μ-Analysis and Synthesis Toolbox

For Use with MATLAB®

Gary J. Balas
John C. Doyle
Keith Glover
Andy Packard
Roy Smith

Computation

Visualization

Programming

User's Guide

Version 3

The MathWorks

How to Contact The MathWorks:

💻	`www.mathworks.com`	Web
	`comp.soft-sys.matlab`	Newsgroup
@	`support@mathworks.com`	Technical support
	`suggest@mathworks.com`	Product enhancement suggestions
	`bugs@mathworks.com`	Bug reports
	`doc@mathworks.com`	Documentation error reports
	`service@mathworks.com`	Order status, license renewals, passcodes
	`info@mathworks.com`	Sales, pricing, and general information
☎	508-647-7000	Phone
📠	508-647-7001	Fax
✉	The MathWorks, Inc. 3 Apple Hill Drive Natick, MA 01760-2098	Mail

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

μ-Analysis and Synthesis Toolbox User's Guide

© COPYRIGHT 1993 - 2001 by MUSYN Inc. and The MathWorks, Inc.

The software described in this document is furnished under a license agreement. The software may be used or copied only under the terms of the license agreement. No part of this manual may be photocopied or reproduced in any form without prior written consent from The MathWorks, Inc.

FEDERAL ACQUISITION: This provision applies to all acquisitions of the Program and Documentation by or for the federal government of the United States. By accepting delivery of the Program, the government hereby agrees that this software qualifies as "commercial" computer software within the meaning of FAR Part 12.212, DFARS Part 227.7202-1, DFARS Part 227.7202-3, DFARS Part 252.227-7013, and DFARS Part 252.227-7014. The terms and conditions of The MathWorks, Inc. Software License Agreement shall pertain to the government's use and disclosure of the Program and Documentation, and shall supersede any conflicting contractual terms or conditions. If this license fails to meet the government's minimum needs or is inconsistent in any respect with federal procurement law, the government agrees to return the Program and Documentation, unused, to MathWorks.

MATLAB, Simulink, Stateflow, Handle Graphics, and Real-Time Workshop are registered trademarks, and Target Language Compiler is a trademark of The MathWorks, Inc.

Other product or brand names are trademarks or registered trademarks of their respective holders.

Printing History:	July 1993	First printing
	November 1995	Second printing
	June 1998	Third printing
	January 2001	Fourth Printing for Version 3.05 (Release 12)
	June 2001	Online only for Version 3.06 (Release 12.1)

Contents

1 Overview of the Toolbox

2 Working with the Toolbox

3 *H*∞ Control and Model Reduction

4 Modeling and Analysis of Uncertain Systems

5 Control Design via μ Synthesis

6 Graphical User Interface Tools

7 Robust Control Examples

8 Reference

1

Overview of the Toolbox

The μ-Analysis and Synthesis Toolbox (μ-Tools) is a collection of functions (commands) developed primarily for the analysis and synthesis of control systems, with an emphasis on quantifying the effects of uncertainty. μ-Tools provides a consistent set of data structures for the unified treatment of systems in either a time domain, frequency domain, or state-space manner. μ-Tools also gives MATLAB® users access to recent developments in control theory, namely H_∞ optimal control and m analysis and synthesis techniques. This package allows you to use sophisticated matrix perturbation results and optimal control techniques to solve control design problems. Control design software, such as μ-Tools, provides a link between control theory and control engineering.

Computational algorithms for the structured singular value, μ, are main features of the toolbox. μ is a mathematical object developed to analyze the effect of uncertainty in linear algebra problems. μ is particularly (though not exclusively) useful in analyzing the effect of parameter uncertainty and unmodeled dynamics on the stability and performance of multiloop feedback systems. μ-Tools is a collection of tools designed to help you *analyze* the sensitivity of closed-loop systems to detailed and complex types of modeling errors. μ-Tools is also suitable to *design* control systems that are insensitive to classes of variations that you expect between your model and the actual physical process which must be controlled.

The μ framework appropriately generalizes notions such as gain margin, phase margin, disturbance attenuation, tracking, and noise rejection into a common framework suitable for analysis and design, in both single-loop and multiloop feedback systems. Even when working with single-loop feedback systems, some multi-input, multi-output (MIMO) systems arise during the analysis. Hence, a unified framework to deal with MIMO linear systems is important, with full support for both the time and frequency domain. μ-Tools provides the capability to build complex interconnections (such as cascade, parallel, and feedback connections), compute properties (such as poles and zeros), calculate time and frequency responses, manipulate these responses (FFT for the time domain signals, Bode analysis for the frequency domain functions), and plot results. μ-Tools supports two data types in addition to the standard matrices: SYSTEM matrices for state-space realizations and VARYING matrices for time and frequency responses.

The advanced features of μ-Tools are aimed at:

- Analyzing the effect of uncertain models on achievable closed-loop performance
- Designing controllers for optimal worst-case performance in the face of the plant uncertainty

Hence, it is imperative that you understand the following:

- The characterization of “good” closed-loop performance
- How to represent model uncertainty in this framework
- The technical tools available to answer questions about the robustness of a given closed-loop system to certain forms of model uncertainty
- The technical tools available to design controllers which achieve good performance in the face of the model uncertainty

The characterization of performance is discussed in Chapter 3. In Chapter 4, we concentrate on modeling uncertainty, and the effect it can have on the guaranteed performance level of the closed-loop system. The tools for design are discussed in the latter part of Chapter 3 and in Chapter 5. Chapter 6 presents graphical user interfaces for the workspace, control design and time simulation. Chapter 7 contains a number of examples to show how to apply μ-Tools to robust control problems.

Organization of This Manual

The contents of this manual are not intended to be read in sequence. The chapters are grouped by topic, and coverage within sections varies from syntax descriptions to theoretical justifications.

After skimming the sections on command use and syntax, we feel that the examples are the quickest manner in which you can get a feel for the techniques. However, it may be necessary to refer back to the concept sections as different topics become relevant.

For basic use of the toolbox (system interconnections, system calculations, frequency responses, time responses and plotting), the recommended sections are shown below.

Emphasis	Topic	Pages
Use	Working with the Toolbox	Chapter 2
Examples	SISO Gain and Phase Margins MIMO Loop-at-a-Time Margins	Chapter 7
GUI	Workspace Tool LFT Time Simulation Tool	Chapter 6

For *robust stability analysis*, additional recommended reading is

Emphasis	Topic	Pages
Concepts	Modeling Uncertainty Robust Stability Analysis	Chapter 4
Examples	SISO Gain and Phase Margins MIMO Loop-at-a-Time Margins MIMO Margins Using μ Space Shuttle Robustness Analysis Two Tank System	Chapter 7
GUI	LFT Time Simulation Tool	Chapter 6

For *robust performance analysis*, additional recommended reading is

Emphasis	**Topic**	**Pages**
Concepts	Performance Objectives Robust Performance Analysis	Chapter 3 Chapter 4
Examples	Unstable SISO Analysis HIMAT Robust Performance F-14 Lateral-directional Control Space Shuttle Robustness Analysis Two Tank System	Chapter 7
GUI	LFT Time Simulation Tool	Chapter 6

For *robust control design*, additional recommended reading is

Emphasis	**Topic**	**Pages**
Concepts	H_∞ Performance Objectives H_∞ Control Design H_∞ Loop Shaping μ Upper Bound Robust Control Design	Chapter 3 Chapter 4 Chapter 5
Examples	HIMAT Robust Performance Design F-14 Lateral-Directional Control Design Space Shuttle Robustness Analysis Two Tank System	Chapter 7
GUI	D-K Iteration Tool	Chapter 6

Advanced topics are

Emphasis	Topic	Pages
Concepts	Control Theory	Chapter 3
	Discrete-time and Sampled-Data Control	
	Model Reduction	Chapter 4
	Structured Singular Value Theory	

There are three graphical user interfaces, described in detail in Chapter 6. The interfaces are:

- `wsgui` is a Workspace Manager. It allows you to select, save and clear the workspace variables, based on their type (VARYING, SYSTEM, CONSTANT) and other more complicated selection rules. This tool is useful during all MATLAB sessions, and is described in the “Workspace User Interface Tool: wsgui” section of Chapter 6.
- `simgui` is a time-domain simulation package for uncertain closed-loop systems. It is powerful enough to build templates for the complex plotting requirements of a large MIMO control design report. This tool is described in “LFT Time Simulation User Interface Tool: simgui” section of Chapter 6.
- `dkitgui` is a control design program to assist you with the DK iteration. It aids in understanding the DK iteration process. The flexibility allows you to easily modify performance objectives and uncertainty models during the iteration. This tool is described in “DK Iteration User Interface Tool: dkitgui” section of Chapter 6.

2

Working with the Toolbox

This chapter gives a basic introduction, with examples, to the μ-Analysis and Synthesis Toolbox (μ-Tools) data structure and commands. Introductory examples are found in the demo programs `msdemo1.m` and `msdemo2.m`. Other demonstration files are introduced in subsequent chapters. You can copy these files from the `mutools/subs` source into a local directory and examine the effects of modifying some of the commands.

Command Line Display

All μ-Tools commands have a built-in use display. Any command called with no input arguments, or the incorrect number of input arguments results in a brief description of the correct command line use. For example, at the command line

```
mu
usage: [bonds,dvec,sens,pvec] = mu(matin,blk,opt)
```

The Data Structures

μ-Tools represents systems (either in state-space form or as frequency dependent input/output data) as single data entries. Data structures This allows you to have all of the information about a system in a single MATLAB variable. In addition, the μ-Tools functions that return a single variable can be nested, allowing you to build complex operations out of a few nested operations. Examples of this are found throughout this chapter.

The layout of the data structure is quite simple. Consider a typical MATLAB matrix, which is usually made up of real and complex numbers. In addition to these, MATLAB also allows for a few special values, such as `NaN` (not a number), `Inf` (infinity), and `-Inf`. Since these are allowable values, but are not typically found in state-space realizations, frequency responses, or time responses, they can be used to differentiate more complicated data types from plain, constant matrices. This is the approach taken by μ-Tools.

