicates failure to meet some requirement. In this example, the best value of `gam = 1.24` indicates a reasonable degree of success meeting the specified crossover requirement.

Validate the frequency domain response of the tuned result using `loopview`.

```
loopview(G,C,Info)
```



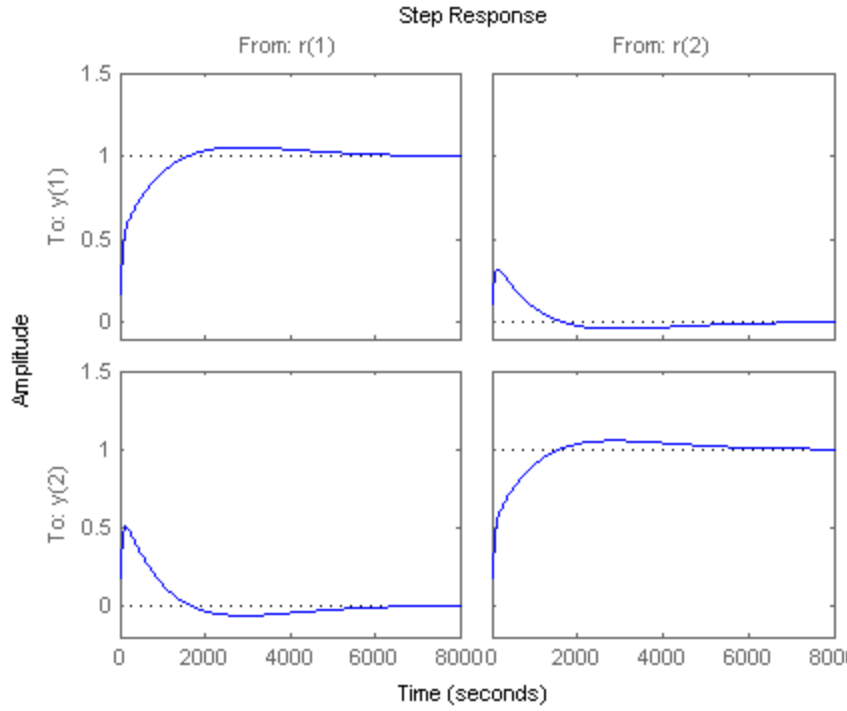
The first plot shows that the open-loop gain crossovers fall close to the specified interval `[0.1,1]`. This plot also includes the maximum and tuned values of the sensitivity function `S = inv(1-G*C)` and complementary sensitivity `T = 1-S`. The curve marked `S/T Max` shows the maximum allowed

`S` on the low-frequency side of the plot, and the maximum allowed `T` on the high-frequency side. These curves are the constraints that `looptune` imposes on `S` and `T` to enforce the target crossover range `wc`.

The second plot shows that the MIMO stability margins of the tuned system (blue curve) do not significantly exceed the upper limit (yellow curve).

Validate the time domain response using Control System Toolbox analysis commands such as `step`.

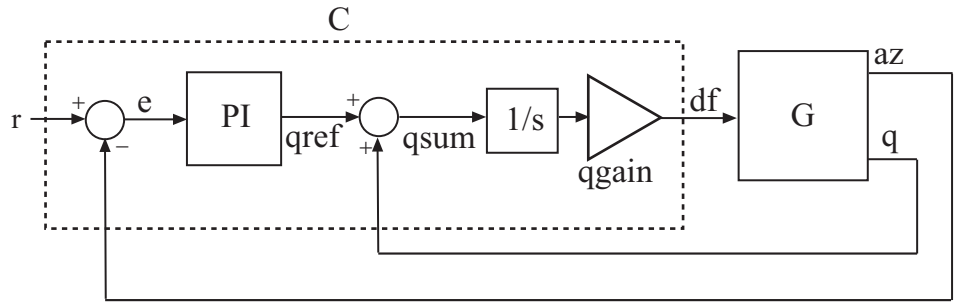
```
T = connect(G,C,'r','y');
step(T)
```



The tuned controller provides good setpoint tracking with reasonably small crosstalk between channels. To impose further constraints on crosstalk, use the `TuningGoal.MaxGain` requirements object.

