NI MATRIXx™

Xmath™ Robust Control Module

NATIONAL INSTRUMENTS™

Worldwide Technical Support and Product Information

`ni.com`

National Instruments Corporate Headquarters

11500 North Mopac Expressway   Austin, Texas 78759-3504   USA   Tel: 512 683 0100

Worldwide Offices

Australia 1800 300 800, Austria 43 662 457990-0, Belgium 32 (0) 2 757 0020, Brazil 55 11 3262 3599,
Canada 800 433 3488, China 86 21 5050 9800, Czech Republic 420 224 235 774, Denmark 45 45 76 26 00,
Finland 385 (0) 9 725 72511, France 33 (0) 1 48 14 24 24, Germany 49 89 7413130, India 91 80 41190000,
Israel 972 3 6393737, Italy 39 02 413091, Japan 81 3 5472 2970, Korea 82 02 3451 3400,
Lebanon 961 (0) 1 33 28 28, Malaysia 1800 887710, Mexico 01 800 010 0793, Netherlands 31 (0) 348 433 466,
New Zealand 0800 553 322, Norway 47 (0) 66 90 76 60, Poland 48 22 3390150, Portugal 351 210 311 210,
Russia 7 495 783 6851, Singapore 1800 226 5886, Slovenia 386 3 425 42 00, South Africa 27 0 11 805 8197,
Spain 34 91 640 0085, Sweden 46 (0) 8 587 895 00, Switzerland 41 56 2005151, Taiwan 886 02 2377 2222,
Thailand 662 278 6777, Turkey 90 212 279 3031, United Kingdom 44 (0) 1635 523545

For further support information, refer to the *Technical Support and Professional Services* appendix. To comment
on National Instruments documentation, refer to the National Instruments Web site at `ni.com/info` and enter
the info code `feedback`.

Important Information

Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this document is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

Copyright

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

National Instruments respects the intellectual property of others, and we ask our users to do the same. NI software is protected by copyright and other intellectual property laws. Where NI software may be used to reproduce software or other materials belonging to others, you may use NI software only to reproduce materials that you may reproduce in accordance with the terms of any applicable license or other legal restriction.

Trademarks

MATRIXx™, National Instruments™, NI™, ni.com™, and Xmath™ are trademarks of National Instruments Corporation. Refer to the *Terms of Use* section on ni.com/legal for more information about National Instruments trademarks.

Other product and company names mentioned herein are trademarks or trade names of their respective companies.

Members of the National Instruments Alliance Partner Program are business entities independent from National Instruments and have no agency, partnership, or joint-venture relationship with National Instruments.

Patents

For patents covering National Instruments products, refer to the appropriate location: **Help»Patents** in your software, the patents.txt file on your CD, or ni.com/patents.

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.
(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

Conventions

The following conventions are used in this manual:

[ ]   Square brackets enclose optional items—for example, [`response`].

»   The **»** symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.

          This icon denotes a note, which alerts you to important information.

bold   Bold text denotes items that you must select or click in the software, such as menu items and dialog box options. Bold text also denotes parameter names.

italic   Italic text denotes variables, emphasis, a cross-reference, or an introduction to a key concept. Italic text also denotes text that is a placeholder for a word or value that you must supply.

`monospace`   Text in this font denotes text or characters that you should enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames, and extensions.

`monospace bold`   Bold text in this font denotes the messages and responses that the computer automatically prints to the screen. This font also emphasizes lines of code that are different from the other examples.

Contents

Chapter 1
Introduction

Chapter 2
Robustness Analysis

Chapter 3
System Evaluation

Chapter 4
Controller Synthesis

Appendix A
Bibliography

Appendix B
Technical Support and Professional Services

Index

1

Introduction

The Xmath Robust Control Module (RCM) provides a collection of analysis and synthesis tools that assist in the design of robust control systems.

This chapter starts with an outline of the manual and some use notes. It continues with an overview of the Xmath Robust Control Module (RCM) functions.

Using This Manual

This manual provides complete documentation for all the RCM functions along with their associated theoretical background, references, and examples.

Document Organization

This manual includes the following chapters:

- Chapter 1, *Introduction*, describes the Robust Control Module (RCM) and shows the RCM function structure.

- Chapter 2, *Robustness Analysis*, covers the robustness analysis tools and introduces the concepts of uncertainty, robustness, and performance degradation in the framework of closed-loop systems. The *Modeling Uncertain Systems* section should be read by all those interested in robustness analysis or performance degradation, which are explained in the *Stability Margin (smargin)* section and the *Worst-Case Performance Degradation (wcbode)* section. The *Advanced Topics* section provides additional information but this material is not prerequisite to the use of RCM functions.

- Chapter 3, *System Evaluation*, describes system analysis functions that create singular value Bode plots, performance plots, and calculate the L_∞ norm of a linear system. This chapter should be of interest to all users.

- Chapter 4, *Controller Synthesis*, discusses synthesis tools in two categories, H_∞ and H_2. This manual does not attempt to explain all of the theory of H_∞, LQG/LTR, and frequency shaped LQG design

techniques. The general problem setup is explained together with known limitations; the rest is left to the references.

Bibliographic References

Throughout this document, bibliographic references are cited with bracketed entries. For example, a reference to [DoS81] corresponds to a document published by Doyle and Stein in 1981. For a table of bibliographic references, refer to Appendix A, *Bibliography*.

Commonly-Used Nomenclature

This manual uses the following general nomenclature:

- Matrix variables are generally denoted with capital letters; vectors are represented in lowercase.

- $G(s)$ is used to denote a transfer function of a system where s is the Laplace variable. $G(q)$ is used when both continuous and discrete systems are allowed.

- $H(s)$ is used to denote the frequency response, over some range of frequencies of a system where s is the Laplace variable. $H(q)$ is used to indicate that the system can be continuous or discrete.

- A single apostrophe following a matrix variable, for example, x', denotes the transpose of that variable. An asterisk following a matrix variable (for example, A*) indicates the complex conjugate, or Hermitian, transpose of that variable.

Related Publications

For a complete list of MATRIXx publications, refer to Chapter 2, *MATRIXx Publications, Help, and Online Support*, of the *MATRIXx Getting Started Guide*. The following documents are particularly useful for topics covered in this manual:

- *MATRIXx Getting Started Guide*
- *Xmath User Guide*
- *Xmath Control Design Module*
- *Xmath Interactive Control Design Module*
- *Xmath Interactive System Identification Module, Part 1*
- *Xmath Interactive System Identification Module, Part 2*
- *Xmath Model Reduction Module*

- *Xmath Optimization Module*

- *Xmath Robust Control Module*

- *Xmath Xμ Module*

MATRIXx Help

Robust Control Module function reference information is available in the *MATRIXx Help*. The *MATRIXx Help* includes all Robust Control functions. Each topic explains a function's inputs, outputs, and keywords in detail. Refer to Chapter 2, *MATRIXx Publications, Help, and Online Support*, of the *MATRIXx Getting Started Guide* for complete instructions on using the Help feature.

Overview

RCM functionality is structured as shown in Figure 1-1.
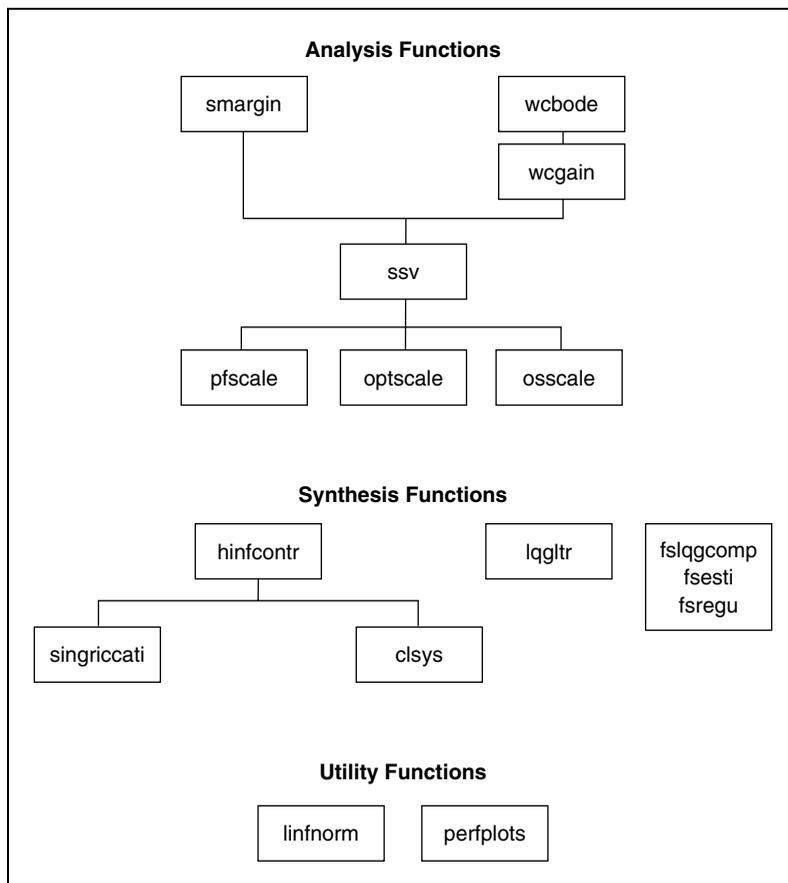
Figure 1-1. RCM Function Structure

Many RCM functions are based on state-of-the-art algorithms implemented in cooperation with researchers at Stanford University. The robustness analysis functions are based on structured singular value calculations. The synthesis tools expand on existing LQG (H_2) techniques (LQG/LTR and frequency shaping) while adding new H_∞ design functions.

2

Robustness Analysis

This chapter describes RCM tools used for analyzing the robustness of a closed-loop system. The chapter assumes that a controller has been designed for a nominal plant and that the closed-loop performance of this nominal system is acceptable. The goal of robustness analysis is to determine whether the performance will remain acceptable if the plant differs from the nominal plant.

Modeling Uncertain Systems

This section describes the method RCM uses to model an uncertain system. The closed-loop system is modeled as a *known* or *nominal* closed-loop system with input *w* and output *z*, together with *k unknown* or *uncertain* transfer functions $\delta_1(j\omega)$, …, $\delta_k(j\omega)$, as shown in Figure 2-1.

Uncertain Transfer Function

Known Nominal System

q_1 r_1

δ_1

w

z

q_2 r_2

δ_2
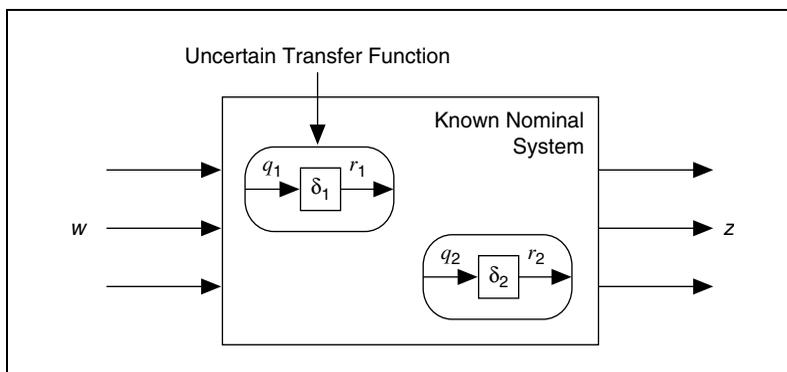
Figure 2-1. Model of an Uncertain System

The following transfer functions are assumed to be stable:

$$\left|\delta_i(j\omega)\right| \le l_i(j\omega) \tag{2-1}$$

where the l_i are given non-negative functions of frequency. This type of uncertainty model is known as *structured nonparametric uncertainties*. To describe this model, you also must describe the nominal closed-loop

system, including how the uncertain transfer functions are connected to the system and the magnitude bound functions $l_i(w)$.

To do this, extract the uncertain transfer functions and collect them into a k-input, k-output transfer matrix Δ, where:

$$\Delta(j\omega) = diagonal(\delta_1(j\omega),...,\delta_k(j\omega)) \qquad (2\text{-}2)$$

The resulting closed-loop system can be viewed as a *feedback connection* of the nominal closed-loop system with transfer matrix $H(j\omega)$ and the uncertain transfer matrix $\Delta(j\omega)$. You describe your nominal closed-loop system as a linear system with

input $\begin{bmatrix} w \\ r \end{bmatrix}$ and output $\begin{bmatrix} z \\ q \end{bmatrix}$.

Note   The signals r and q are not really inputs and outputs of the nominal system; r and q show how the uncertain transfer functions connect to your nominal system. The signals r and q each have k components.

You will partition H into the four submatrices,

$$H = \begin{bmatrix} H_{zw} & H_{zr} \\ H_{qw} & H_{qr} \end{bmatrix}$$

so that H_{zw} is the nominal transfer matrix from w to z, H_{zr} is the nominal transfer matrix from r to z, H_{qw} is the nominal transfer matrix from w to q, and H_{qr} is the nominal transfer matrix from r to q.

The magnitude bound functions $l_i(j\omega)$ from Equation 2-1 are described with the PDM `delb`:

$$\text{DELB} = \begin{bmatrix} \begin{bmatrix} \omega_1 \\ \vdots \\ \omega_m \end{bmatrix}, & \begin{bmatrix} l_1(\omega_1) & \ldots & l_k(\omega_1) \\ \vdots & & \vdots \\ l_1(\omega_m) & \ldots & l_k(\omega_m) \end{bmatrix} \end{bmatrix}$$

Thus, a complete description of your system requires the system `SysH` to represent H_{jw} and the response `delb` to represent the bounds.

Stability Margin (smargin)

Assume that the nominal closed-loop system is stable. That belief raises a question: Does the system remain stable for all possible uncertain transfer functions that satisfy the magnitude bounds (Equation 2-1)? If so, the system is said to be *robustly stable*. If the magnitude bounds are small enough, the uncertainties will not destabilize the system; your system will be robustly stable.

Roughly speaking, the stability margin of your system is defined as the factor by which you can increase all the magnitude bounds l_i and still maintain stability for all possible uncertain transfer functions δ_i. If this number is larger than one (0 dB), then you know that there are no uncertain transfer functions that satisfy the magnitude bound and destabilize your system. Moreover, the number tells you how much more uncertainty your system could tolerate than the given bounds $l_i(\omega)$. If the margin is less than one, then there are uncertain transfer functions that satisfy the magnitude bound (Equation 2-1) and result in an unstable system. In this case, the margin tells you how much you must reduce the magnitude bounds before you have robust stability.

More precisely, the stability margin at frequency ω is defined as the smallest α such that the system can have a pole at $j\omega$, with the uncertain transfer functions satisfying $|\delta_i(j\omega)| \le \alpha l_i(\omega)$:

$$\text{margin(w)} = \min\{ \ \alpha|\ \text{systems can have a pole at } j\omega \text{ with magnitude bounds } \alpha l_i(j\omega) \ \}$$

The stability margin also can be expressed as:

$$\text{margin(w)} = \min\{ \ \alpha|\ \det I - H_{qr}j\omega\Delta \ne 0 \text{ such that } |\Delta_{ii}| \le \alpha l_i(\alpha) \ \}$$

Note The stability margin only depends on H_{qr}.