SYSTEM Matrices

Consider a linear, finite dimensional system, modeled by the state-space representation

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

If the system $\mathbf{R}$ has n_x states, n_u inputs, and n_y outputs, then $A \in \mathbf{R}^{n_x \times n_x}$, $B \in \mathbf{R}^{n_x \times n_u}$, $C \in \mathbf{R}^{n_y \times n_x}$, and $D \in \mathbf{R}^{n_y \times n_u}$. Systems of this type are represented in μ-Tools by a single MATLAB data structure, referred to as a SYSTEM matrix.

Throughout this manual, we use the notation

$$\left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right]$$

This is not to be confused with a CONSTANT MATLAB matrix containing the state-space data. Rather, this notation usually refers to the causal, linear, dynamical system described by the differential equations

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

However, sometimes the notation stands for the transfer function

$D + C(sI - A)^{-1} B,$

and sometimes it pertains to the SYSTEM matrix containing the state-space data. In any event, the exact meaning is purposely left vague, and easily determined from context.

The command `pck` creates SYSTEM matrices from separate state-space data. The matrices

```
a = [-.15,.5; -.5, -.15];
b = [.2 4; -.4 0];
c = [5 5];
d = [.1 -.1];
```

represent the state-space data of a two-state, two-input, single-output system. The SYSTEM matrix, `sys`, is created by

```
sys = pck(a,b,c,d);
```

Structural information about the matrix `sys` can be obtained with the command `minfo`.

```
minfo(sys)

system:        2 states    1 outputs       2 inputs
```

The command unpck extracts the a, b, c, and d matrices from a SYSTEM matrix. You can also examine the contents of the a, b, c, and d matrices without explicitly forming them as new variables. Use the command see for this purpose.

```
see(sys)

A matrix

   -0.1500    0.5000

   -0.5000   -0.1500

press any key to move to B matrix

B matrix

   -0.2000    4.0000

   -0.4000         0

press any key to move to C matrix

C matrix

         5         5

press any key to move to D matrix

D matrix

   -0.1000   -0.1000
```

The commands minfo and see work on any of the μ-Tools data structures. The command pss2sys converts CONSTANT matrix data in packed form, [A B; C D], into a μ-Tools SYSTEM matrix. The command sys2pss transforms a SYSTEM matrix in a packed CONSTANT matrix. Alternatively, you can generate a purely random SYSTEM matrix with the command sysrand by specifying its number of states, inputs and outputs.

The command spoles finds the eigenvalues of the *A* matrix of a SYSTEM matrix. The transmission zeros are calculated using szeros. In this example, sys has no transmission zeros. A formatted display of the system poles is produced with the μ-Tools command rifd.

```
spoles(sys)
ans =

 –0.1500 + 0.5000i

 –0.1500 – 0.5000i

rifd(spoles(sys))
    real       imaginary     frequency      damping

 –1.5000e-01  –5.0000e-01   5.2202e–01   2.8735e–01

 –1.5000e-01   5.0000e–01   5.2202e–01   2.8735e–01
```

SYSTEM matrices can easily be interconnected (cascade, parallel, feedback) to give new SYSTEM matrices. See “Interconnecting Matrices” for more information.

VARYING Matrices

Matrix-valued functions of a single, independent real variable are common in systems theory. VARYING matrices The frequency response of a multiple-input, multiple-output (MIMO) system is a good example of such a function. The independent variable is frequency, and at each frequency the transfer function between the inputs and the outputs is a complex matrix. represents these types of matrix functions with a data structure called a VARYING matrix.

In general, suppose G is a matrix-valued function of a single real variable $G{:}\mathrm{R} \to \mathrm{C}^{n \times m}$. One method to store this function on the computer is to evaluate the function G at N discrete values of $x \in \mathrm{R}$, call them $x_1,x_2,\ldots,x_N$ and store all of the evaluations. This is the approach taken by μ-Tools.

Consider a simple example.

```
mat1 = [.1 -.1;.25.5];
iv1 = 0;
mat2 = 2*mat1;
iv2 = 1;
mat3 = 2*mat2;
iv3 = 2;
```

The command vpck creates the VARYING matrix data structure from *column stacked* matrix and independent variable data. This is done as follows.

```
matdata = [mat1; mat2; mat3];
ivdata = [iv1; iv2; iv3];
vmat = vpck(matdata,ivdata)
```

minfo displays structural characteristics of the matrix and displays the data.

```
minfo(vmat)

varying:3 pts   2 rows   2 cols


see(vmat)

2 rows 2 columns

iv = 0

   0.1000   –0.1000

   0.2500    0.5000

iv = 1

   0.2000   –0.2000

   0.5000    1.0000

iv = 2

   0.4000   –0.4000

   1.0000    2.0000
```

Note that variable name iv stands for independent variable in the above display. The command seeiv displays only the independent variable values of the VARYING matrix vmat.

```
seeiv(vmat)
```

Analogous to vpck the command vunpck unpacks the matrix data and independent variable data from a VARYING matrix.

Several commands allow manipulation of the matrix and independent variable data. `tackon` simply appends one VARYING matrix to another (both must have the same number of rows and columns in the matrix data). This destroys any sequential properties of the independent variable data; i.e., the data will not be in monotonically increasing order. `sortiv` reorders the matrices within a VARYING matrix so that the independent variables are increasing (or decreasing).

All of the information about the structure is contained in a single MATLAB matrix, hence functions returning single matrices can be nested. Hence, sophisticated manipulations can be formed as single line commands. For example, to merge two VARYING matrices and reorder the independent variable's values, use the command `sortiv(tackon(vmat1,vmat2))`.

CONSTANT Matrices

If a MATLAB variable is neither a SYSTEM nor a VARYING matrix it is treated by μ-Tools as a CONSTANT matrix. CONSTANT matrices CONSTANT matrices can be arguments to functions that normally expect VARYING or SYSTEM matrix arguments.

The treatment of CONSTANT matrices is consistent with that of a constant gain linear system. In operations normally performed on SYSTEM matrices, the CONSTANT matrix is analogous to a linear system with only a D matrix. In operations where a VARYING interpretation is required, the CONSTANT matrix is assumed to be constant across all values of the independent variable. This is consistent with the frequency response (or step response) of a constant gain linear system.

Acknowledgments

The data structures used in are based on the following paper.

Stein, Gunter, and Stephen Pratt, "LQG Multivariable Design Tools," *AGARD Lecture Series No 117, Multi-variable Analysis and Design Techniques*, September 1981.

Accessing Parts of Matrices

The SYSTEM matrix and VARYING matrix data structures allow a large amount of data to be stored in a single MATLAB entity. It is often necessary to display or operate on only a part of that data.

The command sel creates subsystems from VARYING, SYSTEM, or CONSTANT matrices. The subsystem rows (or outputs) and columns (or inputs) are specified. Conceptually, the options are

```
lcl subvmat = sel(vmat,rows,columns);
subconst = sel(const,rows,columns);
subsyst = sel(syst,outputs,inputs);
```

For example,

```
subvmat = sel(vmat,1:2,2);
minfo(subvmat)

varying:3 pts  2 rows  1 cols
```

selects rows 1 and 2 and column 2 from each matrix in vmat. You can use the MATLAB colon notation in the specification of the rows and columns. To select *all* rows or columns, use the character string ':' in single quotes. When sel is used on a SYSTEM matrix, only the dimensions of the B, C, and D matrices change. All the states remain, which may result in a (non)minimal system. Extra states of the system can be removed by performing a balanced realization (sysbal) or with the commands strunc and sresid.

For VARYING matrices you can access a portion of the independent variables with the command xtract. For example,

```
vmat2 = xtract(vmat,0.5,1.5);
```

selects the matrices in vmat with independent variables between 0.5 and 1.5. In this case it is a VARYING matrix with a single data point.

```
see(vmat2)

2 rows 2 columns

iv = 1

    0.2000   –0.2000
    0.5000    1.0000
```

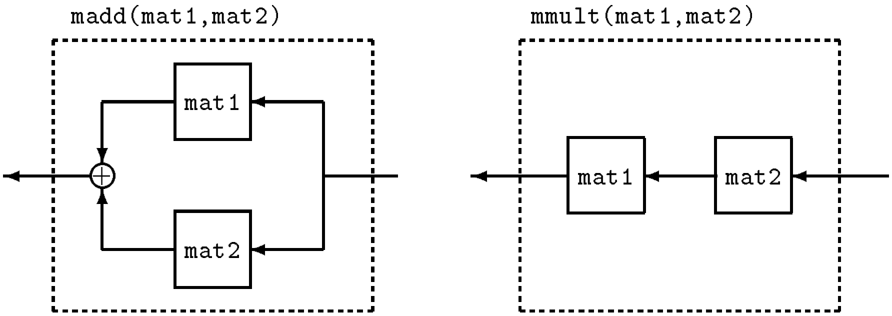
A VARYING matrix can be converted to a CONSTANT matrix via the command var2con.

The command xtracti extracts (as a VARYING matrix) the data by independent variable *index*, rather than by independent variable value. As in the case of xtract, xtracti returns a VARYING matrix.

Interconnecting Matrices

μ-Tools provides several functions for connecting matrices together. All of the functions described here work with interconnections of SYSTEM and CONSTANT matrices or VARYING and CONSTANT matrices. If the matrices represented are consistent, the combination is allowed. The interconnection of a SYSTEM and a VARYING matrix is not allowed in μ-Tools (actually it is allowed — see veval for examples of VARYING SYSTEM matrices).

The commands madd, msub, and mmult perform the appropriate arithmetic operations on the matrices. A block diagram representation is shown in the following figure.



Note that for multivariate matrices, the order of the arguments is important. In the VARYING matrix case, the arithmetic operations are performed matrix-by-matrix, for each value of the independent variable. The following example illustrates this:

A two-row and one-column VARYING matrix, vmat3, is constructed with three independent variables values.