Tune a Two-Loop Control System to Meet Specific Design Requirements

This example shows how to tune the control system of the following illustration to make the signal `az` track the reference signal `r` with a 1 s response time.



Load the plant.

```
load looptune_ex G
G.InputName = 'df';
G.OutputName = {'az','q'};
```

`G` is a 1-input, 2-output state-space model. Naming the inputs and outputs of `G` tells the software how to connect the plant and controller when tuning the control system.

The controller `C` is the portion of the control system that receives the measurement signals `q` and `az`, and produces the control signal `df`. Create a tunable model of the controller using Control Design Blocks, summing junctions, and `connect`.

```
PI = ltiblock.pid('PI','pi');
PI.InputName = 'e'; PI.OutputName = 'qref';

qgain = ltiblock.gain('qgain',0);
Cdf = qgain*tf(1,[1 0]);
Cdf.InputName = 'qsum'; Cdf.OutputName = 'df';

sum1 = sumblk('e = r - az');
```

```
sum2 = sumblk('qsum = qref + q');

CO = connect(PI,Cdf,sum1,sum2,{'r','az','q'},'df');
```

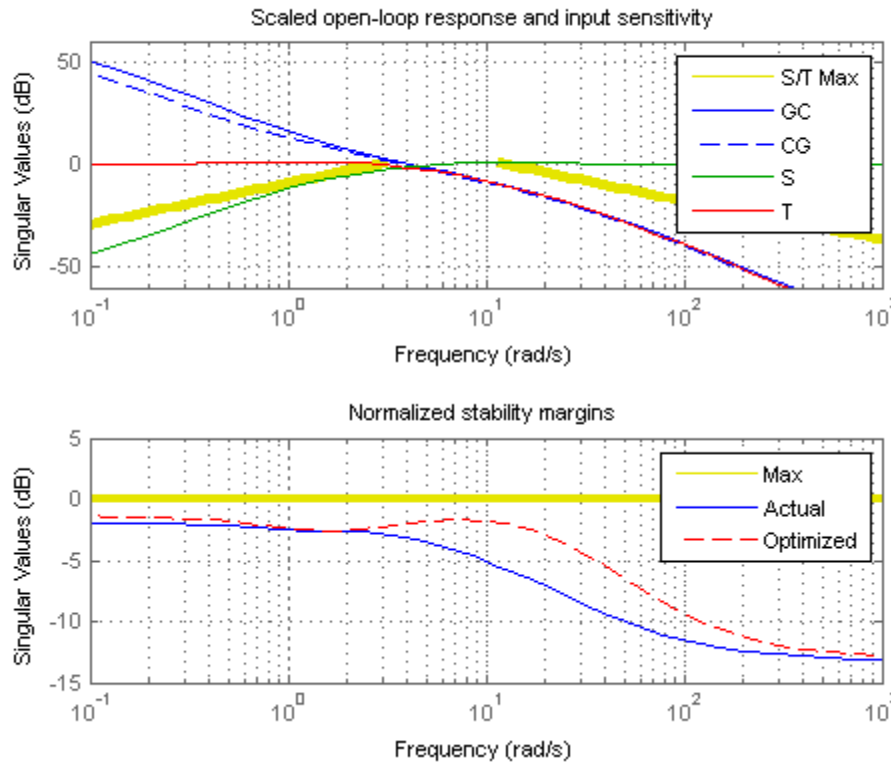
`CO` is a `genss` model. The tunable parameters of `CO` are the proportional and integral gains of the PI controller `PI`, and the gain of `qgain`.

Tune the control system. Use a target bandwidth range of [3,12] to get a response time on the order of 1 s.

```
rng('default');
Options = looptuneOptions('RandomStart',4);
[G,C,gam,Info] = looptune(G,CO,[3,12],Options);
```

Examine the frequency-domain performance of the tuned controller `C`.