The margin often is expressed in dB. If the margin is greater than zero for all frequencies, then your system is robustly stable. If the margin is less than zero for some frequencies, then your system is not robustly stable. In particular, there are uncertain transfer functions that satisfy the magnitude bound (Equation 2-1) and cause the system to have a pole at those frequencies where the margin is negative. This does not mean that any δ_i values that satisfy the magnitude bound will destabilize the system: it means that there are some bad δ_i values that satisfy the magnitude bounds and destabilize the system.

smargin( )

```
marg = smargin(SysH, delb {scaling, graph})
```

The smargin( ) function plots an approximation to the stability margin of the system as a function of frequency. For a full discussion of smargin( ) syntax, refer to the *MATRIXx Help*. The approximation is exact if the number of uncertain transfer functions is less than four and scaling="OPT" (optimum scaling).

In other cases, the approximation is generally considered to be extremely good. Refer to the *Approximation with Scaled Singular Values* section. The approximation is always conservative. smargin( ) always will report a margin that is less than or equal to the actual margin.

The smargin( ) function counts the columns in delb to calculate the number of uncertainties *k*. It then assumes that the last *k* inputs of SysH are signal *r* in Figure 2-2, and the last *k* outputs are signal *q*. To create a Nominal System, refer to the *Creating a Nominal System* section.
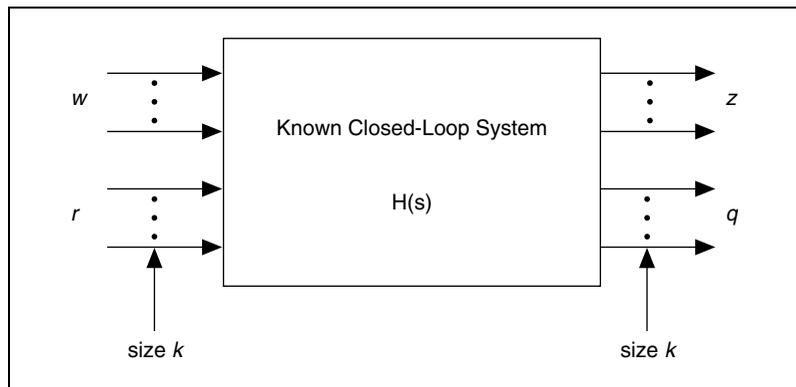
Figure 2-2. Nominal Closed-Loop System

Creating a Nominal System

To better understand how to create *H*(*s*) in Figure 2-3, you will examine a SISO tracking system with three uncertainties. δ_1 is a multiplicative actuator uncertainty, while δ_2 and δ_3 are multiplicative sensor uncertainties.
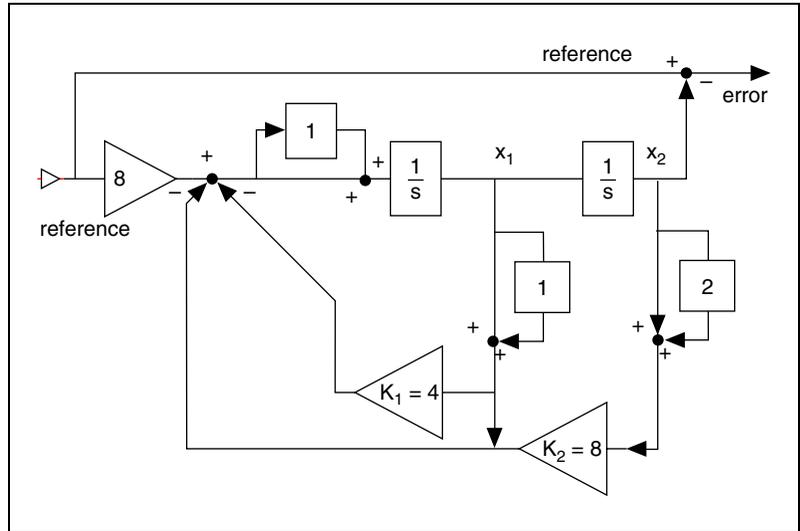
Figure 2-3. SISO Tracking System with Three Uncertainties

The *H* system will have the reference input as input1 and the error output as output1 (*w* and *z*, respectively, in Figure 2-2). Removing the δ values will create inputs 2 through 4 and outputs 2 through 4 (*r* and *q*, respectively, in Figure 2-2).

1. The A, B, C, D matrices of the state-space system representing H are as follows:

    ```
    A=[-4,-8;1,0];
    B=[8,1,-4,-8;zeros(1,4)];
    C=[0,-1;-4,-8;1,0;0,1];
    D=[1,0,0,0;8,0,-4,-8;zeros(2,4)];
    H = system(A,B,C,D,{inputNames=["reference",
        "r1","r2","r3"],outputNames=["error",
        "q1","q2","q3"],stateNames=["x1","x2"]});
    ```

2. Specify the uncertainty bounds.

    The sensor uncertainty δ_3 is known to be bounded by $l_3(w)$, according to Equation 2-1. Because the position x_2 sensor model is known to be accurate to 10% up to one radian per second, and very inaccurate at high frequencies, the l_3 shown in Figure 2-4 is selected.
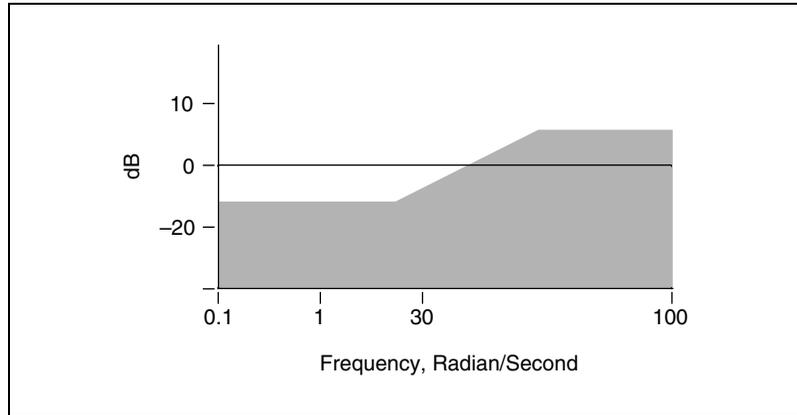
Figure 2-4. Bound for Sensor Uncertainty

✎ **Note** A value of l_3 at one radian per second of –20 dB indicates that modeling uncertainties of up to 10% (–20 dB = 0.1) are allowed.

The actuator and sensor uncertainties δ_1 and δ_2 are bounded by –20 dB at all frequencies. You will use these values to interpolate to obtain l_3. First, create the bound for δ_3 in Hz.

```
L3 = pdm([-20,-20,10,10],[0.1,1,30,100]/2/pi);
```

3. Now interpolate to obtain 30 points:

```
L3 = interpolate(L3,logspace(0.01,10,30),{xlog});
```

4. Create L1 and L2 (bounds for δ_1 and δ_2):

```
L1=-20*ones(L3); L2 = L1;
delb = [L1,L2,L3];
```

5. Calculate the stability margin:

```
marg=smargin(H,delb);
```

smargin --> Scaling algorithm is type: PF
smargin --> Margin computation 10% complete
smargin --> Margin computation 50% complete
smargin --> Margin computation 90% complete

The output indicates that Perron-Frobenius scaling (the default) is used. Refer to the *Approximation with Scaled Singular Values* section.

The stability margin plot is shown in Figure 2-5. The minimum margin is about 8 dB at about 1/2 Hz. This implies that all three l_1 values (uncertainty bounds) could be increased (relaxed) simultaneously by 8 dB, and the system would still remain robustly stable.
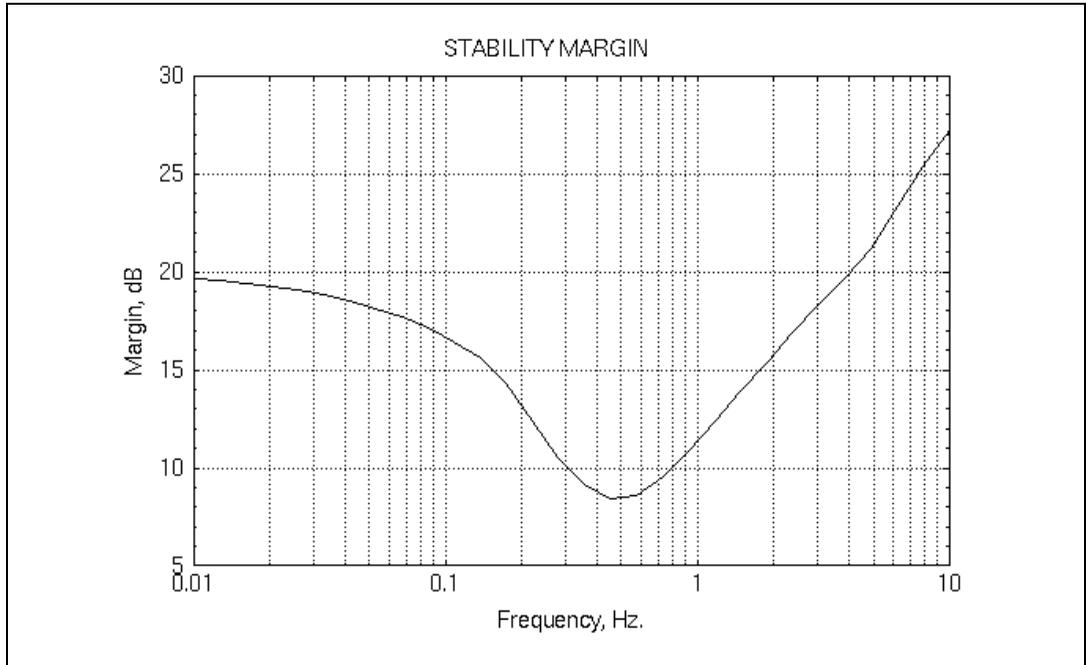
Figure 2-5. Stability Margin

Now examine the effect on the stability margin of discretizing $H(s)$ at 100 Hz.

```
dt = 0.01;
Hd = discretize(H,dt);
margD = smargin(Hd,delb);
```

smargin --> Scaling algorithm is type: PF
smargin --> Margin computation 10% complete
smargin --> Margin computation 50% complete
smargin --> Margin computation 90% complete

100 Hz is a high discretization frequency for H, so the stability margin is unchanged in the discrete-time case. The new plot is not much different from Figure 2-6. Again, minimum margin is about 8 dB at about 1/2 Hz.

Worst-Case Performance Degradation (wcbode)

Even if a system is robustly stable, the uncertain transfer functions still can have a great effect on performance. Consider the transfer function from the qth input, w_q, to the pth output, z_p. With $\delta_1 = ... = ...\delta_k = 0$, you have the nominal system, and this transfer function is the p,q entry of H_{zw}. This is called the *nominal transfer function*.

When the δ values are not zero, the transfer function from w_q to z_p is the p,q entry of H_{pert} given by the formula:

$$H_{pert} \;=\; H_{zw} + H_{zr}\Delta(I - H_{qr}\Delta)^{-1}H_{qw}$$

This is referred to as the *perturbed transfer function*. The perturbed transfer function depends on the particular $\delta_1, \ldots, \delta_k$.

If the magnitude bounds are small enough, then you expect the perturbed transfer function H_{pert} to be close to the nominal transfer function. Roughly speaking, small perturbations should not significantly alter the closed-loop transfer function from w_q to z_p.

The *worst-case gain* is defined as the largest magnitude of the perturbed transfer function, considering all δ values that satisfy the magnitude bound. More precisely:

$$wcgain(\omega) \;=\; max\{\|H_{pert,pq}\| \;\mid\; \Delta \;=\; \text{diagonal}(\delta_1,...,\delta_k),\, |\delta_i| \leq l_i(\omega)\} \quad (2\text{-}3)$$

wcgain(ω) is always larger than the nominal gain, $|H_{zw,pq}(j\omega)|$. This is not because the uncertain transfer functions only can increase the magnitude of the transfer function from w_q to z_q. In fact, it is possible that for a lucky choice of the δ values, the perturbed transfer function actually can be smaller than the nominal transfer function over all frequencies. But in the worst-case gain, you consider only the worst possible δ values, and these always increase the perturbed gain over the nominal gain.

Intuitively, if the stability margin is large, then the uncertain transfer functions should not greatly effect the gain from w_q to z_p, so that *wcgain(ω)* should be not much larger than the nominal gain $|H_{zw,pq}(j\omega)|$. If the stability margin is small, however, *wcgain(ω)* could be much larger than the nominal gain. An extreme case occurs if the stability margin is negative (in dB) at the frequency δ. Then you have *wcgain(ω) = ∞*, although `wcbode( )` clips the worst-case gain curve so that it never exceeds (the maximum nominal gain) * 100, or +20 dB. Of course, instability is an extreme form of performance degradation.

wcbode( )

```
[WCMAG, NOMMAG] = wcbode (SysH, delb, {input, output,
graph})
```

The wcbode( ) function computes and plots the worst-case gain of a closed-loop transfer function.

This function is useful for checking a system that already has been verified to be robustly stable using smargin( ). For example, a system can have a minimum stability margin of 4 dB, so it is robustly stable. If the worst-case gain from a function input to the output it commands has a 20 dB peak, then even though the system is robustly stable, the design is unacceptable. On the other hand, if you verify that the perturbed closed-loop transfer function increases only 2 dB over the nominal, then the design is probably acceptable.

The wcbode( ) function computes and plots an approximation to *wcgain(ω)*, the largest possible magnitude of a perturbed closed-loop transfer function that can be caused by uncertain transfer functions that satisfy the magnitude bound. The wcbode( ) function is conservative: it does not under-report the maximum of the perturbed transfer function.

A large value of wcbode( ) indicates instability: $wcgain(\omega) = \infty$. In this case, wcbode( ) returns a maximum value of ten times the maximum of the nominal transfer function over all frequencies. Consequently, the window is clipped at 20 dB above the maximum of the nominal transfer function over all frequencies. The wcbode( ) function also plots the nominal transfer function for reference.