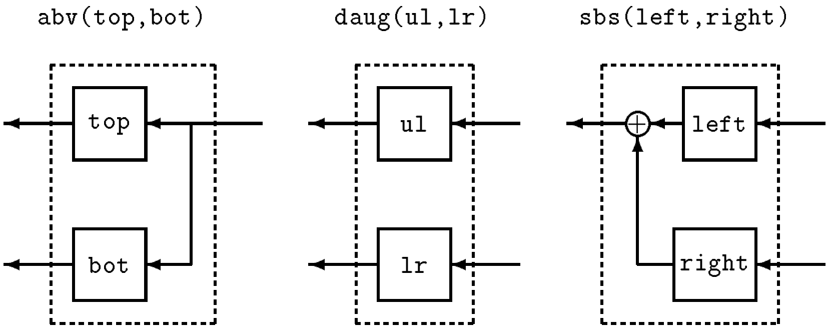
```
vmat3 = vpck([2 2 4 4 8 8]',[0 1 2]');
minfo(vmat3)

varying:   3 pts   2 rows  1 cols
```

The VARYING matrix vmat3 is multiplied by vmat to form vmat4 and the value of the resulting matrix at the independent variables between 0 and 0.5 are displayed.

```
vmat4 = mmult(vmat,vmat3);
minfo(vmat4) 60.23in

varying:   3 pts   2 rows  1 cols

see(xtract(vmat4,0,0.5))

2 rows 1 column

iv =   0
       0
       1.5000
```

Each matrix in vmat is 2 × 2. Each matrix in vmat3 is 2 × 1, and so the multiplication results in each matrix of vmat4 being 2 × 1. Commands madd, msub, and mmult allow up to nine matrices of compatible dimensions to be added, subtracted or multiplied simultaneously by including them as input arguments. When interconnecting VARYING matrices with any of the commands in this section, a check is made to verify that each matrix has the same independent variable values. If not, an error is returned.

There are additional commands for combining matrices: abv, daug, and sbs. These can be interpreted as placing matrices above one another, diagonal augmentation of matrices, and placing matrices side-by-side, as shown in the following figure.



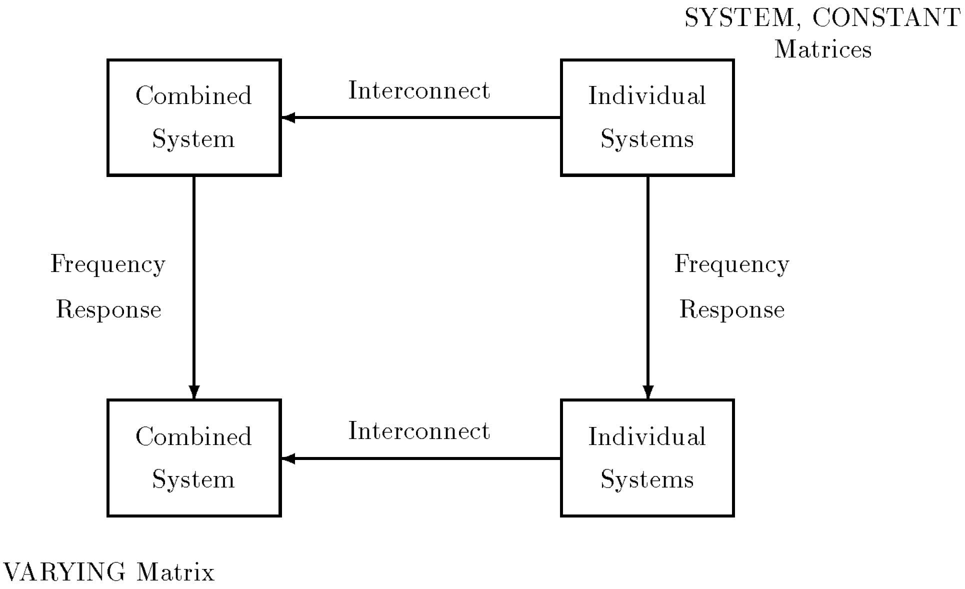
VARYING matrices are combined as described above. For example,

```
see(xtract(daug(vmat,vmat3),0.5,1.5))

4 rows     3 columns

iv = 1

       0.2000      –0.2000            0
       0.5000       1.0000            0
            0            0       4.0000
            0            0       4.0000
```

`abv`, `daug`, and `sbs` allow up to nine matrices of compatible dimensions to be combined simultaneously.

For large interconnections of matrices, it is tedious building them up piece-by-piece using these commands (even using an M-file to do it). The program `sysic` performs general interconnections, using algebraic descriptions of the relationships between the inputs, outputs, and internal matrices. See “Interconnection of SYSTEM Matrices: sysic” for more detail.

μ-Tools handles CONSTANT, VARYING, and SYSTEM matrices, such that the following diagram is commutative.



Consider beginning in the upper-right corner (individual systems of SYSTEM and CONSTANT matrices) and proceeding to the lower left corner (a single VARYING matrix). The result will be independent of the path taken. This is because in linear systems, the frequency response of an interconnection is the algebraic interconnection of the individual frequency responses. However, due to numerical roundoff, the calculations actually are not commutative, and there may be small differences between the two results. For example, it is sometimes numerically better to interconnect the frequency response of two systems rather than interconnect the two systems and then take their frequency response. This can be true when there are a large number of states in the interconnection structure.

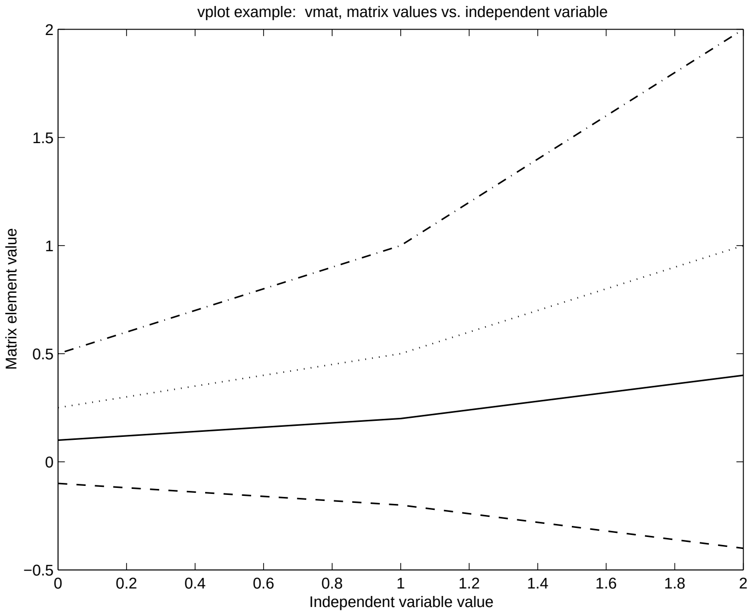
Interconnections involving feedback are performed with `sysic`, described in “Interconnection of SYSTEM Matrices: sysic” . The basic feedback loop interconnection program used by `sysic` is called `starp`, and is described in Chapter 4, “m-Tools Commands for LFTs” on page 4-10.

The commands (`sbs`, `mmult`, `starp`, etc.) to form the interconnection step are identical whether you are dealing with VARYING or SYSTEM representations.

Plotting VARYING Matrices

The function vplot plots VARYING matrices. The arguments for vplot are similar to MATLAB's plot command, with the exception that it is not necessary to specify the values for the *x*-axis. The *x*-axis data corresponds to the independent variable's values, which are already stored within each VARYING matrix. For example,

```
vplot(vmat)
tmp1 = 'vplot example: vmat, matrix values';
tmp2 = ' vs. independent variable';
title([ tmp1 tmp2 ]) xlabel('independent variable value')
ylabel('matrix element value')
```



Note that every element of the matrix is plotted against the appropriate independent variable. In the above example vmat is a 2×2 VARYING matrix, giving four elements to be plotted. There are only three values (0, 1, and 2) of the independent variable, and by default MATLAB draws a line between points on the plot.

In the MATLAB plot command, different axis types are accessed by different functions, loglog, semilogx, and others. In vplot the axis type is set by an optional string argument. The default, used in the above example, is a linear/linear scale. The generic vplot function call looks like

```
vplot('axistype',vmat1,'linetype1',vmat2,'linetype2',...)
```

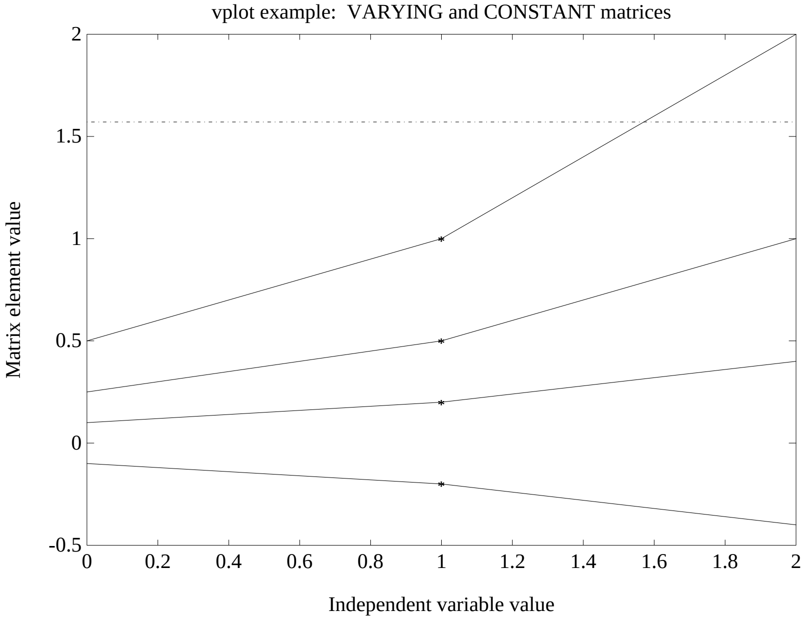
The axistype argument, a character string, allows the specification of logarithmic or linear axes as well as: magnitude, log magnitude, and phase. There are also some control-specific options: bode, nyq, and nic, which specify Bode, Nyquist, and Nichols plots, respectively. The complete range of choices is not demonstrated here. Refer to vplot in the Chapter 8, “Reference”, for more details. Subsequent sections introduce additional control-oriented examples and demonstrate other options in vplot.

The linetype arguments are optional and are identical to those in the MATLAB plot command.

An important feature of vplot is its ability to plot multiple VARYING matrices on the same plot without having to have the same independent variables. A vmat argument may be a CONSTANT. In this case the value of the CONSTANT is plotted over all independent variables. This is consistent with the interpretation given to CONSTANT matrices in the interconnection of systems. Hence a CONSTANT matrix would appear as a horizontal line on the plot. This is to be contrasted with a VARYING matrix containing only one data point, which would appear as a single point on the plot.