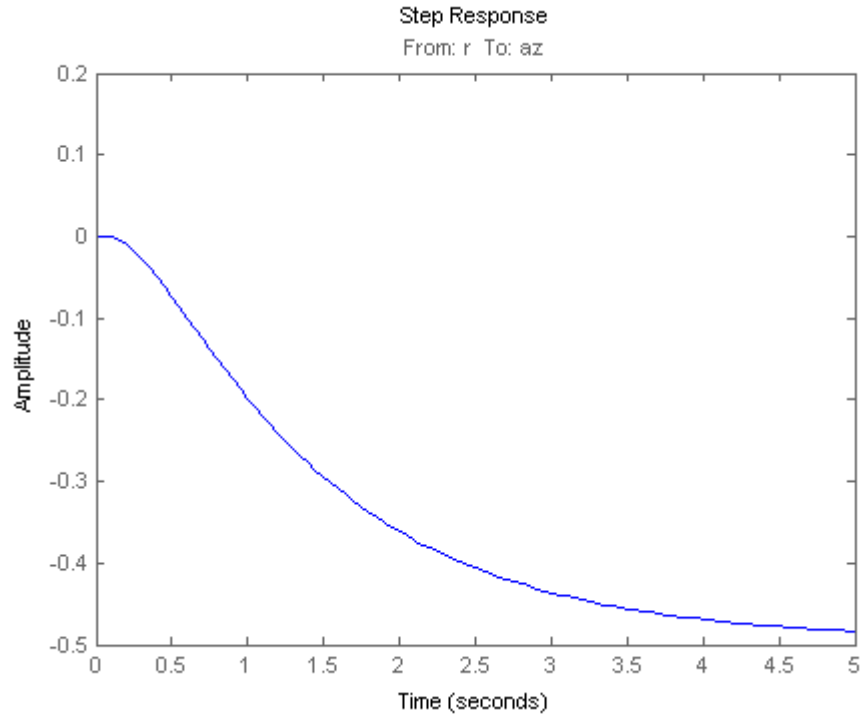
```
loopview(G,C,Info)
```



The crossover frequency is at approximately 4 rad/s. The stability margins do not exceed the target margins.

Examine the closed-loop time-domain response of the tuned control system to see how well `az` tracks the reference signal `r`. Use the tuned controller `C` to form the closed-loop response. `C` is a `genss` model whose current parameter values are the tuned values.

```
T = connect(G,C,'r','az');
step(T,5)
```



In the time domain, it is clear that the output does not track the reference signal, despite the presence of an integrator in the loop. This result occurs because the feedback loop acts on both signals `az` and `q`. The controller does not know what combination of `az` and `q` should track the reference `r`.

Use a `TuningGoal.Tracking` requirement object to add the tracking constraint to the optimization performed by `looptune`.

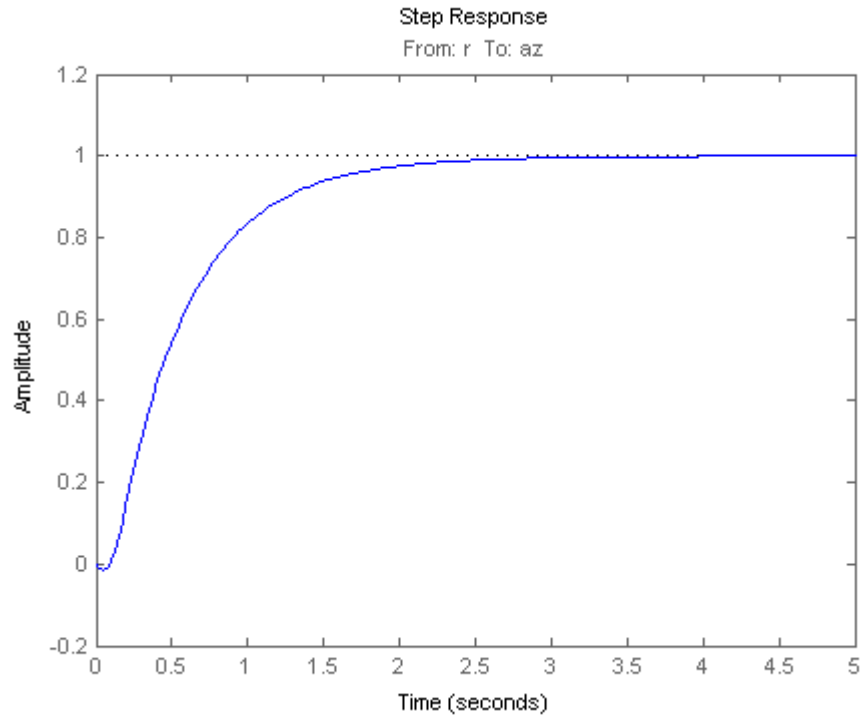
```
Req = TuningGoal.Tracking('r','az',1);
```

Tune the control system again using the tracking requirement.

```
rng('default');
[G,C,gam,Info] = looptune(G,C0,[3,12],Req,Options);
```

Validate the new design.

```
T = connect(G,C,'r','az');
step(T,5)
```



The response now meets the tuning design requirements of good reference tracking with a response time of about 1 s.