Using wcbode( ) to Analyze Performance Degradation

The wcbode( ) function can be used to analyze performance degradation for the system you have been using (Figure 2-3). The transfer function, which should be small, is from reference to error (input 1 to output 1). Figure 2-6 shows the results of the following function call:

```
[NOMMAG,WCMAG]=wcbode(H,delb,{input=1,output=1});
```

The performance degradation due to the uncertainties is small but not negligible.
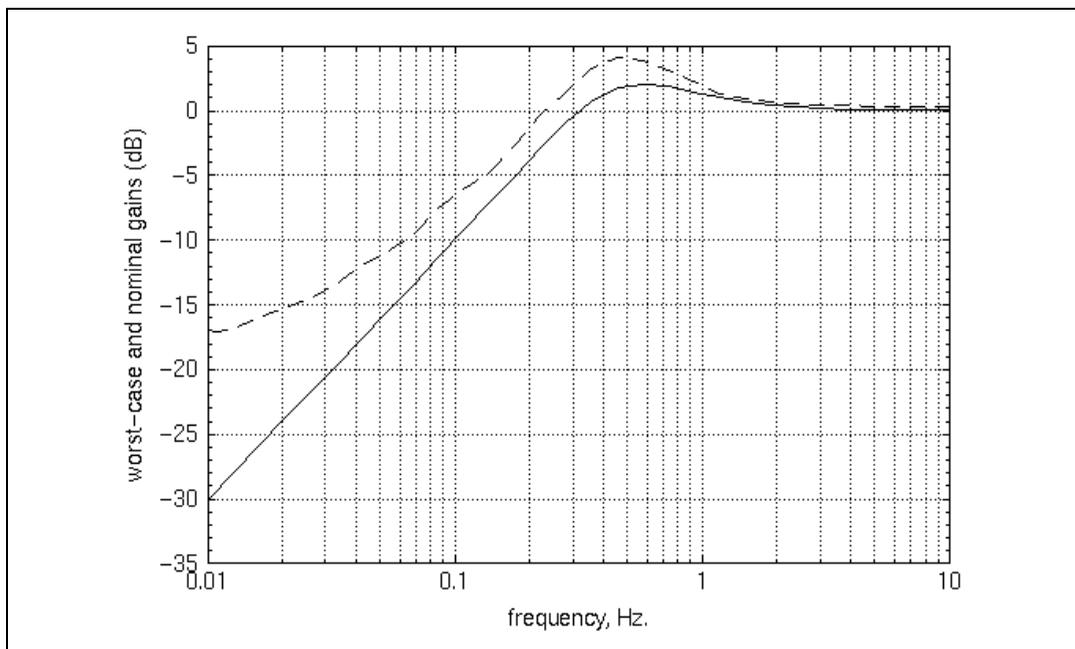
Figure 2-6. Performance Degradation of the SISO Tracking System

Advanced Topics

This section describes the theoretical background on robustness analysis and performance degradation.

Stability Margin

This section discusses advanced aspects of computing the stability margin and the related scaling algorithms.

Stability Margin and Structured Singular Values (μ)

The stability margin was first defined by Safonov in [Saf82]. If you let

$$M = H_{qr}diagonal(l_1(w), ...,l_k(w))$$

then you can express the margin at frequency *d* as

$$\text{margin}(\omega) = max \qquad \{\alpha | det(I - M\Delta) \neq 0$$

for all diagonal Δ such that

$$(\left|\Delta_{ii}\right| \leq \alpha)\} = \frac{1}{\mu(M)}$$

where $\mu(\cdot)$ is the *structured singular value*, introduced by Doyle in [Doy82]. Thus, the margin is the inverse of the structured singular value of H_{qr} diagonally scaled by the magnitude bounds.

There is no numerically efficient algorithm that is guaranteed to compute $\mu(M)$, and hence the stability margin. However, it is possible to compute various good approximations to $\mu(M)$. One of these approximations is often exact.

Stability Margin Bounds Using Singular Values

A popular but conservative method uses singular values:

$$\text{margin}(\omega) \geq \frac{1}{\sigma_{max}(M)} \tag{2-4}$$

Plotting the right side of Equation 2-4 gives a lower bound on the actual stability margin. To get this plot, specify smargin( ) with scaling="SVD". This approximation can be very conservative, meaning that the left side can be much larger than the right side. This fact spurred the study of structured singular values and the other approximations discussed in the following sections.

Use of Scaling Example

For this example, you will use the system in Figure 2-3. This time smargin( ) will be invoked with scaling="SVD", so smargin( ) will calculate Equation 2-4.

```
margSVD = smargin(H,delb,{scaling="SVD"});
```

smargin --> Scaling algorithm is type: SVD
smargin --> Margin computation 10% complete
smargin --> Margin computation 50% complete
smargin --> Margin computation 90% complete

You can compare this margin to that of the example in the *Creating a Nominal System* section; the following inputs produce Figure 2-7.

```
plot ([marg,margSVD],{xlog}
    legend=["PF_SCALE","SVD"],
    ylab="Stability Margin,dB",
    xlab="Frequency, Hz."})
```
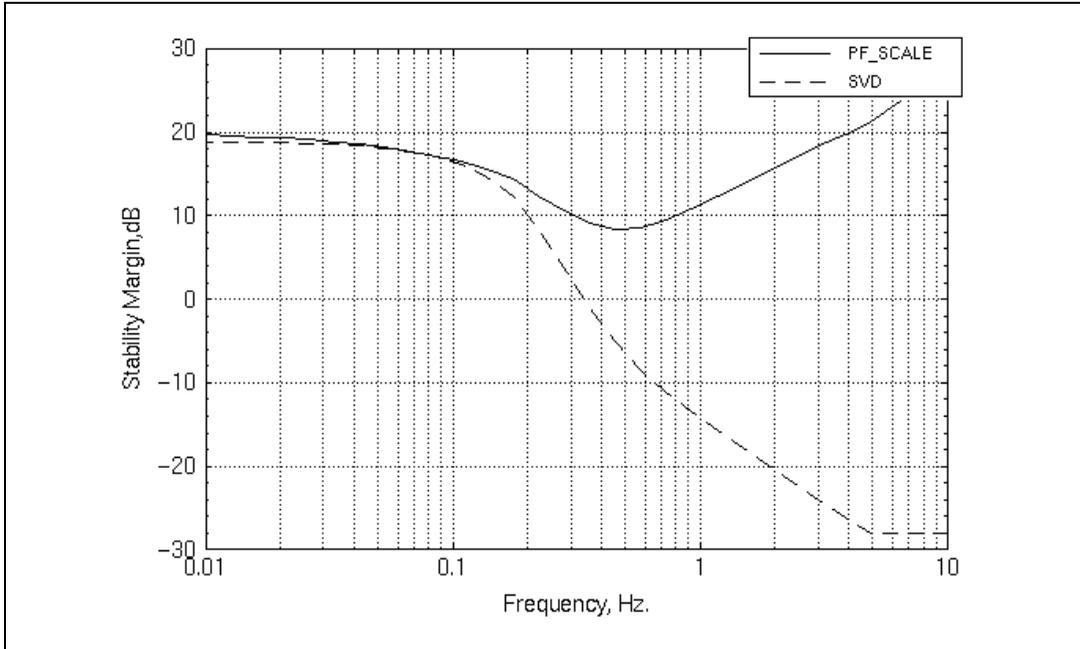

Figure 2-7. pfscale( ) versus svd Stability Margins

Note   The singular value approach gives results that are too conservative, suggesting that the uncertainties can destabilize the system. Conversely, you know from the scaled singular value calculations that the system is robustly stable.

Approximation with Scaled Singular Values

In [Saf82] and [Doy82], the inequality

$$\min_{D \ diagonal} \sigma_{max}(DMD^{-1}) \geq \mu(M) \tag{2-5}$$

is noted. This optimization problem can be shown to be unimodal—for $D > 0$, an assumption that can be made without loss

of generality—so, roughly speaking, it can be solved. [SD83,SD84] discusses this optimization problem.

Notice that:

$$\sigma_{max}(M) = \bar{\sigma}(DMD^{-1}) \qquad for \qquad D = 1$$

so you have the following from Equation 2-5:

$$\sigma_{max}(M) \geq \mu(M)$$

This inequality is thought to be nearly an equality, so that the left side is a good engineering approximation to the right side. No theory supports this generally held belief, but no example is known where the left side is more than 15% larger than the right side. Equality can be shown to hold provided k ≤ 3—for example, if there are three or fewer uncertain transfer functions [Doy82].

Note   The approximation equation of $\mu(M)$ (Equation 2-5) is an *upper bound*. This means that the stability margin calculated using this approximation is conservative, that is, less than the actual stability margin. This optimization problem itself can be difficult. Osborne [Osb60] and Safonov [Saf82] provide two methods for finding good suboptimal scalings for Equation 2-5.

Both Osborne's and Safonov's Perron-Frobenius scalings usually have been found to be close to the optimum for the optimization problem equation. The resulting approximations,

$$\hat{u}_{OS}(M) = \sigma_{max}(D_{OS}MD_{OS}^{-1})$$
$$\hat{u}_{PF}(M) = \sigma_{max}(D_{PF}MD_{PF}^{-1})$$

are thought to be good engineering approximations to μ. `optscale( )` provides an iterative optimization function based on the ellipsoid algorithm.

Comparing Scaling Algorithms

Using the system from the first example (Figure 2-3), you can compare
the results of using the three scaling algorithms:

```
MARG_PF=smargin(H,delb,{scaling="PF",!graph});
MARG_OS=smargin(H,delb,{scaling="OS",!graph});
MARG_OPT=smargin(H,delb,{scaling="OPT",!graph});
plot ([MARG_PF,MARG_OS,MARG_OPT],{xlog,
    legend=["PF","OS","OPT"],xlab="Frequency, Hz.",
    ylab="Stability margin, dB"})
```

Figure 2-8 plots the margins produced by the three scaling algorithms.
Notice that in this problem the three scalings yield identical stability
margins.
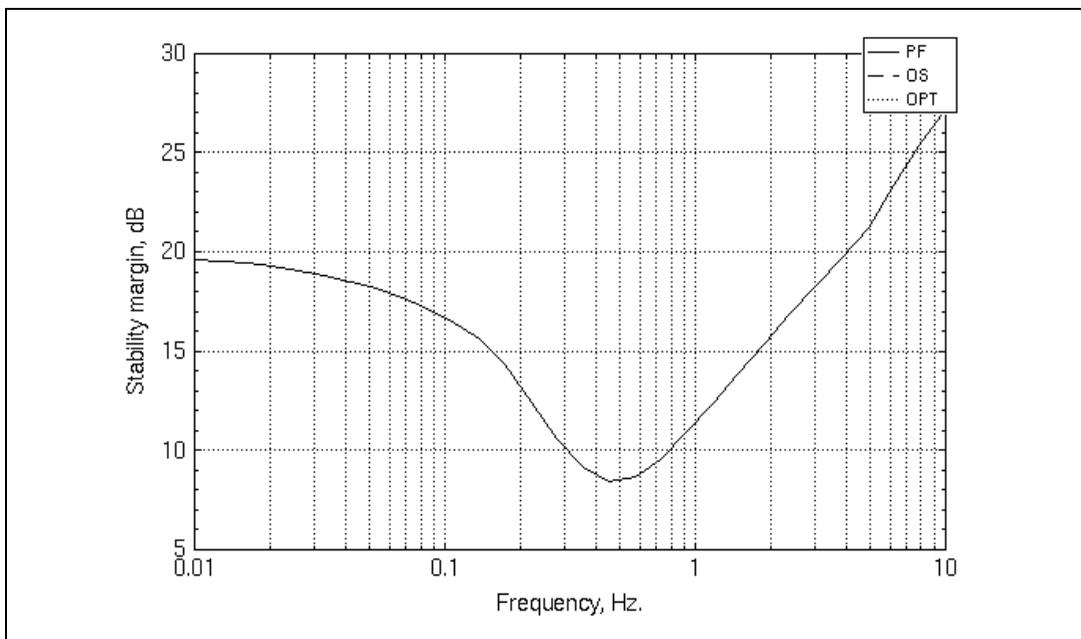
Figure 2-8. Results of Scaling Algorithm Options

ssv( )

```
[v,vD] = SSV(M, {scaling})
```

The `ssv( )` function computes an approximation (and guaranteed upper bound) to the Scaled Singular Value of a complex square matrix *M*, where *M* can be a reducible matrix. The scaled singular value *v(M)* is defined by:

$$v(M) = \inf_{D \in C^{n \times n}, \, det(D) \neq 0, \, diagonal} \overline{\sigma}(DMD^{-1})$$

Scaling can be accomplished with one of three algorithms:

- **Perron-Frobenius**—If `{scaling="PF"}` Safonov's Perron-Frobenius method [Saf82] is used. This method finds the scaled singular value for non-negative real matrices *M*. In general, it is suboptimal if *M* is complex. This algorithm is the default because empirical tests show that is the fastest of the three.

- **Osborne**—If `{scaling="OS"}`, Osborne's Method [Osb60] is used. This method solves the problem of finding D_O such that

$$D_O M D_O^{-1} \inf_{D \; diagonal} \left\| DMD^{-1} \right\|_F$$

  where *D* is diagonal and positive, and $\left\| \cdot \right\|_F$ is the Frobenius norm. Thus, the Osborne method minimizes the Frobenius norm, and is therefore suboptimal.

- **Optimal**—If `{scaling="OPT"}`, Boyd's ellipsoid algorithm [BYB89] is used. This algorithm computes the scaled singular value to a guaranteed accuracy. It is, however, the most computationally expensive of the three algorithms.

ssv( ) Examples

Consider the complex matrix *M*:

```
M = [-1, jay, 0; 0, 2*jay, 1+jay;1, 0, 1];
```

`ssv( )` can return the optimally scaled singular value of *M* using Osborne, Perron-Frobenius, or Boyd methods:

```
VOS=ssv(M,{scaling="OS"})
```
VOS (a scalar) =   2.56723

```
VPF=ssv(M,{scaling="PF"})
```
VPF (a scalar) =   2.45133

```
VOPT=ssv(M,{scaling="OPT"})
```
VOPT (a scalar) =   2.43952
```
VSVD = max(svd(M))
```
VSVD (a scalar) =   2.65886

osscale( )

```
[v, vD] = osscale(M)
```

The osscale( ) function scales a matrix using the Osborne Algorithm.
A diagonal scaling D_{OS} is found that minimizes the Frobenius norm of
$D_{OS}MD_{OS}^{-1}$, which is the square root of the sum of the squares of its
singular values. If *M* is reducible, osscale( ) may encounter a divide
by zero. To avoid this, use ssv( ) with the Osborne scaling option:

```
[v,vD]=ssv(M,{scaling="OS"})
```

pfscale( )

```
[v, vD] = pfscale(M)
```

The pfscale( ) function scales a matrix using the Perron-Frobenius
Algorithm. This scaling is optimal for matrices with all positive entries.
The matrix *M* must be irreducible for this function. If *M* is reducible,
use ssv( ) with the Perron-Frobenius scaling option instead:

```
[v,vD]=ssv(M,{scaling="PF"})
```

The optimum diagonal scaling is found for *M* using the Perron-Frobenius
theory of non-negative matrices. This scaling is given by

$$D_i^{PF} = \sqrt{\frac{p_i}{q_i}}$$

where *p* and *q* are right and left eigenvectors of | associated with its largest
eigenvalue:

$$Mp = \lambda_{max}p, \qquad M^Tq = \lambda_{max}q, \qquad p \neq 0 \neq q$$

optscale( )

```
[v, vOPTD] = optscale (M, {tol})
```

The optscale( ) function optimally scales a matrix. An iterative
optimization (ellipsoid) algorithm which calculates upper and lower
bounds on the left side of Equation 2-5 is used. If these bounds are within
a relative accuracy you have specified (tol), optscale( ) stops.
optscale( ) also will stop and issue a warning if the maximum number
of iterations is reached:

$$200 \times \sqrt{\text{rows(M)}}$$

optscale( ) will find a $\mu(M)$ no larger than pfscale( ).