Consider the following example where vmat2 is a VARYING matrix with only one independent variable value (ie., one data point). The constant pi/2 is also plotted.

```
vplot(vmat,'r-',vmat2,'g*',pi/2,'b-.')
title('vplot example: VARYING and CONSTANT matrices')
xlabel('Independent variable value')
ylabel('Matrix element value')
```


vplot example: VARYING and CONSTANT matrices
Matrix element value
2
1.5
1
0.5
0
-0.5
0
0.2
0.4
0.6
0.8
1
1.2
1.4
1.6
1.8
2
Independent variable value

VARYING Matrix Functions

Many of the MATLAB matrix functions have analogous μ-Tools functions that operate on VARYING matrices. These operations are performed on a matrix-by-matrix basis for every submatrix associated with an independent variable value within the VARYING matrix. If a CONSTANT matrix is the argument of these functions, the operation is identical to the corresponding MATLAB function. These functions are

```
vabs      vceil     vdet      vdiag     veig
veval     vexpm     vfloor    vinv      vimag
vnorm     vpinv     vpoly     vrcond    vreal
vroots    vschur    vsvd
```

The functions `veval` and `vebe` perform a named operation on VARYING matrices. `vebe` performs MATLAB or user-defined functions on the *elements* of a VARYING matrix (for example: `sin`, `tan`. . .). `veval` operates on the entire VARYING matrix and can perform any function including those with multiple input and output arguments. `vebe` and `veval` allow the evaluation of any MATLAB matrix function on VARYING matrices. The following example shows a VARYING matrix with only one independent variable value for brevity.

```
minfo(vmat2)
varying:   1 pts   2 rows  2 cols

see(veval('sin',vmat2))
2 rows     2 columns
iv = 1

   0.1987   –0.1987

   0.4794    0.8415

vmat2dat = var2con(vmat2);
sin(vmat2dat)

ans =

    0.1987  –0.1987

    0.4794   0.8415
```

```
vebe('sqrt',sin(vmat2dat))
ans =

     0.4458          0 + 0.4458i

     0.6924    0.9173
```

All arithmetic operations can be performed on VARYING matrices, sometimes with a built-in μ-Tools function, and sometimes resorting to `veval`. The following table summarizes some standard operation.

MATLAB Matrix Function	μ-Tools VARYING Function
`A + B +...+ H`	`madd(A,B,...,H)`
`A – B –...– H`	`msub(A,B,...,H)`
`A * B *...* H`	`mmult(A,B,...,H)`
`A / B`	`vrdiv(A,B)`
`A \ B`	`vldiv(A,B)`
`A .* B`	`veval('.*',A,B)`
`A ./ B`	`veval('./',A,B)`
`A ^ b`	`veval('^',A,b)`
`A .^b`	`veval('.^',A,b)`
`A'`	`cjt(A)`
`A.'`	`transp(A)`
`conj(A)`	`cj(A)`
`sin(A)`	`vebe('sin',A)`

In each case, `veval` could have been used. However, `veval` can be quite slow, since it is essentially a `for` loop of `eval` commands. For that reason, some specific commands (`madd`, `mmult`, `vldiv`, etc.) are provided. The complete set of VARYING operations should allow you to write algorithms more easily using the data structures in μ-Tools.

More Sophisticated SYSTEM Functions

Frequency Domain Functions

The command `frsp` calculates frequency responses of SYSTEM matrices. You can specify the frequencies at which the response is to be evaluated via the MATLAB `logspace` and `linspace` commands. These become the independent variable values in the VARYING frequency response output.

Given an input vector of N real frequencies, `omega` = $[\omega_1, . . ., \omega_N]$ and a SYSTEM matrix `sys`, the μ-Tools command `frsp`,

```
sys_g = frsp(sys,omega)
```

calculates

$$C(jw_i I - A)^{-1}B + D,\ i = 1, . . ., N$$

for each independent variable w_i and stores it in `sys_g` whose independent variables are the N frequency points. You can specify a discrete-time evaluation by specifying an optional sampling time, T. In this case each matrix in the VARYING output is

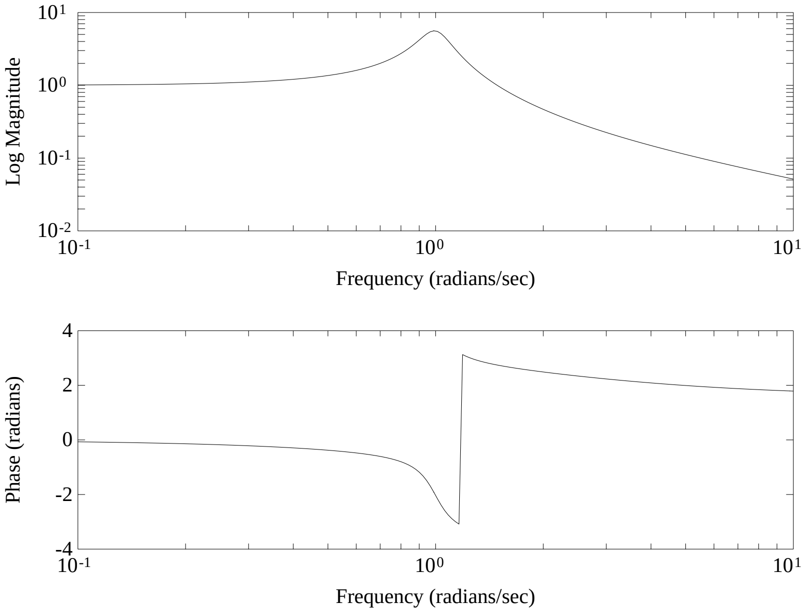
$$C(e^{j\omega_i T}\ I - A)\ B + D.$$

Consider a simple second order example. The function `nd2sys` creates the SYSTEM representation from numerator and denominator polynomials. In the following example the system `sys1` has the transfer function

$$\frac{-0.5s + 1}{s^2 + 0.2s + 1}.$$

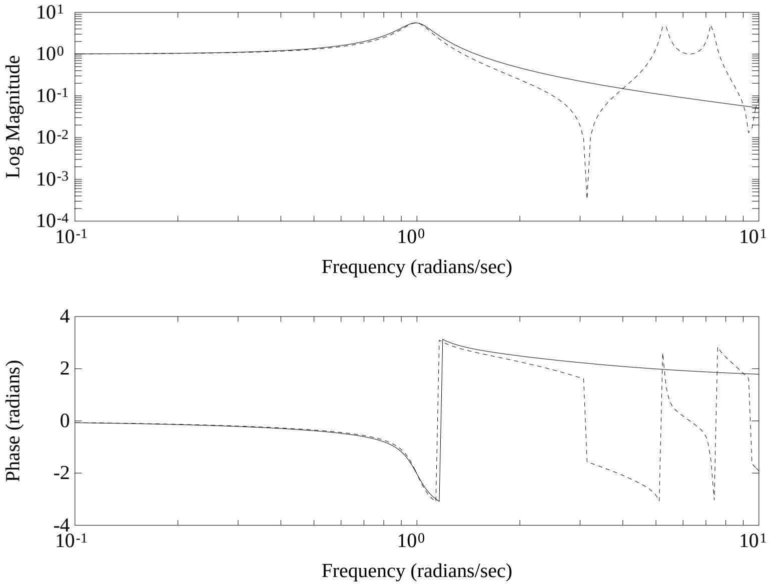
The MATLAB command logspace can create a logarithmically spaced frequency vector.

```
sys1 = nd2sys([-0.5,1],[1,0.2,1]);
minfo(sys1)

system:    2 states    1 outputs   1 inputs

omega = logspace(-1,1,200);
sys1g = frsp(sys1,omega);
minfo(sys1g)

varying:   200 pts     1 rows      1 cols

vplot('bode',sys1g)
```



You can transform sys1 to the digital domain via a prewarped Tustin transformation. The command tustin performs this function. In this example a sample time of one second is used. The prewarping frequency is chosen as one radian/second. For control purposes it is often better to choose the crossover point as the prewarp frequency.

```
dsys1 = tustin(sys1,1,1);
dsys1z = frsp(dsys1,omega,1);
vplot('bode',sys1g,dsys1z)
```

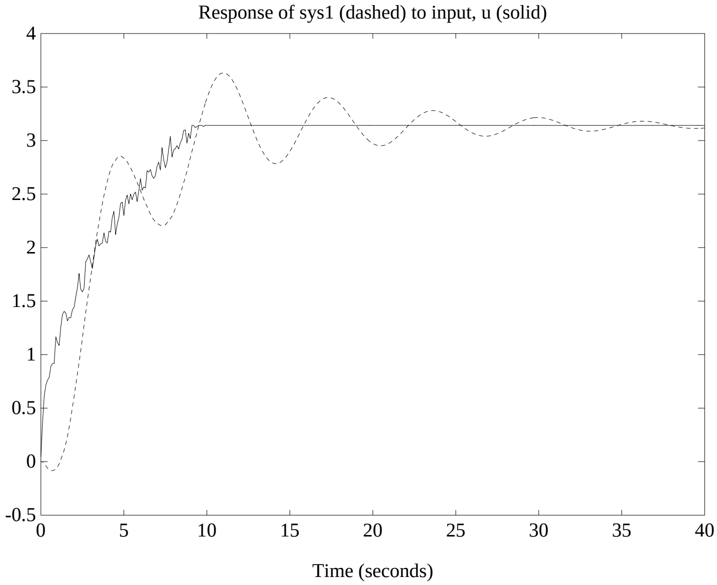


Time Domain Functions

Time responses of continuous systems are calculated with the function trsp. The required input arguments are the SYSTEM matrix and an input matrix. A discrete SYSTEM matrix is handled with the function dtrsp.

User-specified time functions can be created with the function siggen. siggen can create signals based on both random and deterministic functions. In this example, siggen generates an input, u, to the system, sys1. Note how a saturation, in this case π, is implemented.