Tune a MIMO Control System in Simulink

The following demos show how to use the `slTunable` interface to tune a Simulink model with `looptune`. Running these demos requires Simulink Control Design.

- Tuning of a Digital Motion Control System
- Tuning of a Two-Loop Autopilot

- Tuning of Cascaded PID Loops

H-Infinity Tuning with hinfstruct

What is hinfstruct?

`hinfstruct` lets you use the frequency-domain methods of H_∞ synthesis to tune control systems that have predefined architectures and controller structures. To use `hinfstruct`, you describe your control system as a Generalized LTI model that keeps track of the tunable components of your system. `hinfstruct` tunes those parameters by minimizing the closed-loop gain from the system inputs to the system outputs (the H_∞ norm). The methodology and algorithm behind `hinfstruct` are described in [1].

Structured H-Infinity Synthesis Workflow

Performing structured H_∞ synthesis requires the following steps:

1. Formulate your design requirements as H_∞ constraints, which are constraints on the closed-loop gains from specific system inputs to specific system outputs.
2. Build tunable models of the closed-loop transfer functions of Step 1.
3. Tune the control system using `hinfstruct`.
4. Validate the tuned control system.

Formulating Design Requirements as H-Infinity Constraints

Control design requirements are typically performance measures such as response speed, control bandwidth, roll-off, and steady-state error. To use `hinfstruct`, first express the design requirements as constraints on the closed-loop gain.

You can formulate design requirements in terms of the closed-loop gain using loop shaping. *Loop shaping* is a common systematic technique for defining control design requirements for H_∞ synthesis. In loop shaping, you first express design requirements as open-loop gain requirements.

For example, a requirement of good reference tracking and disturbance rejection is equivalent to high (>1) open-loop gain at low frequency. A requirement of insensitivity to measurement noise or modeling error is equivalent to a low (<1) open-loop gain at high frequency. You can then convert these open-loop requirements to constraints on the closed-loop gain using weighting functions.

This formulation of design requirements results in a H_∞ constraint of the form:

$$\left\| H(s) \right\|_\infty < 1,$$

where $H(s)$ is a closed-loop transfer function that aggregates and normalizes the various requirements.

For examples of how to formulate design requirements for H_∞ synthesis using loop shaping, see the following demos:

- Loop Shaping Design with HINFSTRUCT
- Tuning of a Two-Loop Autopilot

For more information about constructing weighting functions from design requirements, see “H-Infinity Performance” on page 5-9.

Building a Tunable Closed-Loop Model

In the previous step you expressed your design requirements as a constraint on the H_∞ of a closed-loop transfer function $H(s)$.

The next step is to create a Generalized LTI model of your closed-loop control system that includes all of the fixed and tunable elements of the control system. The model also includes any weighting functions that represent your design requirements. There are two ways to obtain this tunable model of your control system:

- Construct the model using Control System Toolbox commands.
- Obtain the model from a Simulink model using Simulink Control Design commands.