Reducibility

In some cases, the uncertain transfer functions can be divided into groups
that do not interact. This is illustrated in Figure 2-9.
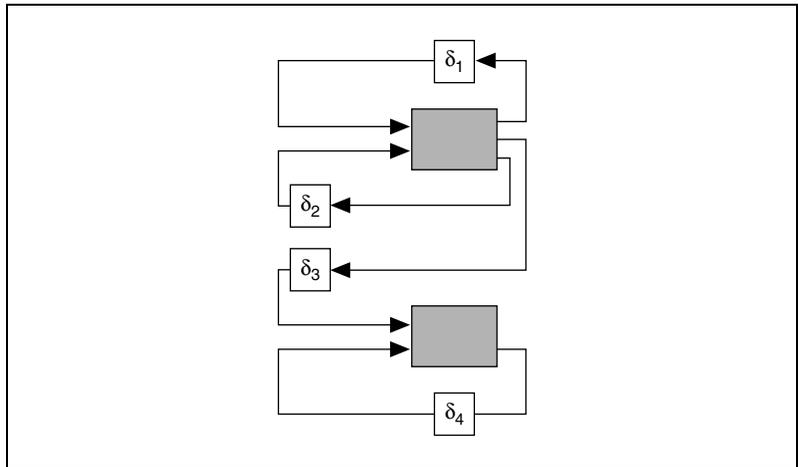
Figure 2-9. Non-Interacting Uncertain Transfer Functions

As you can see, δ_3 does not affect the stability margin at all because there
is no feedback through it. The system in Figure 2-9 can be reduced to the
two separate systems shown in Figure 2-10. The stability margin of
Figure 2-9 is the minimum of the stability margins of the systems in
Figure 2-10.
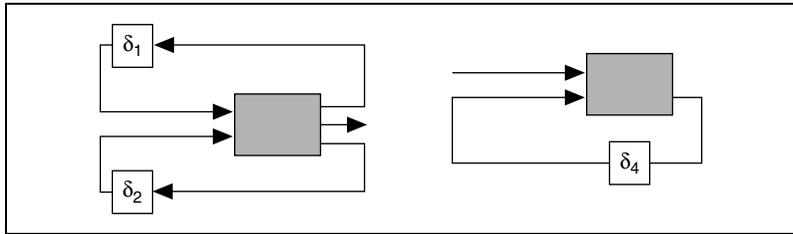
Figure 2-10. Reduction to Separate Systems

In terms of the approximations to the margin discussed above, this reducibility will manifest itself as a problem such as divide-by-zero or nontermination. It really means that the minimum of the optimization problem is not achieved by any finite scaling.

A matrix M can be split into its reducible components using the following technique (refer to[BeP79]):

1. Form the matrix $X = (\alpha I + M) - 1$ for any α larger than the spectral radius of M, for example $2\|M\|$.

2. Form $Y = X + X^T$ where Y has a positive i,j entry if and only if δ_i and δ_j are in the same reduced system; otherwise, the entries will be zero.

`ssv( )` checks for reducibility before invoking a scaling algorithm. The margins of each of the reduced systems then can be calculated separately, and the minimum taken.

Worst-Case Performance Degradation (wcgain)

Conversion to a Stability Margin Problem

In [DWS82], it is shown that a simple relation holds between the worst-case gain defined in Equation 2-3, and the stability margin. For $\gamma > 0$,

```
wcgain (jw) f g if and only if m(Hred(jw) diag(g-1,
l1(w),...,lk(w)) f 1
```

where H_{red} is H with the rows and columns corresponding to all inputs and outputs deleted except the ones of interest (the qth input and the pth output). This can be interpreted as adding a fictitious uncertain transfer function from w_q to z_p with magnitude bound γ^{-1} at the given frequency. This additional uncertainty is called a *performance loop* as described in reference [BoB91].

Using this relation and any of the previously discussed approximations for $\mu(\cdot)$, you can compute an approximation to `wcgain( )`. Because the approximations to $\mu(\cdot)$ are upper bounds, the resulting approximations to `wcgain( )` also are upper bounds. For speed purposes, `wcgain( )` uses Perron-Frobenius scaling to calculate the approximation of μ.

wcgain( )

```
gamma = wcgain(H, gammin, gammax, gam0)
```

The `wcgain( )` function estimates the largest possible magnitude from a given input of the system to a given output, when the other inputs are connected to the other outputs through uncertain transfer functions bounded by one.

For a discussion of `wcgain( )` syntax, refer to the *Xmath Help*. This is a low-level function that calculates the worst-case gain at a single frequency, where the magnitude bounds are normalized to one as follows:

$$l_1 = \ldots = l_k = 1$$

Because it is a lower level function, there is no syntax checking. This function is called by the `wcbode( )` function.

3

System Evaluation

This chapter describes system analysis functions that create singular value Bode plots, performance plots, and calculate the L_∞ norm of a linear system.

Singular Value Bode Plots

The singular value Bode plot is a MIMO generalization of the `bode( )` magnitude plot. It is calculated as

$$\sigma_i(H(jw)), \qquad i = 1\dots k$$

where

$$k = min((n_{inputs}, n_{outputs}))$$

and

$$\sigma_1(H) \geq \sigma_2(H) \geq \dots \geq \sigma_k(H) \geq 0$$

In these equations, $\sigma_i(H(jw))$ can be thought of as the maximum gain of the system at frequency ω, and $\sigma_k(H(jw))$ can be thought of as the minimum gain of the system at frequency ω.

If $\sigma_i(H(jw)) \gg \sigma_k(H(jw))$, then at the frequency ω, the system gain can be large for some input directions and small for other input directions.

The singular value plot allows you to generalize to the MIMO case notions such as "the command-to-tracking error transfer function is small" or "the loop gain is large." For example:

- If the system represents the command-to-tracking error for a closed-loop system, then you would hope that σ_1, and hence all σ values, are small over the bandwidth of the system. This means that the command-to-tracking error is small in all directions at these frequencies.

- The singular value plot of a certain transfer matrix gives a lower bound on the stability margin of the system.

Refer to [BoB91] in Appendix A, *Bibliography*.

Example 3-1   Creating a Singular Value Plot

1. Let a system H be a 2-input/2-output system:

    ```
    tf=makepoly([1,2],"s")/...
        polynomial([0,-2.334,-12],"s")
    ```

    tf (a transfer function) =
    ```
             s + 2
       --------------------
         (s + 2.334)(s + 12)s
       System is continuous
    ```
    ```
    H = [tf, 2*tf; tf*tf, tf+3];
    [outputs,inputs]=size(H)
    ```

    outputs (a scalar) =   2

    inputs (a scalar) =   2

2. Now plot the singular values of the system between 0.01 and 100 Hz
    using `svplot( )`:

    ```
    svplot(H,{Fmin=0.01,Fmax=100})
    ```

The result is shown in Figure 3-1. For a discussion of `svplot( )` syntax,
refer to the *Xmath Help*.
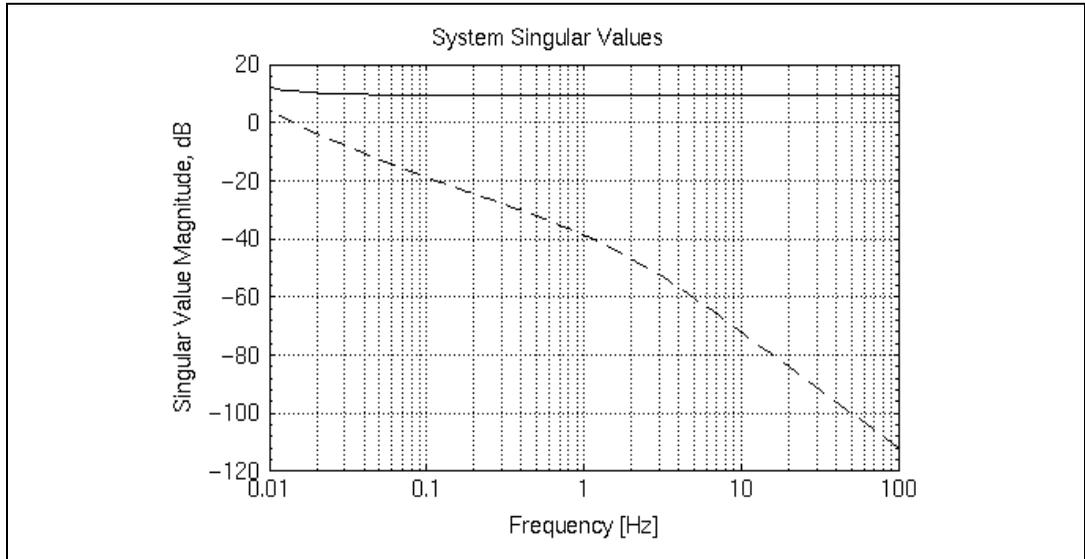
Figure 3-1. Singular Value Plot

L Infinity Norm (linfnorm)

The L_∞ norm of a stable transfer matrix H is defined as:

$$\|H\|_\infty = sup \ \bar{\sigma}(H(j\omega))$$
$$\omega \in \Re$$

where $\bar{\sigma}$ is the maximum singular value and $H(j\omega)$ is the transfer matrix under consideration.

The L_∞ norm of a stable transfer matrix is the maximum of the maximum singular values over frequency. For example, the highest point of its singular values plot. Observe that the L_∞ norm can be calculated even if H is not stable.

A simple interpretation of the L_∞ norm of a stable system can be given as:

$$\|H\|_\infty = max \frac{RMS(y)}{RMS(u)}$$

where u and y are the input and output of H, respectively. This means that $\|H\|_\infty$ is the root mean square (RMS) gain of the system: it is the largest

factor by which the RMS value of a signal flowing through *H* can be increased.

By comparison, the H_2 norm is defined as:

$$\|H\|_2 = \sqrt{\frac{1}{2\pi} \int_{-\infty}^{\infty} \sum_{i=1}^{k} \sigma_i(H(j\omega))^2 \, dw}$$

This norm can be interpreted as the RMS value of the output when the input is unit intensity white noise. It can be computed in Xmath using the `rms( )` function.

For discrete-time systems with a stable *H*,

$$\|H\|_\infty = \max_{\omega \in (-\pi, \pi)} \bar{\sigma}(H(e^{j\omega}))$$

where $\bar{\sigma}$ is the maximum singular value and $H(e^{j\omega})$ is the transfer matrix under consideration.

linfnorm( )

```
[sigma, vOMEGA] = linfnorm( Sys, {tol,maxiter} )
```

The `linfnorm( )` function computes the L_∞ norm of a dynamic system using a quadratically convergent algorithm. The `linfnorm( )` function relies on eigenvalue calculations of a Hamiltonian matrix with twice as many states as `Sys` and, consequently, may be unreliable for large systems. A singular value plot created with `svplot( )` can be used as an alternative in these cases. Refer to the *Singular Value Bode Plots* section.

- The keyword `tol` controls the required relative accuracy. The default is 0.01. `maxiter` is the maximum number of iterations. The default is 15.

- If the maximum norm is found at $\omega = \infty$, `linfnorm( )` returns:

  vOMEGA = Infinity
  sigma = gain at infinity.

- If A has an imaginary eigenvalue at $j\omega_0$, linfnorm( ) returns:

  vOMEGA = ω_0
  SIGMA = Infinity

  where ω_0 is one of the imaginary eigenvalues of A.

- Even if *H* is unstable, linfnorm( ) returns its maximum singular value on the $j\omega$ axis.

For discrete-time systems linfnorm( ) converts a discrete-time L_∞ norm computation problem to a continuous-time problem using a Cayley transformation. For example, it maps the unit circle conformally onto the complex right half plane using a linear fractional transformation. The linfnorm( ) function then calls itself to solve the continuous-time problem, and finally converts the solution back to discrete-time.

Example 3-2   Example of linfnorm( )

```
Sys=system([-0.2,-1;1,0],[1,0]',[0,1],0);
[sigma,omega]=linfnorm(Sys)
```

sigma (a scalar) =   5.07322
omega (a scalar) =   0.157081

The linfnorm( ) function will return the L_∞ norm (sigma) of the transfer matrix $H(j\omega)$ described by Sys, and omega is the vector of frequencies where it is achieved. linfnorm( ) computation can be checked by plotting the singular values of $H(j\omega)$ as a function of ω (Figure 3-2).

```
sv=svplot(Sys,{fmin=.01, fmax=1.0});
```
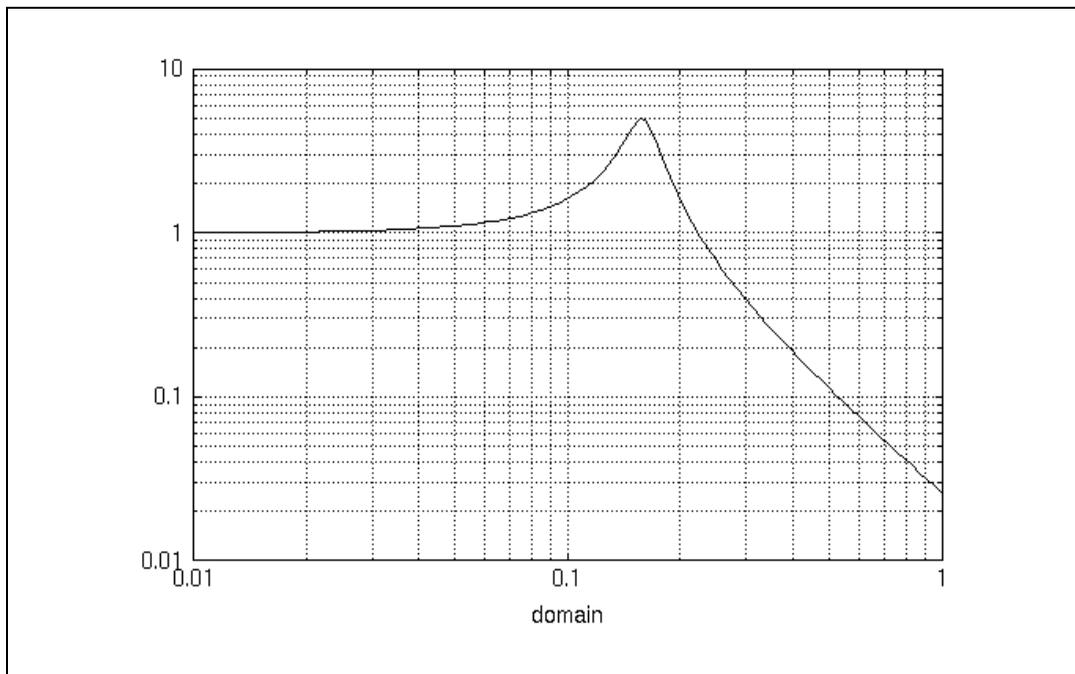
Figure 3-2. Singular Values of $H(j\omega)$ as a Function of ω

✎ **Note**   sv is returned in dBs. Check that sigma is within 0.01 (the default value of tol) of 10**(max(sv,{channels})/20).

```
[sigma,10^(max(sv,{channels})/20)]
```

ans (a row vector) = 5.07322 4.98731

The linfnorm( ) function also can be used on discrete-time systems. Consider a state-space system with a sample rate of 10 Hz:

```
SysD=system([0.5,0.5;0.8,0.5],[0.8,0.5]',
[0,1],0,{dt=0.1})
[sigma,omega]=linfnorm(SysD)
```

sigma (a scalar) =   5.99267
omega (a scalar) =   0

Singular Value Bode Plots of Subsystems

To evaluate the performance achieved by a given controller rapidly, it is useful to check four basic maximum singular value plots—for example, the transfer matrices from process and sensor noises to the error and actuator signals.

perfplots( )

```
SV = perfplots ( Sys, nd, ne, { keywords } )
```

The perfplots( ) function plots the maximum singular value of the four transfer matrices of the system in the following figure.
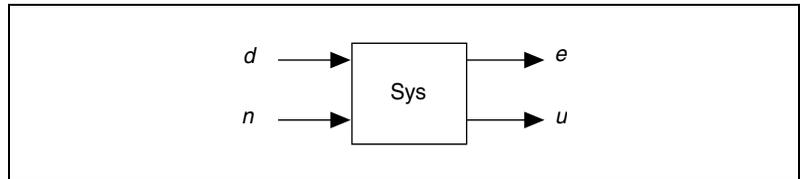
In most applications, the perfplots( ) function is applied to a system of the form shown in Figure 3-3, where P is the plant and K is a proposed controller.
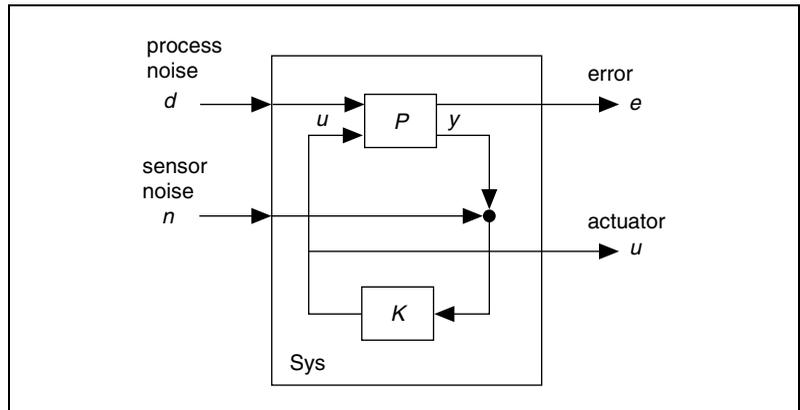
Figure 3-3. Typical System with Plant and Controller

The four transfer matrices are labeled *e/d*, *e/n*, *u/d*, and *u/n* in the final plot. The plots in the top row, consisting of *e/d* and *e/n*, show the *regulation* or *tracking* achieved by the controller. If both these quantities are small, then the disturbance *d* and the sensor noise *n* will not make the error signal *e* large.