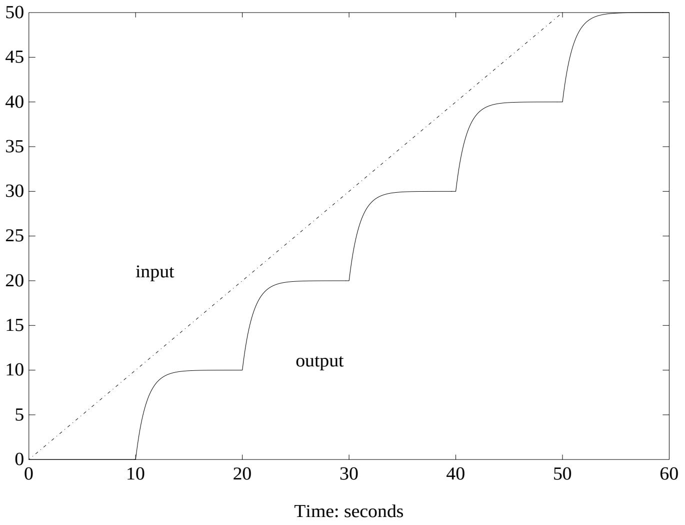
```
u = siggen('min(pi,sqrt(t)+0.25*rand(size(t)))',[0:.1:40]);
y = trsp(sys1,u);

integration step size: 0.1

vplot(u,y)
title('Response of sys1 (dashed) to input, u (solid)')
xlabel('Time (seconds)')
```



`trsp` calculates a default step-size based on the minimum spacing in the input vector and the highest frequency eigenvalue of the A matrix. For high order systems, we recommended you use some form of model reduction (see the "Model Reduction" section in Chapter 3) to remove high frequency modes which do not have a significant effect on the output.

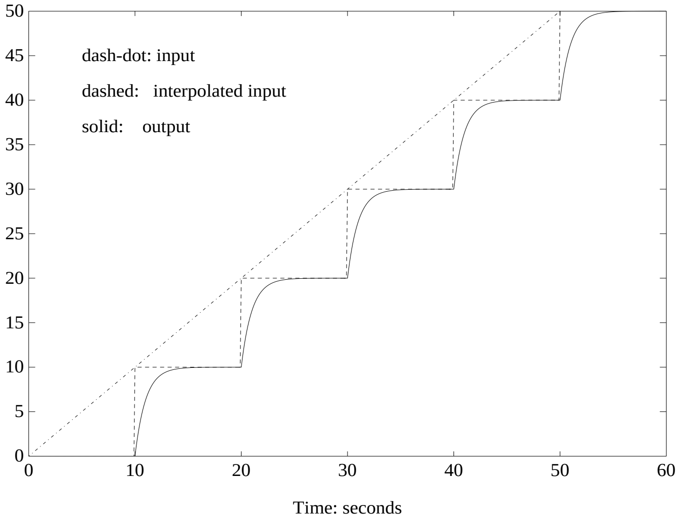
trsp assumes that the input is constant between the values specified in the input vector. The following example illustrates the consequences of this assumption.

```
sys1a = pck(-1,1,1);
minfo(sys1a)

system:    1 states    1 outputs   1 inputs
u1a = vpck([0:10:50]',[0:10:50]');
y1a = trsp(sys1a,u1a,60);

integration step size: 0.1
interpolating input vector (zero under hold)
minfo(y1a)

varying:   601 pts     1 rows      1 cols
vplot(u1a,'-.',y1a,'-')
xlabel('Time: seconds')
text(10,20,'input'), text(25,10,'output')
```



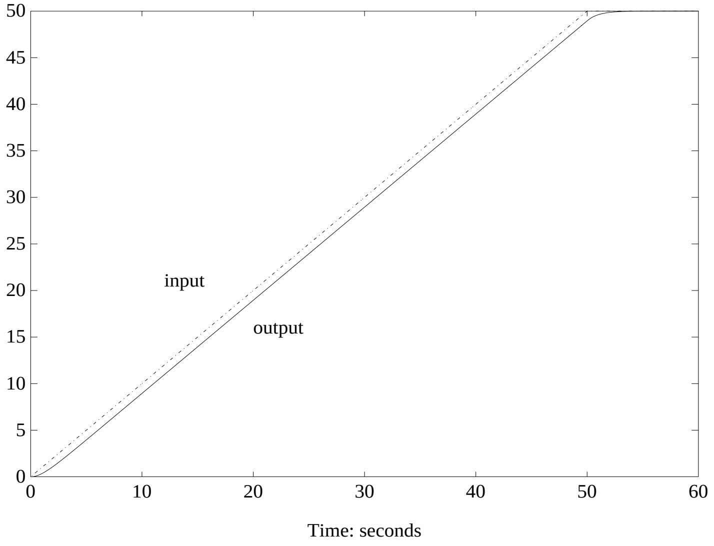
At first glance the output does not seem to be consistent with the plotted input. Remember that trsp assumes that the input is held constant between specified values. The vplot and plot commands display a linear interpolation between points. This can be seen by displaying the input signal interpolated to at least as small a step-size as the default integration step (here 0.1 seconds). The μ-Tools function vinterp performs zero-order hold or linear interpolation of the independent variable.

```
vplot(u1a,'-.',vinterp(u1a,0.1),'--',y1a,'-')
xlabel('Time: seconds')
text(5,44,'dash-dot: input')
text(5,40,'dashed: interpolated input')
text(5,36,'solid: output')
```



The staircase nature of the input is now evident. To have a ramp input, you can use the function vinterp to provide linear interpolation as shown by the following example.

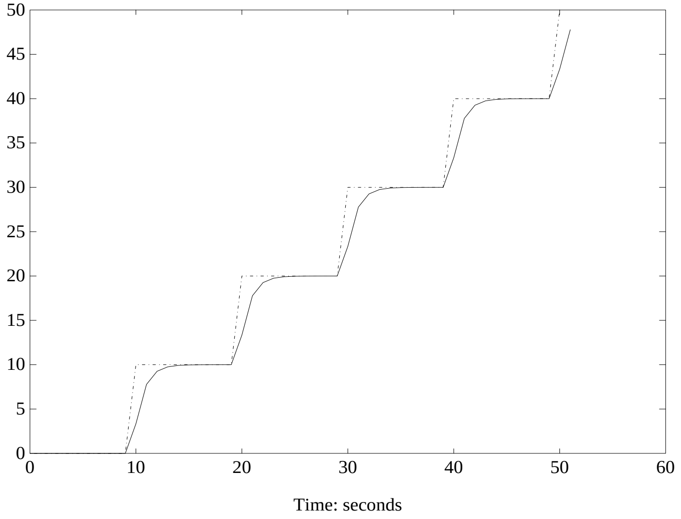
```
uramp = vinterp(u1a,0.1,60,1);
minfo(uramp)

varying:    601 pts    1 rows      1 cols
yramp = trsp(sys1a,uramp);

integration step size: 0.1
vplot(uramp,'-.',yramp,'-')
xlabel('Time: seconds')
text(20,15,'output')
text(12,20,'input')
```



Note that because the input is regularly spaced, with spacing less than or equal to the default integration time, `trsp` does not interpolate the input. No final time was specified in the `trsp` argument list. However 60 seconds was specified to `vinterp` as the final time, and this became the last time in the input vector `uramp`.

To illustrate the use of `dtrsp`, a bilinear transformation generates a digital system. The sample time is chosen as 1 second. The output is plotted against a 1 second interpolation of the input.

```
T = 1;
dsys1a = tustin(sys1a,T);
ydig = dtrsp(dsys1a,u1a,T);
vplot(ydig,'-',vinterp(u1a,1),'-.')
xlabel('Time: seconds')
```



trsp can easily generate large VARYING matrices. If the independent variables are more closely spaced than necessary for a good graphical display, you can use the μ-Tools function vdcmate to select every nth point from the final output.

Note that both vinterp and vdcmate have analogous Signal Processing Toolbox functions, interp and decimate, but do not function in the same manner.

Signal Processing and Identification

The μ-Tools VARYING data structure provides a convenient means of storing large amounts of experimental data in a single MATLAB matrix. To use this in identification experiments, routines have been provided to perform Fast Fourier Transforms (FFTs) and inverse FFTs for VARYING matrices. The functions are vfft and vifft, respectively. These routines call the appropriate MATLAB fft and ifft routines to perform the FFT calculations and have the same function arguments as those for fft and ifft.

Note that the FFTs performed are one dimensional, irrespective of the row and column dimensions of the VARYING matrix. The independent variable is used as the index in the FFT. Consequently xfreq = vfft(xtime) works if xtime is a VARYING matrix with row and column dimensions greater than one. The result, xfreq, is a VARYING matrix of the same size and has the same number of independent variables. The independent variables of xtime are assumed to be time in seconds. xfreq is returned with frequency (in radians/second) as the independent variable.

In the following example a random signal is passed through a single-input, two-output system and FFTs of the outputs are performed with a single vfft function. The sample frequency is 10 Hz, with a foldover frequency of 10π radians/second. This may be hard to distinguish on the log-log plot. Output time history plots are shown in Figure 2-1 and their FFTs are shown in Figure 2-2.

```
time = [0:0.1:102.3]';
u1 = siggen('rand(size(t)) - 0.5',time);
sys2 = nd2sys(1,[5,1]);
sys3 = abv(sys1,sys2);
y1 = trsp(sys3,u1);
   integration step size: 0.1
minfo(y1)
   varying:    1024pts     2 rows      1 cols
vplot(y1)
title('Response of sys3 (solid) to input u1 (dashed)')
xlabel('Time: seconds')

y1f = vfft(y1);
minfo(y1f)

   varying:    1024pts     2 rows      1 cols
vplot('liv,lm',y1f)
z='Fast Fourier Transform magnitude of sys3 response: y1';
title([ z ])
xlabel('Frequency: radians/second')
ylabel('Magnitude')

Warning: Data includes a number that is negative or zero. The LOG
of hits results in NaN or Infinity and is not shown on plot.
```