Constructing the Closed-Loop System Using Control System Toolbox Commands

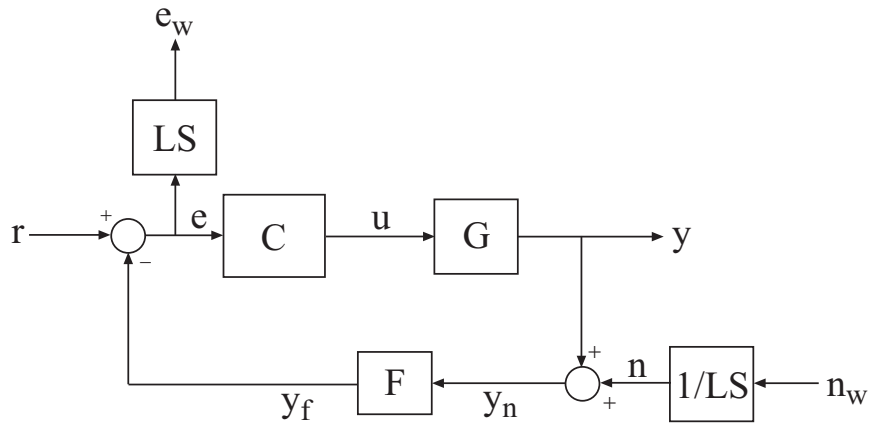
To construct the tunable generalized linear model of your closed-loop control system in MATLAB:

1. Use commands such as `tf`, `zpk`, and `ss` to create numeric linear models that represent the fixed elements of your control system and any weighting functions that represent your design requirements.

2. Use tunable models (either Control Design Blocks or Generalized LTI models) to model the tunable elements of your control system. For more information about tunable models, see "Models with Tunable Coefficients" in the *Control System Toolbox User's Guide*.

3. Use model-interconnection commands such as `series`, `parallel`, and `connect` to construct your closed-loop system from the numeric and tunable models.

Example: Modeling a Control System With a Tunable PI Controller and Tunable Filter. This example shows how to construct a tunable generalized linear model of the following control system for tuning with `hinfstruct`.



This block diagram represents a head-disk assembly (HDA) in a hard disk drive. The architecture includes the plant `G` in a feedback loop with a PI controller `C` and a low-pass filter, `F = a/(s+a)`. The tunable parameters are the PI gains of `C` and the filter parameter `a`.

The block diagram also includes the weighting functions `LS` and `1/LS`, which express the loop-shaping requirements. Let $T(s)$ denote the closed-loop transfer function from inputs (r,n_w) to outputs (y,e_w). Then, the H_∞ constraint:

$$\left\| T(s) \right\|_\infty < 1$$

approximately enforces the target loop shape `LS`.

For more detailed discussion of the HDA system and the H_∞ formulation of this example, see the demo Loop Shaping Design with HINFSTRUCT.

To tune this control system with `hinfstruct`, you construct a generalized model of the closed-loop system $T(s)$, including the weighting functions, as follows.

1 Load the plant `G` from a saved file.

```
load hinfstruct_demo G
```

`G` is a 9th-order SISO state-space (`ss`) model.

2 Create a tunable model of the PI controller.

You can use the predefined Control Design Block `ltiblock.pid` to represent a tunable PI controller.

```
C = ltiblock.pid('C','pi');
```

3 Create a tunable model of the low-pass filter.

Because there is no predefined Control Design Block for the filter `F = a/(s+a)`, use `realp` to represent the tunable filter parameter `a`. Then create a tunable `genss` model representing the filter.

```
a = realp('a',1);
F = tf(a,[1 a]);
```

4 Create a numeric LTI model representing the weighting function `LC`.

```
wc = 1000;
s = tf('s');
LS = (1+0.001*s/wc)/(0.001+s/wc);
```

5 Label the inputs and outputs of all the components of the control system.

Labeling the I/Os allows you to connect the elements to build the closed-loop system $T(s)$.

```
Wn = 1/LS;  Wn.InputName = 'nw';  Wn.OutputName = 'n';
We = LS;  We.InputName = 'e';   We.OutputName = 'ew';
C.InputName = 'e';  C.OutputName = 'u';
F.InputName = 'yn';  F.OutputName = 'yf';
```

6 Use the I/O labels to specify the summing junctions.

```
Sum1 = sumblk('e = r - yf');
Sum2 = sumblk('yn = y + n');
```

7 Use `connect` to combine all the elements into a complete model of the closed-loop system $T(s)$.

```
TO = connect(G,Wn,We,C,F,Sum1,Sum2,{'r','nw'},{'y','ew'});
```

(See in the *Control System Toolbox User's Guide* for more information about connecting models.)