The bottom row of plots, consisting of *u/d* and *u/n*, show the *actuator effort* used by the controller. If these are both small, then the actuator effort *u*, which results from the disturbance *d* and the sensor noise *n* will be small.

A classic trade-off in controller design boils down to a choice between making the top row of a perfplot( ) small (good regulation/tracking) and making the bottom row small (low actuator effort). For example, by varying the design parameter ρ in the lqgltr( ) regulator design process, the magnitude of the top two transfer matrices can be traded off against the magnitude of the bottom two. Increasing ρ makes the top two magnitudes smaller but makes the bottom two larger.

The *columns* of a perfplot( ) have a dual interpretation. The plots in the left column, *e/d* and *u/d*, show how sensitive the system is to the process noise or disturbance *d*. The plots in the right column, *e/n* and u/*n*, show how sensitive the system is to the sensor noise *n*. Again, there is a trade-off between making the magnitudes of the transfer matrices on the left small (good disturbance rejection) and making the magnitudes of the transfer matrices on the right small (low sensitivity to sensor noise). In the lqgltr( ) estimator design, the parameter ρ controls the relative magnitude of the left and right plots. Increasing ρ makes the left two magnitudes smaller but makes the right two larger. Refer to Example 3-3.

Example 3-3   Example of perfplots( )

Consider the simple closed-loop system shown in Figure 3-4.
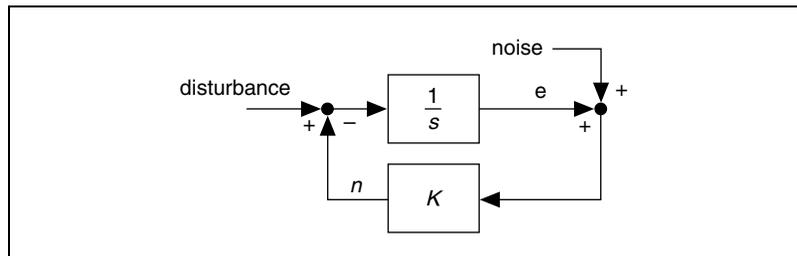
Figure 3-4. Closed-Loop System

The system matrix can be calculated using the `afeedback( )` function for different values of *K*. Consider two cases: *K = 1* and *K = 5*.

```
P = 1/makepoly([1,0],"s")

   P (a transfer function) =
     1
     --
     s

  System is continuous
```

```
K1= 1/makepoly(1,"s")
```

K1 (a transfer function) =
  1

  -

  1

  System is continuous

```
K5= 5/makepoly(1,"s");

Sys1 = afeedback(P,K1);

Sys5 = afeedback(P,K5);
```

The effect of the value of *K* on closed-loop performance can be investigated using `perfplots( )`.

```
sv1 = perfplots(Sys1,1,1);
```

Overlap plots:

```
sv5 = perfplots(Sys5,1,1,{!graph});
for i = 1:4
 plot(sv5(1,i),
    {graphnumber=i,line_style=2,keep})?
endfor
```

In Figure 3-5, you can see that over the bandwidth of 0.1 Hz, the controller *K = 5* has better regulation (*e/d* is smaller for *K = 5* than for *K = 1*, with *e/n* about the same for both cases) but uses slightly more actuator effort. Above the bandwidth of 0.1 Hz, the *e/n* and *u/n* show that the *K = 5* controller is more sensitive to sensor noise. In classic terms, the *K = 5* controller has a higher bandwidth.
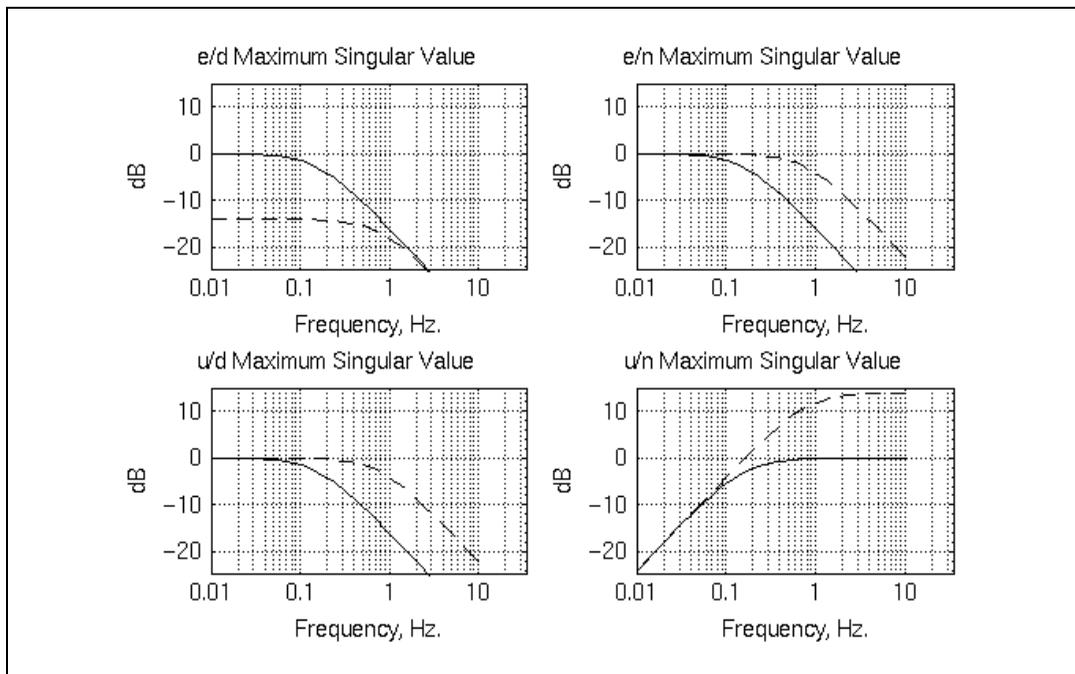
Figure 3-5. Perfplots( ) for *K* = 1 and *K* = 5

clsys( )

```
SysCL = clsys( Sys, SysC )
```

The clsys( ) function computes the state-space realization SysCL, of the closed-loop system from w to z as shown in Figure 3-6.
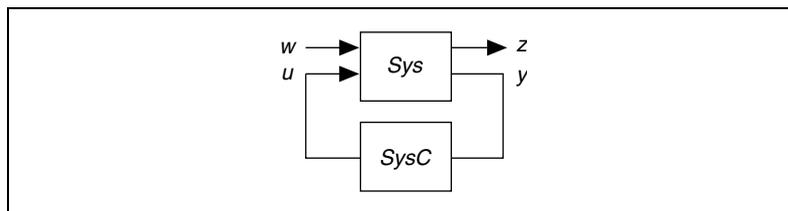
Figure 3-6. Closed Loop System from w to z

Where `SysC=system(Ac,Bc,Cc,Dc)`, `Sys=system(A,B,C,D)`, and `nz` is the dimension of `z` and `nw` is the dimension of `w`:

$$B \text{ is } \begin{bmatrix} B_w \\ B_u \end{bmatrix} \qquad C \text{ is } \begin{bmatrix} C_z, C_y \end{bmatrix} \qquad D \text{ is } \begin{bmatrix} D_{zw} & D_{zu} \\ D_{yw} & D_{yu} \end{bmatrix} \qquad \updownarrow \textbf{nz}$$
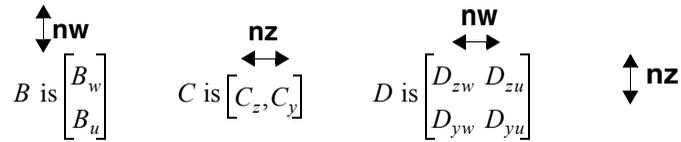
Given the above, `SysCL` is calculated as shown in Figure 3-7.

$$A_{CL} = \begin{bmatrix} A + B_u(I - D_cD_{yu})^{-1}D_cC_y & B_u(I - D_cD_{yu})^{-1}C_c \\ B_cC_y + B_cD_{yu}(I - D_cD_{yu})^{-1}D_cC_y & A_c + B_cD_{yu}(I - D_cD_{yu})^{-1}C_c \end{bmatrix}$$

$$B_{CL} = \begin{bmatrix} B_w + B_u(I - D_cD_{yu})^{-1}D_cD_{yw} \\ B_cD_{yw} + B_cD_{yu}(I - D_cD_{yu})^{-1}D_cD_{yw} \end{bmatrix}$$

$$C_{CL} = \begin{bmatrix} C_z + D_{zu}(I - D_cD_{yu})^{-1}D_cC_y & D_{zu}(I - D_cD_{yu})^{-1}C_c \end{bmatrix}$$

$$D_{CL} = D_{zw} + D_{zu}(I - D_cD_{yu})^{-1}D_cD_{yw}$$

Figure 3-7. Calculation of the Closed Loop System (SysCL)

The closed-loop system is assumed to be well-posed—$(I - D_cD_{yu})$ must be invertible). A well-posed closed-loop system assures that if two given systems, `Sys` and `SysC`, are proper (only proper transfer functions can be represented in state space), then the resulting closed-loop system, `SysCL`, also is proper and therefore realizable in state space.

Figure 3-8 is an example of an ill-posed feedback system, where the closed-loop transfer function is `s+1`, which cannot be represented as a state-space system.
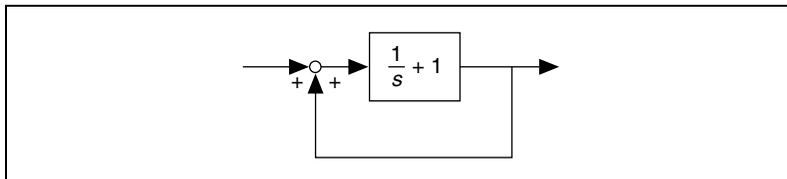
Figure 3-8. Ill-Posed Feedback System

Example 3-4   Example of Closed-Loop System

```
a = 1;
b = [1,0,1];
c = b';
d = [0,0,0;0,0,1;0,1,0];
Sys = SYSTEM(a,b,c,d);
SysC = SYSTEM(-40,2.7,-40,0);
SysCL = clsys(Sys,SysC)
```

SysCL (a state space system) =
** A**
** 1   -40**
** 2.7   -40**

** B**
** 1   0**
** 0   2.7**

** C**
** 1   0**
** 0   -40**

** D**
** 0   0**
** 0   0**

** X0**
** 0**
** 0**

System is continuous

4

Controller Synthesis

This chapter discusses synthesis tools in two categories, H_∞ and H_2. This chapter does not explain all of the theory of H_∞, LQG/LTR, and frequency shaped LQG design techniques. The general problem setup is explained together with known limitations.

H-Infinity Control Synthesis

Problem Definition

The H_∞ control synthesis function `hinfcontr( )` finds a stabilizing multivariable controller K for the plant P, as shown in Figure 4-1.

In the closed-loop system with plant P and controller K, all frequencies ω,

$$\sigma_{max}(H_{ew}(j\omega)) \le \gamma \qquad (4\text{-}1)$$

where H_{ew} is the closed-loop transfer matrix from w to e and γ is some specified limit. Equation 4-1 can be expressed in terms of the H_∞ norm as:

$$\|H_{ew}\|_\infty \le \gamma$$
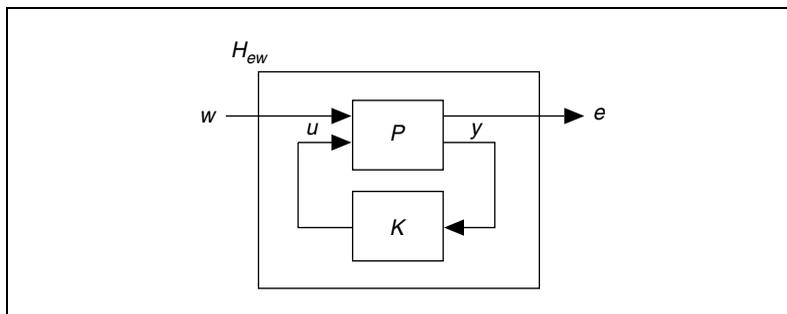
Figure 4-1. Closed-Loop System with Plant P and Controller K

The function `hinfcontr( )` is based on the 2-Riccati state space solutions presented in [GD88,DGKF89]. You can examine these references for theoretical descriptions.

The function `hinfcontr( )` can be used to find an optimal H_∞ controller K that is arbitrarily close to solving:

$$\min_{K} \ \|H_{ew}\|_\infty \le \gamma = \gamma_{opt} \tag{4-2}$$

The `hinfcontr( )` function description in the *hinfcontr( )* section describes how the optimum can be found manually by decreasing γ until an error condition occurs, or conversely by increasing γ until the error condition is fixed.

The particular restrictions, required by the 2-Riccati solutions and summarized in the *Restrictions on the Extended Plant* section are those imposed in [GD88,DGKF89].