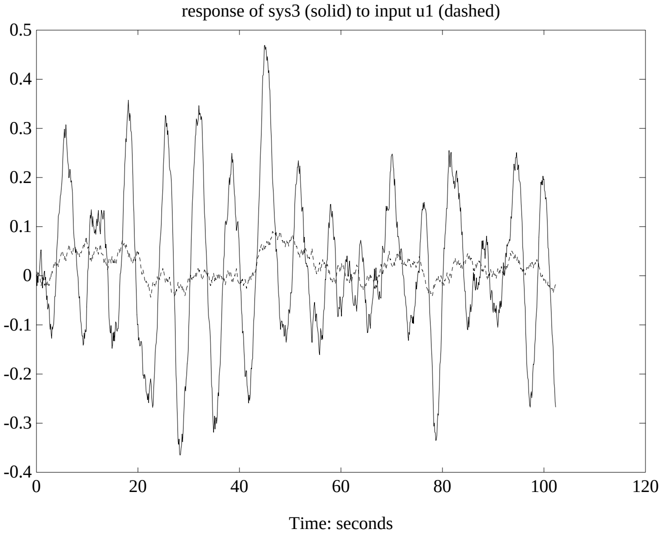


Figure 2-1 System Response Time Histories

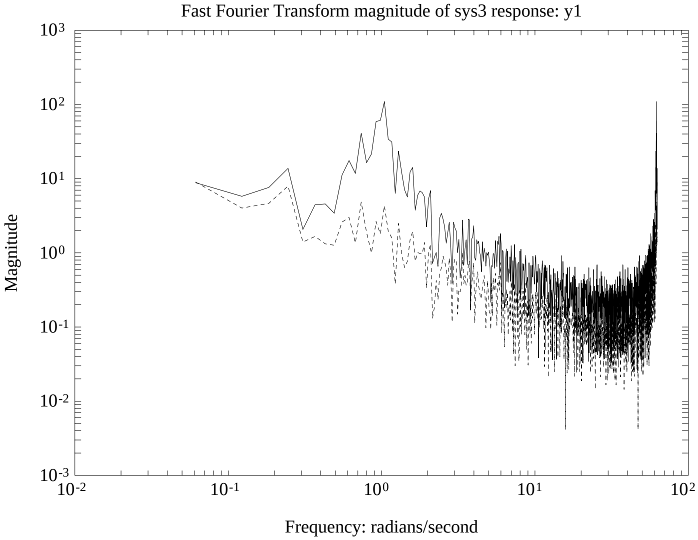


Figure 2-2: FFTs of the Output

The μ-Tools function vplot displays this warning message since there is a data value at zero frequency that cannot be plotted on the log frequency scale.

The Signal Processing Toolbox provides a means of performing spectral analysis with the function spectrum. The μ-Tools function vspect operates in a similar manner on VARYING matrices. Given a signal x and a signal y, vspect can calculate the power spectral density of x (P_{xx}), the power spectral density of y (P_{yy}), the cross spectral density (P_{xy}), the transfer function from x to y (T_{xy}), and the coherence (C_{xy}). The VARYING matrix result will have the following five columns, [P_{xx}, P_{yy}, P_{xy}, T_{xy}, C_{xy}]. The command vspect(x,m) will calculate the power spectral density of each element of the VARYING matrix, x, using averaged FFTs of length m. The algorithm is exactly that used for spectrum. See the Signal Processing Toolbox for further information.

The calculation of transfer functions using vspect restricts the input x to be a one-by-one VARYING matrix and the outputs y to be an nr × 1 VARYING matrix. This corresponds to being able to do single-input, multiple-output (SIMO) identification experiments for a system with nr outputs. Each row of the result then corresponds to an output or its associated SISO transfer function. The calling sequence for vspect is vspect(x,y,m).

In the following example, vspect estimates the single-input, two-output transfer function from the data generated in the previous example. A small amount of random noise is added to the output, y1, to make the problem more realistic. 512 point FFTs are applied to the data with an overlap of 256 points. An optional Hamming window, the fifth input argument, is used on the data in conjunction with the FFTs. The fourth column of the output is the estimated transfer function, which is displayed in Figure 2-3 using the sel function.

```
noise1 = siggen('0.05*(rand(2,1)-[0.5;0.5])',time);
y1meas = madd(y1,noise1);
P1 = vspect(u1,y1,512,256,'hamming');
3 hamming windows in averaging calculation
minfo(P1)

varying: 256 pts   2 rows                 5 cols
vplot('bode',sel(P1,[1:2],4),'-',frsp(sys3,omega),'-.')
tmp1 = 'sys3 (dash-dot) and estimated';
tmp2 = ' transfer function (solid)';
title([ tmp1 tmp2 ])
```

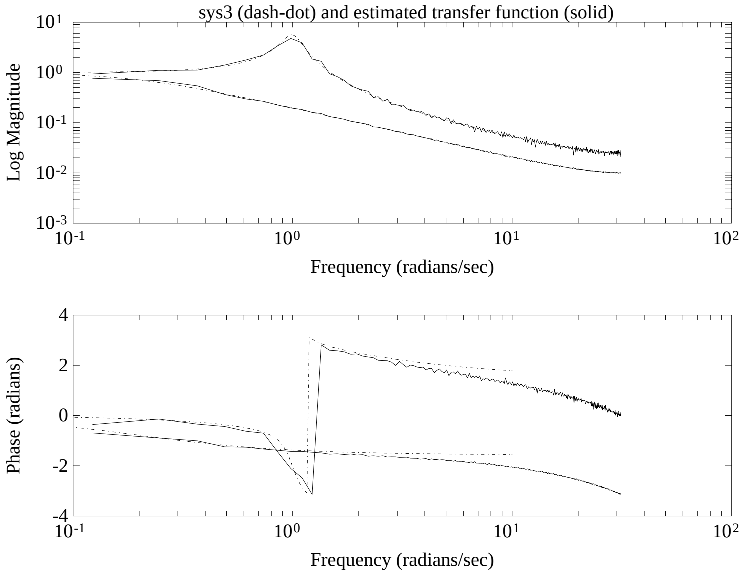


Figure 2-3: Estimated Transfer Functions of the System

The command `fitsys` can be used to construct a state-space realization of the estimated transfer function data. The first input argument to `fitsys` is the frequency response data as a VARYING matrix. This argument, `frdata`, can have dimension of a single-input/multi-output or multiple-input/single-output VARYING matrix. The second input argument, ord, is the state-order of the desired fit. The third input argument, `weight` is a weighting matrix with the same independent variable values as `frdata`. The fourth input argument, `code`, can be set to restrict the curve fitting algorithm to stable transfer functions. The default value for `weight` is 1, and the default value of `code` is 0, placing no restriction on the location of the rational fit's poles. The output of `fitsys` is the SYSTEM `sys`. The state-order of `sys` is ord.

In this example fit the single-input, two-output estimated transfer function data with a third order model. Recall that the fourth column of P is the estimated transfer function data. We will restrict the fitting algorithm to consider only frequency points between 0.1 rad/sec and 10 rad/sec. A plot of the estimated transfer function data and the third order model, `sysord3`, is shown in Figure 2-4. Notice that the poles of `sysord3` are very close to the poles of `sys3`, which was used to create the estimated transfer function data.

```
estdata = sel(P1,[1:2],4);
sysord3 = fitsys(xtract(estdata,0.1,10),3);
vplot('bode',estdata,'-',frsp(sysord3,estdata),'-.')
tmp1 = 'sysord3 (dashed) and estimated';
tmp2 = ' transfer function (solid)';
title([ tmp1 tmp2 ])
rifd(spoles(sysord3))
     real        imaginary     frequency      damping

  –3.1841e–01   –0.0000e+00    3.1841e–01    1.0000e+00

  –1.1555e-01   –9.8732e–01    9.9406e–01    1.1624e–01

  –1.1555e-01    9.8732e–01    9.9406e–01    1.1624e–01

rifd(spoles(sys3))
     real        imaginary     frequency      damping

  –2.0000e–01    0.0000e+00    2.0000e–01    1.0000e+00

   1.0000e–01   –9.9499e–01    1.0000e+00    1.0000e–01

  –1.0000e–01    9.9499e–01    1.0000e+00    1.0000e–01
```

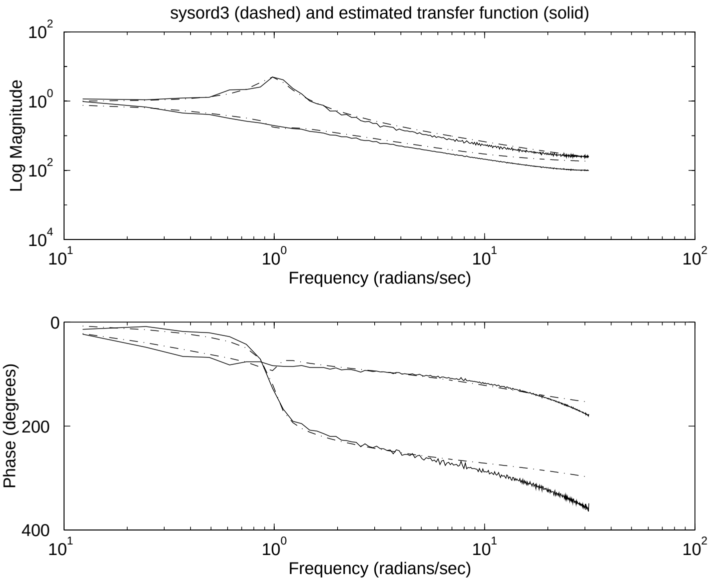


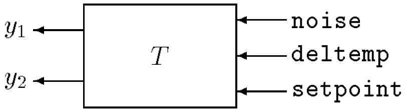
Figure 2-4: Estimated Transfer Functions and Third Order Model

Interconnection of SYSTEM Matrices: sysic

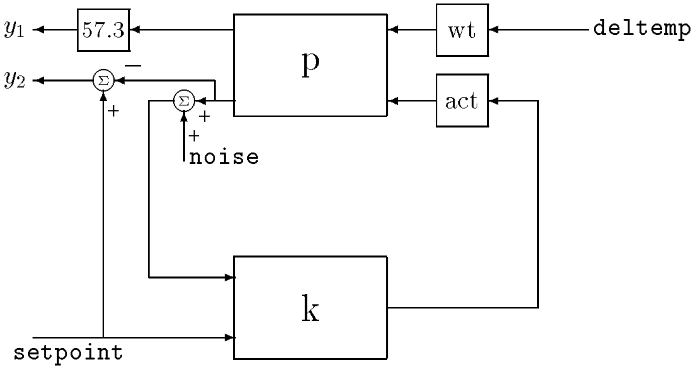
μ-Tools has a simple linear interconnection program called sysic. sysic forms linear interconnections of CONSTANT and SYSTEM matrices (or CONSTANT and VARYING matrices) simply by calculating the loop equations of the interconnection.

Using sysic involves setting up several variables in the MATLAB workspace, and then running the script-file sysic. The defined variables delineate the details of the interconnection.