`TO` is a `genss` object, which is a Generalized LTI model representing the closed-loop control system with weighting functions. The `Blocks` property of `TO` contains the tunable blocks `C` and `a`.

```
TO.Blocks

ans =

    C: [1x1 ltiblock.pid]
    a: [1x1 realp]
```

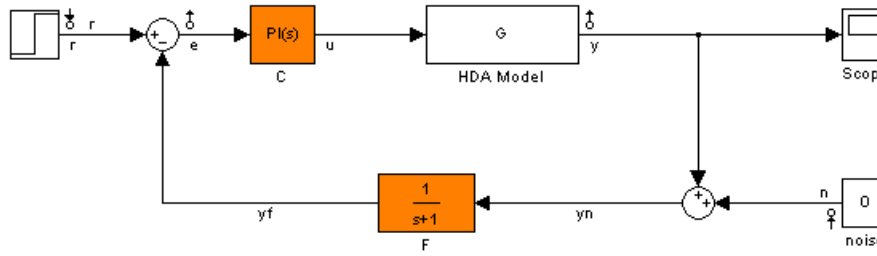
For more information about generalized models of control systems that include both numeric and tunable components, see "Models with Tunable Coefficients" in the *Control System Toolbox User's Guide*.

You can now use `hinfstruct` to tune the parameters of this control system. See "Tuning the Controller Parameters" on page 6-26.

Constructing the Closed-Loop System Using Simulink Control Design Commands

If you have a Simulink model of your control system and Simulink Control Design software, you can use the `linlft` to build a generalized linear model of your control system. To do this, use `linlft` to extract a numerical linear model of the fixed (non-tunable) portion of your control system. Then, use `lft` to combine the extracted model with the tunable blocks.

Example: Creating a Weighted Tunable Model of Control System Starting From a Simulink Model. This example shows how to construct a tunable generalized linear model of the control system in the Simulink model `rct_diskdrive`. For a more detailed discussion of this example see the demo Loop Shaping Design with HINFSTRUCT.



To create a generalized linear model of this control system (including loop-shaping weighting functions):

1 Open the model.

```
open('rct_diskdrive');
```

2 Create an `slTunable` interface to the model. The interface allows you to specify the tunable blocks and extract linearized open-loop and closed-loop responses. (For more information about the interface, see the `slTunable` reference page.)

```
STO = slTunable('rct_diskdrive',{'C','F'});
```

This command specifies that `C` and `F` are the tunable blocks in the model. The `slTunable` interface automatically parametrizes these blocks. The default parametrization of the transfer function block `F` is a transfer function with two free parameters. Because `F` is a low-pass filter, you must constrain its coefficients. To do so, specify a custom parameterization of `F`.

```
a = realp('a',1);    % filter coefficient
setBlockParam(STO,'F',tf(a,[1 a]));
```

3 Extract a tunable model of the closed-loop transfer function you want to tune.

```
TO = getIOTransfer(STO,{'r','n'},{'y','e'});
```

This command returns a `genss` model of the linearized closed-loop transfer function from the reference and noise inputs `r,n` to the measurement and error outputs `y,e`. The error output is needed for the loop-shaping weighting function. For more information about the weighting functions, these weighting functions, see the demo Loop Shaping Design with HINFSTRUCT.

4 Define the loop-shaping weighting functions and append them to `T0`.

```
wc = 1000;
s = tf('s');
LS = (1+0.001*s/wc)/(0.001+s/wc);

T0 = blkdiag(1,LS) * T0 * blkdiag(1,1/LS);
```

The generalized linear model `T0` is a tunable model of the closed-loop transfer function $T(s)$, discussed in "Example: Modeling a Control System With a Tunable PI Controller and Tunable Filter" on page 6-21. $T(s)$ is a weighted closed-loop model of the control system of `rct_diskdrive`. Tuning `T0` to enforce the H_∞ constraint

$$\left\| T(s) \right\|_\infty < 1$$

approximately enforces the target loop shape `LS`.

You can now use `hinfstruct` to tune the parameters of this control system. See "Tuning the Controller Parameters" on page 6-26.

Tuning the Controller Parameters

After you obtain the `genss` model representing your control system, use `hinfstruct` to tune the tunable parameters in the `genss` model .

`hinfstruct` takes a tunable linear model as its input.