Extended Transfer Matrix

Referring to Figure 4-1, plant *P* specifies two groupings of vector inputs and outputs. Such systems or transfer matrices are referred to as *extended transfer matrices* or *systems*. To enter these in Xmath requires a modification of your existing system representation. The standard system has the form $y = G(s)u$ and can be described either in state-space form:

$$x' = Ax + Bu$$
$$y = Ax + Du$$

or as a transfer matrix:

$$G(s) = D + C(sI - A)^{-1}B$$

$G(s)$ can be described in Xmath using the state-space system object:

`G = system(A,B,C,D)`

There is, however, insufficient information in this form to distinguish the input/output groupings in the extended system *P* in Figure 4-1. The state-space form of *P* is:

$$\dot{x} = Ax + B_1 w + B_2 u$$

$$e = C_1 x + D_{11} w + D_{12} u$$

$$y = C_2 x + D_{21} w + D_{22} u$$

Equivalently, as a transfer matrix:

$$P(s) = \begin{bmatrix} D_{11} & D_{12} \\ D_{21} & D_{22} \end{bmatrix} + \begin{bmatrix} C_1 \\ C_2 \end{bmatrix} (sI - A)^{-1} [B_1 \ \ B_2]$$

To enter the extended system, you must know the sizes of e and w shown in Figure 4-1. The extended plant P can be constructed using the Xmath interconnection functions, as shown in Example 4-1.

Building the Plant Model

The general form of the plant P is shown in Figure 4-2.
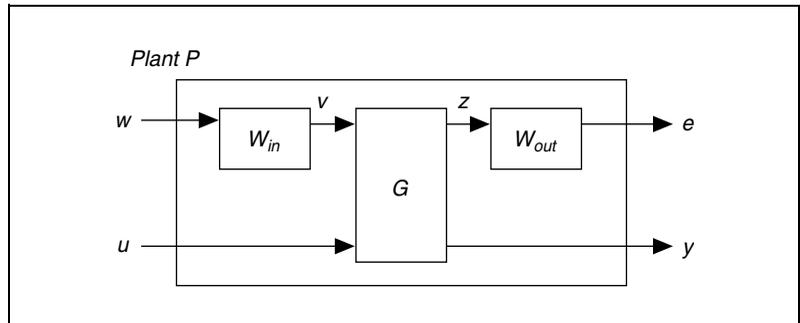
Figure 4-2. Construction of Plant P

The plant consists of three transfer matrices: W_{in} and W_{out}, referred to as weights, and G which can be interpreted as the system dynamics. Both P and G distinguish inputs and outputs into two groups of variables.

The input/output variables are organized as follows:

- actuator/sensor variables

    u—vector of actuator (control) signals

    y—vector of sensor (measured) and other accessible signals

- exogenous inputs

    v—vector of commands and disturbance

    w—vector of "normalized" commands and disturbances

- performance outputs

    z—vector of critical performance signals (regulated variables)

    e—vector of "normalized" critical performance signals

The transfer matrix *G* can be viewed as a model of the underlying system dynamics with *v* and *u* as generalized forces that produce effects in the performance signals *z* and measured signals *y*.

The weight W_{in} is used to model the exogenous input *v* by $v = W_{in}w$.

Similarly, the critical performance variables in the vector *z* are weighted to form the normal critical variables $e = W_{out}z$.

In general, the input weight W_{in} can be viewed as a dynamic model of the exogenous inputs and the output weight W_{out} as the inverse of the desired performance. As an illustration, consider the plant configuration in Figure 4-3.

Figure 4-3. Typical Plant Configuration

The exogenous input vectors *d* and *n* represent disturbances and sensor noise, respectively. These are generated by passing normalized unpredictable signals, ω_{dist} and ω_{noise}, through stable transfer matrices, W_{dist} and W_{noise}, respectively. The critical performance variables are some regulated variables y_{reg}, as well as the actuator commands *u*. These are weighed by the transfer matrices W_{reg} and W_{act} to form the normalized error variables e_{reg} and e_{act}. The sensed variables y_{sens} are contaminated by additive noise *n* to form the measured signal *y*. The transfer matrix G_{dyn} represents the underlying system dynamics. Observe that the transfer matrix *G*, as defined in [BBK88], consists of G_{dyn} with some special output/input connections among the variables *n* and *u* as depicted in Figure 4-3. This is in the form of the familiar LQG setup, except that

here the weighting matrices are transfer matrices, whereas in the LQG setup they are constants.

A description of the plant in Figure 4-3 is as follows:

- Dynamical system G_{dyn}:

$$\dot{x} = Ax + B_{dist}d + B_{act}u$$

$$y_{reg} = C_{reg}x$$

$$y_{sens} = C_{sens}x$$

- Measured variables $y = y_{sens} + n$:
- Input weight W_{in}:

$$\begin{bmatrix} d \\ n \end{bmatrix} = \begin{bmatrix} W_{dist} & 0 \\ 0 & W_{noise} \end{bmatrix} \begin{bmatrix} w_{dist} \\ w_{noise} \end{bmatrix} = W_{in} \begin{bmatrix} w_{dist} \\ w_{noise} \end{bmatrix}$$

- Output weight W_{out}:

$$\begin{bmatrix} e_{reg} \\ e_{act} \end{bmatrix} = \begin{bmatrix} W_{reg} & 0 \\ 0 & W_{act} \end{bmatrix} \begin{bmatrix} y_{reg} \\ u \end{bmatrix} = W_{in} \begin{bmatrix} y_{reg} \\ u \end{bmatrix}$$

Weight Selection

In the standard LQG formulation, the weighting functions (W_{dist}, W_{noise}, W_{reg}, and W_{act}) are all constant matrices. In fact, if Q_{reg} and Q_{act} are positive definite matrices in the LQ cost, for example,

$$J = \frac{1}{2}\left(\int_0^\infty x'Q_{reg}x + u'Q_{act}u \right)dt$$

then the following apply:

$$W_{reg} = Q_{reg}^{1/2} \qquad W_{act} = Q_{act}^{1/2}$$

Similarly, if R_{proc} and R_{sens} are positive definite matrices corresponding to the process and measurement noise intensities, then the following apply:

$$W_{dist} = Q_{proc}^{1/2} \qquad W_{noise} = Q_{sens}^{1/2}$$

Selecting these weights has much the same effect here. Specifically, let H_{zv} be the closed-loop transfer matrix (with $u = K_y$) from inputs:

$$v = \begin{bmatrix} d \\ n \end{bmatrix}$$

to outputs:

$$z = \begin{bmatrix} y_{reg} \\ u \end{bmatrix}$$

Thus,

$$H_{zv} = \begin{bmatrix} H_{y_{reg}d} & H_{y_{reg}n} \\ H_{ud} & H_{un} \end{bmatrix}$$

Suppose that the controller $u = K_y$ approximates Equation 4-2. Thus,

$$\|W_{out} H_{zv} W_{in}\|_\infty \approx \gamma_{opt}$$

In many cases, this means that the maximum singular value of the frequency response matrix $(W_{out}H_{zv}W_{in})(j\omega)$ is constant over all frequencies. That is,

$$\sigma_{max}\left(\begin{bmatrix} W_{reg}H_{y_{reg}d}W_{dist} & W_{reg}H_{y_{reg}n}W_{noise} \\ W_{act}H_{ud}W_{dist} & W_{act}H_{jn}W_{noise} \end{bmatrix}(jw)\right) \approx \gamma_{opt}$$

An interpretation is that the weighting filters W_{in} and W_{out} determine the shape of the closed-loop frequency response $H_{zw}(j\omega)$, and γ_{opt} determines the peak value. This observation helps motivate the selection of the weights so as to shape the closed-loop frequency response matrix $H_{zw}(j\omega)$. Observe, however, that the elements of the frequency response matrix, $(W_{out}H_{zv}W_{in})(j\omega)$, need not be constant. Instead, the maximum singular value of at least one of the four subblocks is within 3 dB of γ_{opt}. For all ω,

$$M(\omega) \le \sigma_{max}[(W_{out}H_{zv}W_{in})(j\omega)] \le \sqrt{2}M(\omega)$$

where

$$M(\omega) = max\{m_{11}(u), m_{12}(u)m_{21}(u)m_{22}(u)\}$$

and

$$m_{11} = \sigma_{max}[W_{reg}H_{y_{reg}d}W_{dist}]$$

$$m_{12} = \sigma_{max}[W_{reg}H_{y_{reg}n}W_{noise}]$$

$$m_{21} = \sigma_{max}[W_{act}H_{ud}W_{dist}]$$

$$m_{22} = \sigma_{max}[W_{act}H_{un}W_{noise}]$$

The weights also can be viewed as "design knobs" (for example, [ONR84]). In this view, the weights are not directly related to specific disturbance or performance models but rather are used as a vehicle to obtain a closed-loop transfer matrix, H_{zv}, from v to z with desired properties. For every selection of weights W_{in} and W_{out}, the closed-loop system has the following property:

$$\|W_{in}H_{zv}W_{out}\|_{\infty} \leq \gamma$$

But H_{zv} has other properties, both good and bad. To some extent, these all can be affected by varying the weights. An effective way to provide a rapid evaluation of performance is with the function perfplots( ), as described in the *perfplots( )* section of Chapter 3, *System Evaluation*. With a few trial and error adjustments of the weights, perfplots( ) will give a good indication of their effect on performance.

Restrictions on the Extended Plant

Not all choices of weights will result in an extended plant $P = W_{out}GW_{in}$ that will solve Equation 4-1. The following conditions, established in references [GD88,DGKF89], if satisfied, will result in a solution. If any are not satisfied, an error condition occurs.

The conditions are:

- (A, B_2, C_2) can be stabilized and detected
- *rank(D₁₂) = NU (number of inputs)*
- *rank(D₂₁) = NY (number of outputs)*

- For all $\omega \geq 0$,

$$rank \begin{bmatrix} A - j\omega I & B_2 \\ C_1 & D_{12} \end{bmatrix} = NS + NU$$

-

$$rank \begin{bmatrix} A - j\omega I & B_1 \\ C_2 & D_{21} \end{bmatrix} = NS + NY$$

Condition 1 is a standard condition to ensure the existence of a stabilizing controller. Condition 2 ensures that the control signal *u* is contained in the normalized error vector *e* (refer to Figure 4-3). Conversely, condition 3 ensures that some exogenous input (disturbance or noise) affects the measured signals (refer to Figure 4-3). Conditions 4 and 5 ensure certain minimal realizations of subblocks of the extended plant ([GD88]).

hinfcontr( )

```
[SysC,Syszw]=hinfcontr(SysAug,gamma,nw,nz,{method})
```

The hinfcontr( ) function designs an H_∞ controller for an augmented plant. The augmented plant should satisfy the five restrictions in the *Restrictions on the Extended Plant* section. The hinfcontr( ) function tests for these restrictions and returns an error if they are violated.

Assuming the restrictions are not violated, a controller satisfying $\|H_{ew}\|_\infty \leq \gamma$ will exist if certain low-level conditions also hold. These involve conditions for the solution of the underlying Riccati equations and conditions for some other constraints. The details can be found in [GD88,DGKF89] and are beyond the scope of this manual. If the low-level conditions are violated, an error statement is displayed:

```
hinfcontr ->No stabilizing controller meets the spec!
   Adjust gamma and try again!
```

When this occurs it means that the original gamma is too small and a larger gamma (for example, a looser spec) is required to eliminate the error condition. If gamma is too small, or any other necessary condition is not met, the hinfcontr( ) function returns a zero controller and the closed-loop system is equal to the open-loop system:

```
SysC = system( [], [], [], 0 ),
Syszw = system(A, B1, C1, D11)
```

If no error message occurs, then $\|H_{ew}\|_\infty \le \gamma$ is guaranteed. However, this does not preclude the possibility that either $\|H_{ew}\|_\infty \ll \gamma$ or that $\gamma_{opt} \ll \|H_{ew}\|\infty$.

For the former case, there are two checks:

- Use the `linfnorm( )` function to compute $\|H_{ew}\|_\infty$.

- Compute the graph $\sigma_{max}[H_{ew}(j\omega)]$ versus ω.

If $\|H_{ew}\|_\infty \ll \gamma$ by about 6 dB or more, then you can decrease `gamma` and try again.

When `gamma` is very large, the specification (Equation 4-1) is easily met. In this case, the `hinfcontr( )` function returns a controller that approximately minimizes the H_2 norm of H_{ew} while satisfying Equation 4-2. Gamma can be interpreted as a "knob" that smoothly transforms the H_2 optimal (LQG) controller, (with `gamma` large), to a H_∞ optimal controller (with `gamma` $\approx \gamma_{opt}$).

Similarly, for a large `gamma`, the controller has good RMS performance with the noise spectra determined by the weights W_{dist} and W_{noise}. For a small `gamma`, the controller has good worst-case performance for noise spectra that lay below the weights W_{dist} and W_{noise}.

Example 4-1   Example of hinfcontr( )

Referring to Figure 4-2, suppose G has the state space description,

$$\dot{x} = x + d + u$$
$$y = x + n$$

where:

$$z = \begin{bmatrix} x \\ u \end{bmatrix} \qquad v = \begin{bmatrix} d \\ n \end{bmatrix}$$

1. The extended system matrix for G is:
   ```
   A = 1;
   B1 = [1,0]; B2 = 1; B = [B1,B2];
   C1 = [1;0]; C2 = 1; C = [C1;C2];
   D11 = zeros(2,2); D12 = [0;1]; D21 = [0,1]; D22 = 0;
   D = [D11,D12; D21,D22];
   G = system(A,B,C,D);
   nw = 2; nz = 2;
   ```

Suppose the input/output weights are as follows:

$$W_{in} = \begin{bmatrix} \dfrac{1}{s+1} & 0 \\ 0 & 0.1 \end{bmatrix} \qquad\qquad W_{out} = \begin{bmatrix} 1 & 0 \\ 0 & 0.1 \end{bmatrix}$$

2. Create the four weights:

```
Wdist = 1/makepoly([1,1],"s")
```

Wdist (a transfer function) =

```
          1
        -----
        s + 1
```

```
Wnoise = 0.1;
Wreg=1/makepoly(1,"s");
Wact = 0.1;
```

3. Combine the weights in W_{in} and W_{out} (refer to Figure 4-4):

```
Win = [Wdist,0,0;0,Wnoise,0;0,0,1];
Wout=[Wreg,0,0;0,Wact,0;0,0,1];
```

The resulting system, P can be obtained by putting in series the plant *G* and the two weights:

```
P = Wout*G*Win;
```

Before using the hinfcontr( ) function, you must decide on an initial guess for gamma.
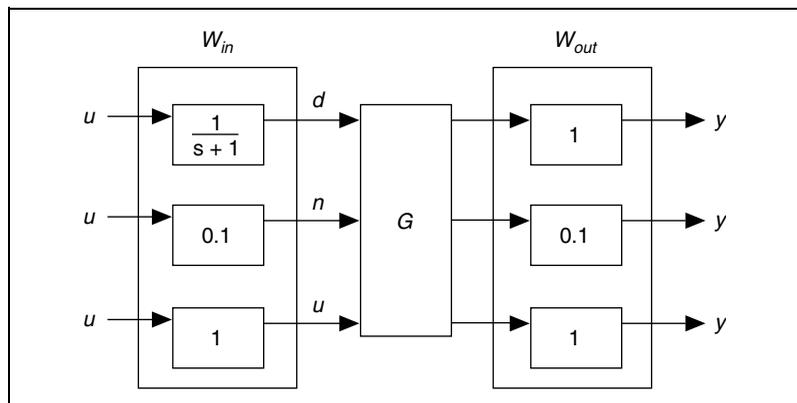
Figure 4-4. Plant and Weights for hinfcontr( )

4. For this example, you will start with gamma=1 as the initial guess and
    enter:

    ```
    [K,Hew] = hinfcontr(P,1,2,2);
    ```

    No error messages are reported. This means that a stabilizing
    controller has been found such that Equation 4-1 holds. That is,
    $\|H_{ew}\|_\infty \le 1$.