In order to explain the meaning of the sysic commands, consider a three-input, two-output SYSTEM matrix *T*,



which has internal structure



Variable Descriptions

Following are descriptions of the variables required by `sysic`.

systemnames

This variable is a character string, which contains the names of the matrices (ie., the subsystems) used in the interconnection. The names must be separated by spaces and/or tabs, and there should be no additional punctuation. The order in which the names appear is not important. Each named system must exist in the MATLAB workspace at the time the program `sysic` is run.

For the interconnection shown, with four components, `k`, `p`, `act`, and `wt`, the following is an appropriate definition for the variable `systemnames`.

```
systemnames = ' k p act wt ';
```

The name of SYSTEM variables used within the `sysic` program is limited to 10 characters. This limitation is due to the MATLAB 19 character limitation on the workspace variable names.

inputvar

This variable is a character string, with names of the various *external inputs* that are present in the final interconnection. The input names are separated by semicolons, and the entire list of input names is enclosed in square brackets [ ]. Inputs can be multivariable signals, for example a windgust input with three directions (x, y, and z) is specified by using `windgust{3}`. This indicates three-variable input to the interconnection called `windgust`. Alternatively, this could be specified as three separate, scalar inputs, say, `wingustx`, `windgusty`, and `windgustz`. The order that the input names appear in the variable `inputvar` is the order that the inputs are placed in the interconnection.

This simple interconnection has three external scalar inputs: sensor noise, temperature disturbance, and a reference input.

```
inputvar = '[ noise; deltemp; setpoint]';
```

outputvar

This variable is a character string, describing the *external outputs* of the interconnection, which *must be linear combinations of the subsystem outputs and the external inputs*. Semicolons separate the channels of the output variables. Between semicolons, signals can be added and subtracted, and multiplied by scalars. For multivariable subsystems, arguments within parentheses specify which subsystem outputs to use and in what order. For instance, plant(2:5,8,1,9:11) specifies outputs 2, 3, 4, 5, 8, 1, 9, 10, 11 from the system plant. If no arguments are specified with a system, then it is assumed that all outputs are being used, and in the order they appear in that system.

In this example, the two outputs of the interconnection consist of the first output of the plant (scaled by 57.3 to change units from radians to degrees) along with a tracking error, which is the difference between the setpoint input and the second plant output.

```
outputvar = '[ 57.3*p(1); setpoint - p(2) ]';
```

input_to_sys

This variable denotes the inputs to a specific system. Each subsystem named in the variable systemnames must have a variable set to define the inputs to the subsystem. If the system name is controller, then call the variable that must be set using input_to_controller. Specify it in the same manner that the variable outputvar is set, with *inputs consisting of linear combinations of subsystem outputs and external inputs*. Separate channels are separated by semicolons, and the order of the inputs in the variable should match the order of the inputs in the system itself.

Corresponding to the systemnames variable set above, there are four input_to_ statements required, which are

```
input_to_k = '[ noise + p(2); setpoint ]';
input_to_act = '[ k ]';
input_to_wt = '[ deltemp ]';
input_to_p = '[ wt; act ]';
```

This means that the input to the controller consists of the sensor noise plus the second output of the plant, and the reference input. The input to the actuator is the output of the controller. The input to the weighting function is the temperature disturbance, and the input to the plant consists of the output of the weighting function, followed by the output of the actuator.

sysoutname

This character string variable is optional. If it exists in the MATLAB workspace when sysic is run, the interconnection that is created by sysic is placed in a MATLAB variable whose name is given by the string in sysoutname. If this variable does not exist in the workspace, then the interconnection is automatically placed in the variable ic_ms.

The command line

```
sysoutname = 'T';
```

will cause sysic to store the final interconnection in a SYSTEM matrix called T.

cleanupsysic

This variable is used to clean up the workspace. After running sysic, all of the above variables that describe the interconnection are left in the workspace. These will be automatically cleared if the optional variable cleanupsysic is set to the character string yes. The default value of the variable is no, which does not result in any of the user-defined sysic descriptions being cleared. The MATLAB matrices listed in the variable systemnames are never automatically cleared.

Running sysic

If the variables systemnames, inputvar, and outputvar are set, and for each name name_i appearing in systemnames, the variable input_to_name_i is set, then the interconnection is created by running the M-file sysic. Depending on the existence/nonexistence of the variable sysoutname, the resulting interconnection is stored in a user-specified MATLAB variable or the default MATLAB variable ic_ms.

Within sysic, error-checking of the consistency and availability of subsystem matrices and their inputs aid in debugging faulty sysic interconnection descriptions.

The input/output dimensions of the final interconnection are defined by inputvar and outputvar variables.

Returning to the initial example, the following sysic commands were used to generate the three-input, two-output SYSTEM matrix clp. (Note that the dimensions of the variables k, p, act, and wt must be consistent with the problem description.)

```
systemnames = ' k p act wt ';
inputvar = '[ noise; deltemp; setpoint]';
outputvar = '[ 57.3*p(1); setpoint - p(2) ]';
input_to_k = '[ noise + p(2); setpoint ]';
input_to_act = '[ k ]';
input_to_wt = '[ deltemp ]';
input_to_p = '[ wt; act ]';
sysoutname = 'clp'; cleanupsysic = 'yes';
sysic;
```

The syntax of sysic is limited, and for the most part restricted to what is shown here. Some additional features are illustrated in the more complicated demonstration problems.

HIMAT Design Example

The HIMAT example provides another example of how to construct interconnection systems from block diagram descriptions. The HIMAT plant model is described in more detail in the section, “HIMAT Robust Performance Design Example” in Chapter 7. The interconnection diagram shown in Figure 2-5 corresponds to the HIMAT design example.

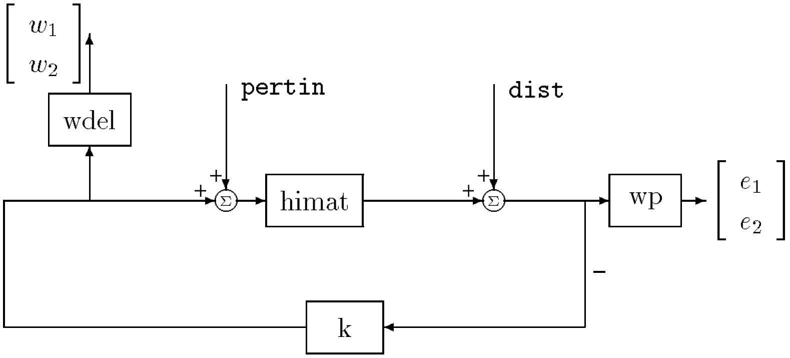
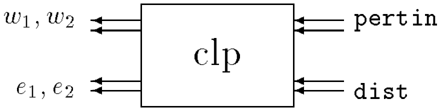


Figure 2-5: HIMAT Interconnection

Given that there are four SYSTEM matrices, named himat, wdel, wp, and k, in the MATLAB workspace, each with two inputs and two outputs, the following 10 lines form the sysic commands to make the interconnection structure shown below, which is placed in the variable clp. These can be executed at the command line (as shown) or placed in a script file. The command mkhimat needs to be run initially to create himat, wdel, and wp.

```
mkhimat
himatic
k = zeros(2,2);
systemnames = ' himat wdel wp k ';
inputvar = '[ pertin(2) ; dist(2) ]';
outputvar = '[ wdel ; wp ]';
input_to_himat = '[ k + pertin ]';
input_to_wp = '[ dist + himat ]';
input_to_wdel = '[ k ]';
input_to_k = '[ -dist - himat ]';
sysoutname = 'clp';
cleanupsysic = 'yes';
sysic;
```

The final interconnection structure is located in clp with two sets of inputs, pertin and dist, and two sets of outputs *w* and *e*, corresponding to the perturbation and error outputs.



3

H_∞ Control and Model Reduction

This chapter covers an introduction to control H_∞analysis and design, sampled-data control, and model reduction.

Optimal Feedback Control

Performance as Generalized Disturbance Rejection

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

Consider a tracking problem, with disturbance rejection, measurement noise, and control input signal limitations, as shown in Figure 3-1. *K* is some controller to be designed and *G* is the system we want to control.

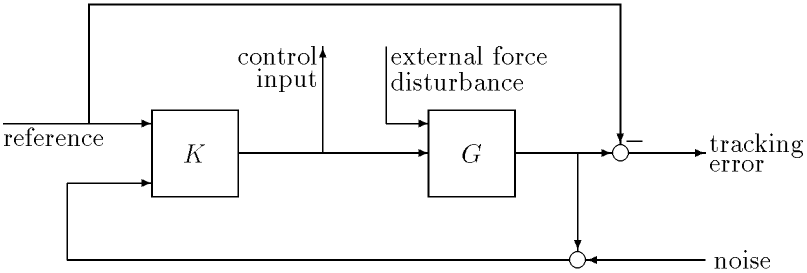


Figure 3-1 Typical Closed-Loop Performance Objectives

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

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

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

We can assess performance by measuring the gain from *outside influences to regulated variables*. In other words, good performance is associated with T being small. Since the closed-loop system is a multi-input, multi-output (MIMO) dynamical system, there are two different aspects to the gain of T:

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

To quantify the term *gain* mathematically, we need to define some additional things.

Norms of Signals and Systems

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

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

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

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

the 2-norm is defined as

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

In μ-Tools the dynamical systems we deal with are exclusively linear, with state-space model

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

or, in the transfer function form

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

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

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

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

$$\|M\|_F := \sqrt{\operatorname{trace}(M^*M)}$$

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

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

then:

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

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

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

Using Weighted Norms to Characterize Performance

In any performance criterion, we must also account for:

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

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

$$\|W_L T W_R\|$$

where the weighting function matrices W_L and W_R are frequency dependent, to account for bandwidth constraints and spectral content of exogenous signals. Within the structured singular value setting considered in Chapter 4, the most natural (mathematical) manner to characterize acceptable performance is in terms of the MIMO $\|\cdot\|_\infty$ (H_∞) norm. For this reason, we discuss some interpretations of the H_∞ norm.

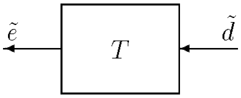


Figure 3-2: Unweighted MIMO System

Suppose T is a MIMO stable linear system, with transfer function matrix $T(s)$. For a given driving signal $\tilde{d}(t)$, define $\tilde{e}$ as the output, as shown in Figure 3-2.