For example, you can tune controller parameters for the example discussed in "Building a Tunable Closed-Loop Model" on page 6-20 using the following command:

```
[T,gamma,info] = hinfstruct(T0);
```

This command returns the following outputs:

- `T`, a `genss` model object. The `Blocks` property of `T` contains the tunable blocks `C` and `a` with parameter values optimized to minimize the closed loop gain across `T`.
- `gamma`, the minimum peak closed-loop gain of `T` achieved by `hinfstruct`.
- `info`, a structure containing additional information about the minimization runs.

Interpreting the Outputs of hinfstruct

Output Model is Tuned Version of Input Model

`T.Blocks` contains tunable Control Design Blocks (or tunable parameters) of the same types as the blocks in the input model, `T0`. For example, `T.Blocks.C` is a modified version of `T0.Blocks.C` that contains the tuned PI controller parameters.

`T` contains the tuned values of all tunable parameters in `T0`.

You can use `getValue` to evaluate any tunable model that depends on the same parameters. For example, the following command:

```
Ctuned = getValue(C,T);
```

evaluates the controller `C` for the tuned parameter values in `T`. `Ctuned` is a `pid` controller object that represents the tuned controller.

For tuning Simulink models, you can use `slTunable.setBlockValue` to propagate the tuned parameter values to the `slTunable` interface. For example:

```
setBlockValue(ST0,T);
```

This command sets the parameter values in `ST0` to the tuned values of `T`.

Use `slTunable.getBlockValue` to extract tuned values of the tunable blocks. For example:

```
Ctuned = getBlockValue(ST0,'C");
```

You can also use the `slTunable` interface for further analysis. For example:

```
step(getIOTransfer(STO,'r','y'));
```

`slTunable.getIOTransfer` extracts a closed-loop response from the `slTunable` interface.

Interpreting gamma

`gamma` is the smallest H_∞ norm achieved by the optimizer. Examine `gamma` to determine how close the tuned system is to meeting your design constraints. For the loop-shaping example of "Building a Tunable Closed-Loop Model" on page 6-20, a typical design constraint is `gamma < 1`.

The value of `gamma` that `hinfstruct` returns is a local minimum of the gain minimization problem. For best results, use the `RandomStart` option to `hinfstruct` to obtain several minimization runs. Setting `RandomStart` to an integer `N > 0` causes `hinfstruct` to run the optimization `N` additional times, beginning from parameter values it chooses randomly. For example:

```
opts = hinfstructOptions('RandomStart',5);
[T,gamma,info] = hinfstruct(TO,opts);
```

You can examine `gamma` for each run to identify an optimization result that meets your design requirements.

For more details about `hinfstruct`, its options, and its outputs, see the `hinfstruct` and `hinfstructOptions` reference pages.

Validating the Controller Design

To validate the `hinfstruct` control design, analyze the tuned output models described in "Interpreting the Outputs of hinfstruct" on page 6-27. Use these tuned models to examine the performance of the tuned system.

Validating the Design in MATLAB

This example shows how to obtain the closed-loop step response of a system tuned with `hinfstruct` in MATLAB.

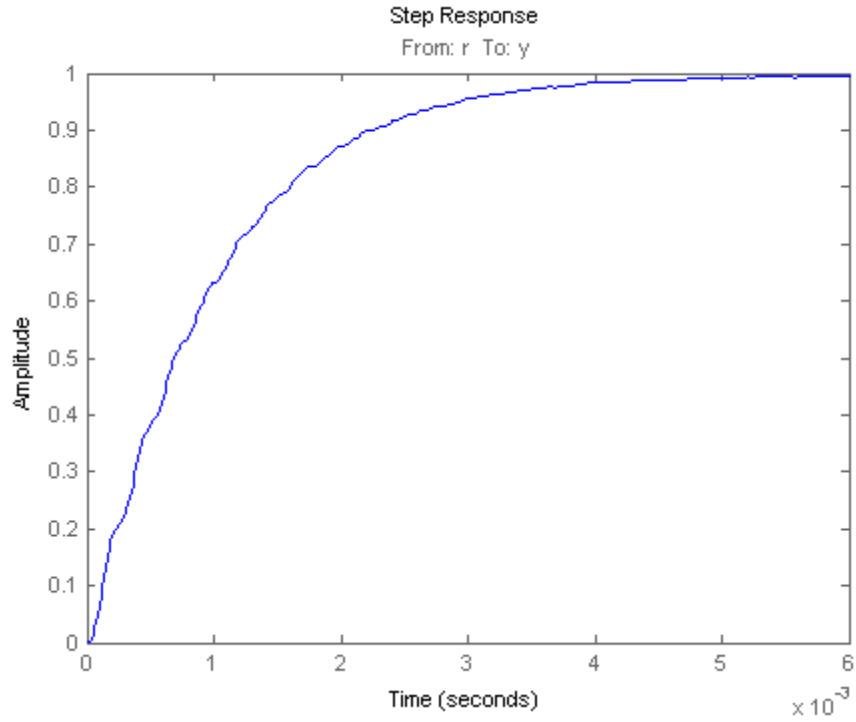
You can use the tuned versions of the tunable components of your system to build closed-loop or open-loop numeric LTI models of the tuned control system. You can then analyze open-loop or closed-loop performance using other Control System Toolbox tools.