    The actual $H\infty$ norm is found from:

    ```
    normHew=linfnorm(Hew)
    ```

    normHew (a scalar) =   0.211984

    The result is that on this first iteration:

    ```
    gamma = 1 Æ   normHew = 0.212
    ```

    Continuing to iterate for the optimal gamma:

    ```
    [K,Hew] = hinfcontr(P,.2,2,2);
    normHew=linfnorm(Hew)
    ```

    normHew (a scalar) =   0.173218

    ```
    [K,Hew] = hinfcontr(P,.15,2,2);
    normHew=linfnorm(Hew)
    ```

    normHew (a scalar) =   0.147418

    ```
    [K,Hew] = hinfcontr(P,.13,2,2);
    normHew=linfnorm(Hew)
    ```

    normHew (a scalar) =   0.13103

    ```
    [K,Hew] = hinfcontr(P,.12,2,2);
    normHew=linfnorm(Hew)
    ```

    normHew (a scalar) =   2.02252

    **hinfcontr -->   No stabilizing controller meets the
    spec.!!**
        Adjust gamma and try again

    The iterations establish that γ_{opt} lies between 0.12 and 0.13. Figure 4-5
    shows the output of the perfplots function on the closed-loop
    system H_{ew} for $\gamma = 0.13$.

    ```
    [K,Hew] = hinfcontr(P,.13,2,2);
    normHew = linfnorm(Hew, {tol=1e-3})
    ```

    normHew (a scalar) =   0.129863

    ```
    svHew = perfplots(Hew,1,1,{Fmin=0.01,Fmax=100});
    ```
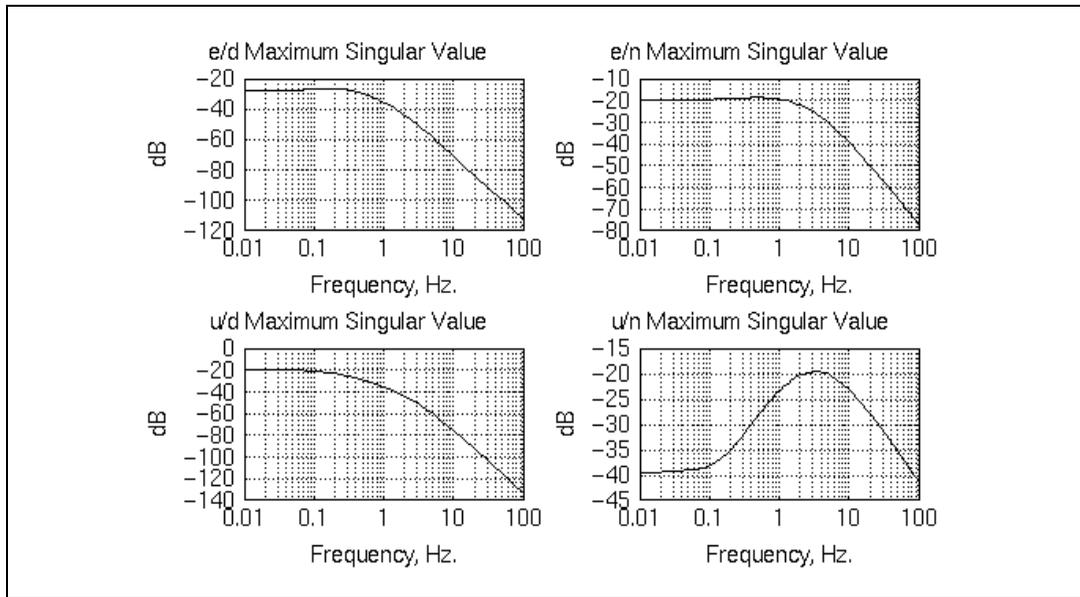
Figure 4-5. Perfplots for H_{ew}

It also is useful to perform `perfplots( )` on the unweighted closed-loop system, H_{zv}, which in this case is the closed-loop transfer matrix from (d,n) into (x,u).

The following function calls produce Figure 4-6:

```
Hzv=clsys(G,K);
Hzv=perfplots(Hzv,1,1,{vF=domain(svHew)});
```
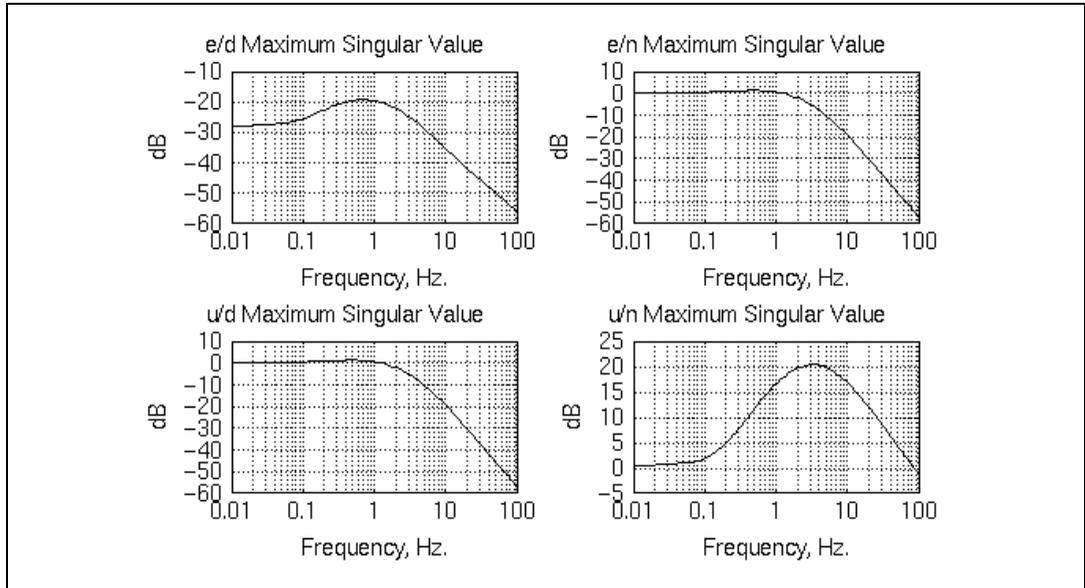
Figure 4-6. Perfplots for H_{zv}

singriccati( )

```
[P, solstat] = singriccati(A,Q,R {method})
```

The singriccati( ) function solves the Indefinite Algebraic Riccati Equation (ARE):

$$A'P + PA - PRP + Q = 0$$

The ARE is solved by decomposing the Hamiltonian:

$$\begin{bmatrix} A & -R \\ -Q & -A' \end{bmatrix}$$

The required decomposition of the Hamiltonian can be achieved using method="eig" or method="schur". The Schur decomposition is slower, but might handle some ill-conditioned problems. After solving the ARE, singriccati( ) calculates the residue ($A^T P + PA - PRP + Q$). A warning is displayed if the residue is large. For an example of this function, refer to the *Xmath Help*.

Linear-Quadratic-Gaussian Control Synthesis

The H_2 Linear-Quadratic-Gaussian (LQG) control design methods are based on minimizing a quadratic function of state variables and control inputs. Conventionally, the problem is specified in the time domain. By converting the LQG performance index into the frequency domain, it becomes obvious that the conventional LQG places equal penalty on states and control inputs at all frequencies. It is possible to realize significant improvement in robustness and performance by making the penalty weighting matrices functions of frequency.

LQG Frequency Shaping

Bryson's rule [BH69] can be extended to initially select a frequency shaping for a particular problem. For the control design problem, the frequency-shaped weighting matrices should be large at frequencies where control inputs are less desirable. For example, a large weighting on control signals at high frequency would produce less control activity at those frequencies, leading to a closed-loop system with lower bandwidth. Similar ideas apply to selection of state weighting in control design and the development of robust state estimators.

Three functions are available to solve the problem of frequency-shaped cost functionals:

- `fsregu( )`—frequency-shaped regulator
- `fsesti( )`—frequency-shaped estimator
- `fslqgcomp( )`—frequency-shaped linear-quadratic-gaussian compensator

fsregu( )

```
[SysC, SysCC, vEV] = fsregu(SysA, ns, RXXA, RUUA, {RXUA})
```

The `fsregu( )` function computes a frequency-shaped control law. It assumes you start with:

$$\dot{x} = Ax + Bu$$

and

$$J = \frac{1}{2} \int_{-\infty}^{\infty} [x^*(j\omega)Q(j\omega)x(j\omega) + u^*(j\omega)R(j\omega)u(j\omega)]d\omega$$

This expression can be converted into the following form [Gu80]:

$$
\begin{bmatrix} \dot{x} \\ \dot{x}' \end{bmatrix} = \underbrace{\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}}_{A_c} \begin{bmatrix} x \\ x' \end{bmatrix} + \underbrace{\begin{bmatrix} B_1 \\ B_2 \end{bmatrix}}_{B_c} v
$$

$$
u = \underbrace{\begin{bmatrix} 0 & C_{12} \end{bmatrix}}_{C_c} \begin{bmatrix} x \\ x' \end{bmatrix} + \underbrace{D}_{D_c} v
$$

If $R(j\omega)$ is not a function of frequency, then $C_{12} = 0$ and $D = I$.

Note   The system has a new input v and the old input u is now the output of the system. This structure is only used for computational convenience.

Define:

$$
x_A = \begin{bmatrix} x \\ x' \end{bmatrix}
$$

Thus,

$$
\dot{x}_A = A_c x_A + B_c v
$$

$$
u = C_c x_A + D_c v
$$

$$
J = \frac{1}{2} \int^{\infty} (x_A^* R_{xx_A} x_A + v^* R_{uu_A} v + 2 x_A^* R_{xu_A} v) d\omega
$$

After the system is put in this form, you are ready to use the `fsregu( )` function. For more information on the `fsregu( )` syntax, refer to the *Xmath Help*.

fsesti( )

```
[SysF, vEV] = fsesti(SysA, ns, QWWA, QVVA, {QWVA})
```

The `fsesti( )` function computes a frequency-shaped state estimator. The estimation problem is stated as follows:

$$\dot{x} = Ax + Bu + w$$
$$y = Cx + Du + v$$

The frequency-shaped filter design problem is to minimize,

$$J = \frac{1}{2} \int_{-\infty}^{\infty} (w^* Q_{ww}(j\omega)w + v^* Q_{vv}(j\omega)v)d\omega$$

which can be written as:

$$\dot{x}_A = A_e x_A + B_e u + w_A$$

or it can be written as:

$$\begin{bmatrix} \dot{x} \\ \dot{x}' \end{bmatrix} = \underbrace{\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}}_{A_e} \underbrace{\begin{bmatrix} x \\ x' \end{bmatrix}}_{x_A} + \underbrace{\begin{bmatrix} B \\ 0 \end{bmatrix}}_{B_e} u + w_A$$

$$u = \left(\underbrace{\begin{bmatrix} C_1 & C_2 \end{bmatrix}}_{C_e} \right) \begin{bmatrix} \dot{x} \\ \dot{x}' \end{bmatrix} + D_e u + v_A$$

$$J = \frac{1}{2} \int_{-\infty}^{\infty} w_A^* Q_{ww_A} w_A + v_A^* Q_{vv_A} v_A + 2w_A^* Q_{wv_A} v)d\omega$$

For more information on the `fsesti( )` syntax, refer to the *Xmath Help*.

fslqgcomp( )

```
[SysCC, vEV] = fslqgcomp(SysF, SysC)
```

The `fslqgcomp( )` function combines filter and control law to compute a
controller from a control law and an estimator. For more information on the
`fslqgcomp( )` syntax, refer to the *Xmath Help*.

Frequency-Shaped Control Design Commands

This extended example uses the previously discussed functions to
demonstrate frequency-shaped control design techniques. A four-state,
poorly damped system is studied to demonstrate how robustness can be
attained using frequency shaping. The control law and filter are designed
on a reduced second order system with and without frequency shaping.

1. Create a full-order system:

    ```
    a=[0,1,0,0;-1,-.01,0,0;0,0,0,1;0,0,-25,-.05];
    b=[0,1,0,1]';
    c=[0,1,0,-1]; d=0;
    Sys=system(a,b,c,d);
    ```

2. Calculate open-loop eigenvalues:

    ```
    eig(a)
    ```

    ans (a column vector) =

    ```
        -0.005 + 0.999987 j
        -0.005 - 0.999987 j
        -0.025 + 4.99994  j
        -0.025 - 4.99994  j
    ```

3. Create a reduced order system by selecting only the first mode:

    ```
    ar=a(1:2,1:2);br=b(1:2);cr=c(1:2);
    Sysr=system(ar,br,cr,d);
    ```

4. Design an LQG compensator for the reduced-order system, without
    using frequency shaping:

    ```
    qxx=diagonal([0,1]);quu=1;
    kr=regu(Sysr,qxx,quu);      # Linear-Quadratic-Regulator
    ke=esti(Sysr,qxx,quu);      # Linear-Quadratic-Estimator
    Sysc=lqgcomp(Sysr,kr,ke);   # LQG Compensator
    Syscl=feedback(Sysr,Sysc);  # Reduced-order closed-loop
    poles(Syscl)
    ```

    ans (a column vector) =
    ```
      -0.500025 + 0.866011 j
      -0.500025 - 0.866011 j
    ```

```
-0.500025 + 0.866011 j
-0.500025 - 0.866011 j
```

5. Try the LQG compensator with the full-order system:

```
[Syscl_fo]=feedback(Sys,Sysc);

poles(Syscl_fo)
```

ans (a column vector) =
```
-0.401519  + 0.864869 j
-0.401519  - 0.864869 j
-0.638796  + 0.855861 j
-0.638796  - 0.855861 j
 0.0152647 + 4.90994  j
 0.0152647 - 4.90994  j
```

6. Find a stabilizing reduced order controller using frequency shaping.

   To stabilize this system, try a frequency-shaped control-law, where the weighting on the control signal will be $1 + (\omega)^4$.

   First, form an augmented system, defining two additional states, μ and $\dot{u}$.

   The augmented system will be:

$$\frac{d}{dt}\begin{bmatrix} x_r \\ u \\ \dot{u} \end{bmatrix} = \begin{bmatrix} A_r & B_r & 0 \\ & & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}\begin{bmatrix} x_r \\ u \\ \dot{u} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}v = A_a x_A + B_a v \qquad (4\text{-}3)$$

$$u = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} x_r \\ u \\ \dot{u} \end{bmatrix} + (0)v = C_a x_A + D_a v \qquad (4\text{-}4)$$

Note The augmented system has two extra states implementing the frequency shaping on the control system.

```
aa=[ar,br,[0;0];0,0,0,1;0,0,0,0];
bb=[0,0,0,1]';cc=[0,0,1,0];dd=0;
Sysa=system(aa,bb,cc,dd)
```

Sysa (a state space system) =
```
  A
  0     1      0    0
 -1   -0.01    1    0
```

```
   0        0          0     1
   0        0          0     0
  B
  0
  0
  0
  1
  C
  0      0     1      0
  D
  0
  X0
  0
  0
  0
  0

  System is continuous
```

7. Frequency-weight the control signal.

   Transfer the weight on U from RUU to the third diagonal entry in RXXA.