In this example, create and analyze a closed-loop model of the HDA system tuned in "Tuning the Controller Parameters" on page 6-26. To do so, extract the tuned controller values from the tuned closed-loop model.

```
Ctuned = getValue(C,T);
Ftuned = getValue(F,T);
```

Then construct the tuned closed-loop response of the control system from `r` to `y`.

```
Try = connect(G,Ctuned,Ftuned,Sum1,Sum2,'r','y');
step(Ttry)
```



Validating the Design in Simulink

This example shows how to write tuned values to your Simulink model for validation.

The `slTunable` interface linearizes your Simulink model. As a best practice, validate the tuned parameters in your nonlinear model. You can use the `slTunable` interface to do so.

In this example, write tuned parameters to the `rct_diskdrive` system tuned in "Tuning the Controller Parameters" on page 6-26.

Make a copy of the `slTunable` interface to preserve the current parameter values.

```
ST = copy(ST0);
```

Update the interface `ST` with the tuned parameter values obtained from `hinfstruct`.

```
setBlockValue(ST,T)
```

This command writes the parameter values from the tuned closed-loop model `T` to the corresponding parameters in the interface `ST`.

Write the parameter values from the `slTunable` interface to the Simulink model.

```
writeBlockValue(ST)
```

You can now simulate the full nonlinear Simulink model using the tuned parameter values to validate the controller design.

Application Examples

For additional examples of performing structured H_∞ synthesis on several different types of control systems, see the following Robust Control Toolbox demos:

- Decoupling Controller for a Distillation Column
- Loop Shaping Design with HINFSTRUCT
- Fixed-Structure Autopilot for a Passenger Jet

Supported Blocks for Tuning in Simulink

`slTunable` automatically assigns a parameterization to supported Simulink blocks. You can write tuned parameter values back to the Simulink model using `slTunable.writeBlockValue`.

Supported blocks include the following:

Supported Blocks	Simulink Library
Gain	Math Operations
LTI System	Control System Toolbox
Discrete Filter	Discrete
PID Controller (one-degree-of-freedom only)	• Continuous • Discrete
State-space blocks	• Continuous • Discrete • Simulink Extras Additional Linear
Zero-pole blocks	• Continuous • Discrete
Transfer function blocks	• Continuous • Discrete

LTI blocks, state-space blocks, transfer function blocks, and zero-pole blocks discretized using the Model Discretizer are also supported.

Tuning Unsupported Blocks

You can specify unsupported blocks as tuned blocks in the `slTunable` interface. `slTunable` assigns a state-space parametrization to such blocks based upon the block linearization. Use `slTunable.getBlockParam` to examine the parametrization of a block.

You cannot write parameter values to an unsupported block using `slTunable.writeBlockValue`.

Bibliography

[1] P. Apkarian and D. Noll, "Nonsmooth H-infinity Synthesis," *IEEE Transactions on Automatic Control*, Vol. 51, Number 1, 2006, pp. 71-86.

A

Examples

Use this list to find examples in the documentation.

Getting Started

Index

Symbols and Numerics

A

B

C

D

E

F

G

H

I

U

W