📝 **Note** In Equation 4-3, *u* is the third state of the augmented system. RUU weighs the new frequency-shaped control signal *v*.

Design a frequency-shaped regulator:

```
rxxa=diagonal([0,1,1,0]);ruua=1;
[Sysfs_sr,,fs_evr]=fsregu(Sysa,2,rxxa,ruua)
```

```
Sysfs_sr (a state space system) =
  A
   0              1
  -1.95171    -1.97571
  B
  0              0
  0.951712    -0.228069
  C
  1     0
  D
  0     0
  X0
  0
  0
```

```
    System is continuous

fs_evr (a column vector) =
  -0.645263 + 0.587929 j
  -0.645263 - 0.587929 j
  -0.347592 + 1.09155  j
  -0.347592 - 1.09155  j
```

8. Calculate the frequency-shaped estimator:

```
Sysaf=system(ar,br,cr,0);qwwa=qxx;qvva=quu;
[Sysfs_se,fs_eve]=fsesti(Sysaf,2,qwwa,qvva)

Sysfs_se (a state space system) =
  A
   0      1
  -1     -1.00005

  B
  5.52357e-17    0
  0.99005        1

  C
  1     0
  0     1

  D
  0     0
  0     0

  X0
  0
  0

  System is continuous

fs_eve (a column vector) =
  -0.500025 + 0.866011 j
  -0.500025 - 0.866011 j
```

The compensator should be structured as shown in Figure 4-7.
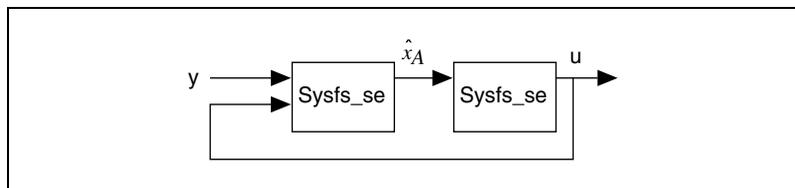
Figure 4-7. Frequency-Shaped Compensator

The fslqgcomp( ) function can be used to develop the compensator.

9. Design the LQG compensator.

```
[Sysfs_sc,fs_evc]=fslqgcomp(Sysfs_se,Sysfs_sr)
```

Sysfs_sc (a state space system) =

A

0	1	0	0
-1	-1.00005	1	0
0	0	0	1
0.951712	-0.228069	-1.95171	-1.97571

B
5.52357e-17
0.99005
0
0

C
0   0   1   0

D
0

X0
0
0
0
0

System is continuous

fs_evc (a column vector) =
-0.373302
-1.17564
-0.713411 + 1.33028 j
-0.713411 - 1.33028 j

Enforce negative feedback:

```
Sysfs_sc = -Sysfs_sc;
```

10. Compute the closed-loop system for the reduced order plant and the frequency-shaped compensator:

```
[Sysfs_scl]=feedback(Sysr,Sysfs_sc);
poles(Sysfs_scl)
```

ans (a column vector) =
** -0.645263 + 0.587929 j**
** -0.645263 - 0.587929 j**
** -0.500025 + 0.866011 j**
** -0.500025 - 0.866011 j**
** -0.347592 + 1.09155 j**
** -0.347592 - 1.09155 j**

11. Compute the closed-loop system for the full-order plant and the frequency-shaped compensator.

```
Sysfs_scl_fo = feedback(Sys,Sysfs_sc);
poles(Sysfs_scl_fo)
```

ans (a column vector) =
** -0.690216 + 0.522898 j**
** -0.690216 - 0.522898 j**
** -0.419783 + 0.892632 j**
** -0.419783 - 0.892632 j**
** -0.381722 + 1.10668 j**
** -0.381722 - 1.10668 j**
** -0.0261589 + 5.00027 j**
** -0.0261589 - 5.00027 j**

The full-order closed-loop system is stable. The open-loop eigenvalues of the unmodelled mode have not moved much, which is a sign of good robustness. The eigenvalue of the unmodelled mode changed from $-.0250 \pm 5j$ to $-0.0262 \pm 5j$.

Loop Transfer Recovery (lqgltr)

Loop transfer recovery (LTR) is fully described in references [KS72, DoS79,DoS81,SA88]. The properties of the recovery pertain to the LQG feedback system as shown in Figure 4-8.

The parameter ρ (rho) can be manipulated by the user to obtain loop transfer recovery through the regulator (lqrltr) or the estimator (lqeltr).
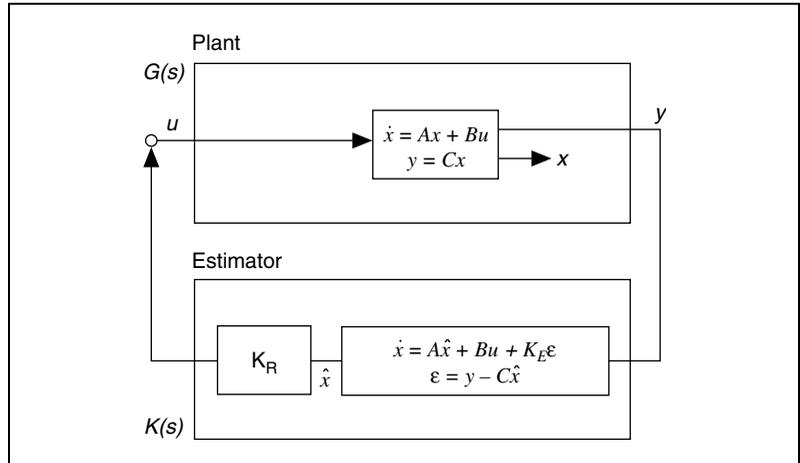
Figure 4-8. LQG Feedback System for Loop Transfer Recovery

lqgltr( )

```
[SysC,EV,Kr] = lqgltr(Sys,Wx,Wy,K,rho,{keywords})
```

The lqgltr( ) function designs an estimator or regulator which *recovers* loop transfer robustness through the design parameter ρ (rho). For a discussion of the syntax and a full listing of keywords, refer to the *Xmath Help*.

The keyword recover specifies whether the recovery should be achieved through regulator or estimator design.

If the keyword is set to recover="regulator", the loop-transfer is recovered by designing a regulator with the following model:

$$\dot{x} = Ax + w$$
$$y = Cx + v$$

and the objective function:

$$J = \frac{1}{2}\int_0^\infty (x'Qx + u'Ru)dt$$

with:

$$Q = R_{xx} + \rho C'C$$
$$R = R_{uu}$$

Then ρ is increased so that pointwise in *s*:

$$K(s)G(s) \rightarrow K_R(sI - A)^{-1}B$$

Regulator recovery is only guaranteed if *G(s)* is minimum-phase and there are at least as many control signals *u* as measurements *y*.

If recover="estimator", the loop-transfer is recovered by designing an estimator with the following model:

$$\dot{x} = Ax + w$$
$$y = Cx + v$$

where *w* and *v* have noise intensities:

$$W = Q_{xx} + \rho BB'$$
$$V = Q_{yy}$$

Then ρ is increased so that pointwise in *s*:

$$G(s)K(s) \rightarrow C(sI - A)^{-1}K_E$$

Recovery is only guaranteed if *G(s)* is minimum-phase and there are at least as many measurements *y* as there are control signals *u*.

If graph is on (default) the singular value loop transfer response is plotted. The solid line represents the original loop transfer, and the dashed line represents the loop transfer after recovery.

A

Bibliography

[BBK88]   S. Boyd, V. Balakrishnan, and P. Kabamba. "A bisection method for computing the L_∞ norm of a transfer matrix and related problems." *Mathematical Control Signals, Systems* Vol. 2, No. 3, pp 207–219, 1989.

[BeP79]   A. Berman and R.J. Plemmons. "Nonnegative Matrices in the Mathematical Sciences." *Computer Science and Applied Mathematics Series*, Academic Press, 1979.

[BoB90]   S. Boyd and V. Balakrishnan. "A regularity result for the singular values of a transfer matrix and a quadratically convergent algorithm for computing its L_∞ norm." *Systems Control Letters*, Vol. 15, pp 1–7, 1990.

[BoB91]   S. Boyd and C. Barratt. *Linear Controller Design: Limits of Performance.* Prentice-Hall, 1991.

[BH69]   A.E. Bryson and Y.C. Ho. *Applied Optimal Control*, p 149. Blaisdell Publishing Co., 1969.

[DoS79]   J.C. Doyle and G. Stein. "Robustness with Observers." *IEEE Transactions on Automatic Control*, August 1979.

[DoS81]   J.C. Doyle and G. Stein. "Multivariable Feedback Design: Concepts for a Classical/Modern Synthesis." *IEEE Transactions on Automatic Control*, Vol. AC-26, pp 4–16, February 1981.

[Doy82]   J.C. Doyle. "Analysis of Feedback Systems with Structured Uncertainties." *IEEE Proceedings*, November 1982.

[DWS82]   J.C. Doyle, J.E. Wall, and G. Stein. "Performance and Robustness Analysis for Structure Uncertainties." Proceedings *IEEE Conference on Decision and Control*, pp 629–636, 1982.

[FaT88]   M.K. Fan and A.L. Tits. "m-form Numerical Range and the Computation of the Structured Singular Value." *IEEE Transactions on Automatic Control*, Vol. 33, pp 284–289, March 1988.

[FaT86]   M.K. Fan and A.L. Tits. "Characterization and Efficient Computation of the Structured Singular Value." *IEEE Transactions on Automatic Control*, Vol. AC-31, pp 734–743, August 1986.

[Fr87]   B. Francis. *A Course in L_∞ Control Theory*. Springer-Verlag, Berlin-New York, 1987.

[FPGM87]   D.S. Flamm, S. Boyd, G. Stein, and S.K. Mitter. "Tutorial Workshop on L_∞ Control Theory." pre-conference workshop, *Proceedings 26th IEEE Conference on Decision and Control*, December 1988.

[GD88]   K. Glover and J.C. Doyle. "State-space formulae for all stabilizing controllers that satisfy an L_∞ norm bound and relations to risk sensitivity." *Systems and Control Letters*, Vol. 11, pp 167–172, 1988.

[DGKF89]   J.C. Doyle, K. Glover, P.K. Khargonekar, and B. Francis. "State-space solutions to standard H_2 and L_∞ control problems." *IEEE Transactions on Automatic Control*, Vol. AC-34, No. 8, pp 831–847, August 1989.

[Gu80]   N.K. Gupta. "Frequency Shaping of Cost Functionals: An extension of LQG Design Methods." *AIAA Journal of Guidance and Control*, Vol. 3, No. 6, December 1980.

[ONR84]   *ONR/Honeywell Workshop on Advances in Multivariable Control*. Lecture Notes, Minneapolis, MN, 1984.

[Osb60]   E.E. Osborne. "On Preconditioning of Matrices." *JACM*, 7:338–345, 1960.

[Saf82]   M.G. Safonov. "Stability Margins of Diagonally Perturbed Multivariable Feedback Systems." *IEEE Proceedings*, 129-D:251–256, November 1982.

[SD83]   M.G. Safonov and J.C. Doyle. "Optimal Scaling for Multivariable Stability Margin Singular Value Computation." *Proceedings of MECO/EES 1983*, *Symposium*, 1983.

[SD84]   M.G. Safonov and J.C. Doyle. "Minimizing Conservativeness of Robust Singular Values." *Multivariable Control*, pp 197–207, S.G. Tzafestas, editor. D. Reidel Publishing Company, 1984.

[SLH81]   M.G. Safonov, A.J. Laub, and G.L. Hartmann. "Feedback Properties of Multivariable Systems: The Role and Use of the Return Difference Matrix." *IEEE Transactions on Automatic Control*, Vol. AC-26, February 1981.

[SA88]   G. Stein and M. Athans. "The LQG/LTR Procedure for Multivariable Control Design." *IEEE Transactions on Automatic Control*, Vol. AC-32, No. 2, pp 105–114, February 1987.

[Za81]   G. Zames. "Feedback and optimal sensitivity: model reference transformations, multiplicative semi-norms, and approximate inverses." *IEEE Transactions on Automatic Control*, Vol. AC-26, pp 301–320, 1981.

[KS72]   H. Kwakernaak and R. Sivan. *Linear Optimal Control Systems*. Wiley, 1972.

B

Technical Support and Professional Services

Visit the following sections of the National Instruments Web site at `ni.com` for technical support and professional services:

- **Support**—Online technical support resources at `ni.com/support` include the following:

  - **Self-Help Resources**—For answers and solutions, visit the award-winning National Instruments Web site for software drivers and updates, a searchable KnowledgeBase, product manuals, step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on.

  - **Free Technical Support**—All registered users receive free Basic Service, which includes access to hundreds of Application Engineers worldwide in the NI Discussion Forums at `ni.com/forums`. National Instruments Application Engineers make sure every question receives an answer.

    For information about other technical support options in your area, visit `ni.com/services` or contact your local office at `ni.com/contact`.

- **Training and Certification**—Visit `ni.com/training` for self-paced training, eLearning virtual classrooms, interactive CDs, and Certification program information. You also can register for instructor-led, hands-on courses at locations around the world.

- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, National Instruments Alliance Partner members can help. To learn more, call your local NI office or visit `ni.com/alliance`.

If you searched `ni.com` and could not find the answers you need, contact your local office or NI corporate headquarters. Phone numbers for our worldwide offices are listed at the front of this manual. You also can visit the Worldwide Offices section of `ni.com/niglobal` to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

Index

A

Algebraic Riccati Equation (ARE), 4-13

C

clsys( ), 3-10
conventions used in the manual, *iv*

D

diagnostic tools (NI resources), B-1
documentation
    conventions used in the manual, *iv*
    NI resources, B-1
drivers (NI resources), B-1

E

examples (NI resources), B-1
extended transfer matrices, 4-2

F

frequency-shaped
    control law, 4-14
    filter design, 4-16
fsesti( ), 4-16
fslqgcomp( ), 4-17
fsregu( ), 4-14

H

$H\infty$ control synthesis, 4-1
    building the plant model, 4-3
    extended transfer matrix, 4-2

    problem definition, 4-1
    restrictions on extended plant, 4-7
    weight selection, 4-5
H_2 control synthesis, 4-14
H_2 norm, 3-4
help, technical support, B-1
hinfcontr( ), 4-2, 4-8

I

instrument drivers (NI resources), B-1

K

KnowledgeBase, B-1

L

$L\infty$ norm, 3-3
linfnorm( ), 3-4
loop transfer recovery, 4-22
LQG frequency shaping, 4-14
lqgltr( ), 4-23

M

MATRIXx Help, 1-3
modeling uncertain systems, 2-1

N

National Instruments support and services, B-1
NI support and services, B-1
nomenclature, 1-2
nominal closed-loop system, 2-2
    creating, 2-4