Note that it is more traditional to write the diagram in Figure 3-2 with the arrows going from left to right as in Figure 3-3.

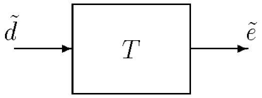


Figure 3-3: Unweighted MIMO System: Vectors from Left to Right

The diagrams in Figure 3-2 and Figure 3-3 represent the exact same system. We prefer to write these block diagrams with the arrows going right to left to be consistent with matrix and operator composition.

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

$$\beta := \|T\|_\infty := \max_{w \in \mathbf{R}} \bar{\sigma}[T(j\omega)] \tag{3-1}$$

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

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

Moreover, there are specific disturbances d that result in the ratio $\frac{\|\tilde{e}\|_2}{\|\tilde{d}\|_2}$

arbitrarily close to β. Because of this, $\|T\|_\infty$ is referred to as the L_2 (or RMS) gain of the system.

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

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

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

$$\tilde{e}_{ss}(t) = \begin{bmatrix} b_1 \sin(\bar{\omega} t + {}_1) \\ \vdots \\ b_{n_e} \sin(\bar{\omega} t + {}_{n_e}) \end{bmatrix}$$

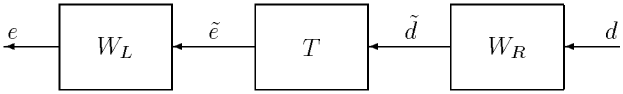
The vector $b \in \mathbf{R}^{n_e}$ will satisfy $\|b\|_2 \leq \beta$. Moreover, β, as defined in equation Figure 3-1, is the smallest number such that this fact is true for every $\|a\|_2 \leq 1$, $\bar{\omega}$, and ϕ.

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

Consider the diagram of Figure 3-4, along with Figure 3-2.

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

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



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

Figure 3-4: Weighted MIMO System

Bounds on the quantity $\|W_L T W_R\|_\infty$ will imply bounds about the sinusoidal steady-state behavior of the signals $\tilde{d}$ and $\tilde{e}(= T\tilde{d})$ in Figure 3-2. Specifically, for sinusoidal signal $\tilde{d}$, the steady-state relationship between $\tilde{e}(= T\tilde{d})$, $\tilde{d}$ and $\|W_L T W_R\|_\infty$ is as follows: The steady-state solution $\tilde{e}_{ss}$, denoted as

$$\tilde{e}_{ss}(t) = \begin{bmatrix} \tilde{e}_1 \sin(\bar{\omega}t + \varphi_1) \\ \vdots \\ \tilde{e}_{n_e} \sin(\bar{\omega}t + \varphi_{n_d}) \end{bmatrix} \tag{3-2}$$

satisfies $\sum_{i=1}^{n_e} \left| W_{L_i}(j\bar{w})\tilde{e}_i \right|^2 \le 1$ for all sinusoidal input signals $\tilde{d}$ of the form

$$\tilde{d}(t) = \begin{bmatrix} \tilde{d}_1 \sin(\bar{\omega}t + \phi_i) \\ \vdots \\ \tilde{d}_{n_d} \sin(\bar{\omega}t + \phi_{n_d}) \end{bmatrix} \tag{3-3}$$

satisfying

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

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

This approximately (*very* approximately — the next statement is not actually correct) implies that $\|W_L T W_R\|_\infty \le 1$ if and only if for every fixed frequency $\bar{\omega}$, and all sinusoidal disturbances $\tilde{d}$ of the form (3-3) satisfying

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

the steady-state error components will satisfy

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

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

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

Interconnection with Typical MIMO Performance Objectives

Throughout this manual, we formulate closed-loop performance objectives as weighted closed-loop transfer functions which are to be made small through feedback. A generic example, which includes many relevant terms, is shown in block diagram form in Figure 3-5. In the diagram, G denotes the plant model and K is the feedback controller.

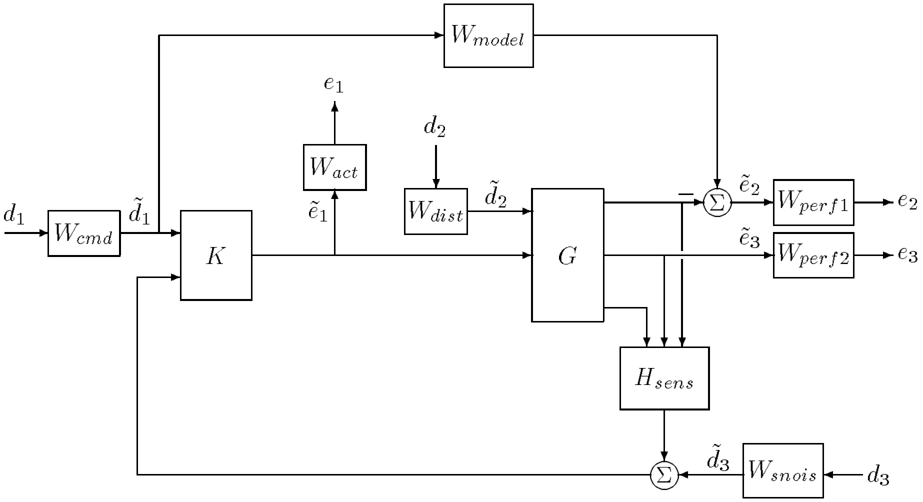


Figure 3-5: Generalized and Weighted Performance Block Diagram

The blocks in Figure 3-5 might be scalar (SISO) and/or multivariable (MIMO), depending on the specific example. The mathematical objective of H_∞ control is to make the closed-loop MIMO transfer function T_{ed} satisfy $\|T_{ed}\|_\infty < 1$. The weighting functions are used to scale the input/output transfer functions such that when $\|T_{ed}\|_\infty < 1$, the relationship between $\tilde{d}$ and $\tilde{e}$ is suitable.

This shows the interpretation of the signals, weighting functions and models.

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

W_{cmd}

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

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

W_{model}

represents a desired ideal model for the closed-looped system, used for problems with tracking requirements. For good command tracking response, we might desire our closed-loop system to respond like a well-damped second-order system. The ideal model would then be

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

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

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

W_{dist}

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

W_{perf1}

W_{perf1} weights the difference between the response of the plant and the response of the ideal model, W_{model}. Often we desire accurate matching of the ideal model at low frequency and require less accurate matching at higher frequency, in which case W_{perf1} is flat at low frequency, rolls off at first or second order, and flattens out at a small, nonzero value at high frequency. The inverse of the weight should be related to the allowable size of tracking errors, in the face of the reference commands and disturbances described by W_{ref} and W_{dist}.

W_{perf2}

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

W_{act}

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

W_{snois}

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

H_{sens}

H_{sens} represents a model of the sensor dynamics or an external anti-aliasing filter. The transfer functions used to describe H_{sens} are based on physical characteristics of the individual components. These models might also be lumped into the plant model *G*.

This generic block diagram has tremendous flexibility and many control performance objectives can be formulated using this block diagram description. In Chapter 4, we see how to incorporate uncertainty into the model of *G* (and possibly H_{sens} as well), and how to analyze the implications on performance due to uncertainty. Chapter 7 presents a number of examples, which explain in detail how individual performance weighting functions are selected.

Commands to Calculate the H_2 and H_∞ Norm

There are five μ-Tools functions to calculate the H_2 and H_∞ norm.

```
dhfnorm    h2norm    hinfnorm    pkvnorm     vnorm
```

H_2 norm

The H_2 norm of a stable, strictly proper continuous-time SYSTEM matrix can be calculated using the command `h2norm`. Its calling sequence is

```
out = h2norm(sys)
```

The output variable, `out`, is a scalar whose value is the two-norm of the SYSTEM `sys`. Given a state-space description of a system as

$$\text{sys} = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right]$$

The H_2 norm of the SYSTEM follows from the solution to the Lyapunov equation

$$AX + XA' + BB' = 0.$$

with $\|\text{sys}\|_2 = \sqrt{[tr(CXC)']}$.

H_∞ norm

The H_∞ norm of a stable, continuous-time SYSTEM, `sys`, can be calculated using the command `hinfnorm`. Its calling sequence is

```
out = hinfnorm(sys,tol)
```

The output from `hinfnorm` is a 1×3 vector, `out`, which is made up (in order) of a lower bound for $\|\text{sys}\|_\infty$, an upper bound for $\|\text{sys}\|_\infty$, and a frequency, ω_o, at which the lower bound is achieved.

$$\text{out}(1) = \bar{\sigma}(\text{sys}(j\ \text{Þ}\ \text{out}(3))) \le \|\text{sys}\|_\infty \le \text{out}(2)$$

The$\|\cdot\|_\infty$norm calculation is an iterative process and requires a test to stop. See the `hinfnorm` manual page in Chapter 8, “Reference” for more details. If the first input argument to `hinfnorm`, `sys`, is a VARYING matrix, then `hinfnorm` calls pkvnorm to find the maximum singular value of the VARYING matrix across frequency.

The H_∞ norm of a frequency VARYING matrix, sysg, can be calculated using pkvnorm or vnorm. The calling sequences are

```
[peak,indv,index] = pkvnorm(matin)
out = vnorm(matin)
```

pkvnorm sweeps through the independent variable and calculates the largest singular value of matin. The three output arguments all pertain to the peak norm across frequency and its location: peak value, peak, the independent variable's value, indv, and the independent variable's index, index.

vnorm is a VARYING matrix version of MATLAB's norm command. The operation is identical, except that it also works on CONSTANT and VARYING matrices, producing a CONSTANT or VARYING output. vnorm returns the matrix out with its norm at each independent variable value. The default is the largest singular value of matin at each independent variable value.

Discrete-Time H_∞ Norm

The H_∞ norm of a discrete-time SYSTEM can be calculated using the command dhfnorm. Its calling sequence is

```
out = dhfnorm(sys)
```

The first input argument, sys, can be either a discrete-time SYSTEM, a CONSTANT or VARYING matrix. The output of dhfnorm, out, is a 1×3 vector giving a lower bound, upper bound of the discrete-time H_∞ norm and the frequency where the lower bound occurs.

Commands to Design H_∞ Output Feedback Controllers

Given a linear system P with four types of external variables:

- Exogenous disturbances (d)
- Regulated variables, i.*e*., errors (e)
- Manipulated variables, i.*e*., controls (u)
- Sensed variables, i.*e*., measurements (y)

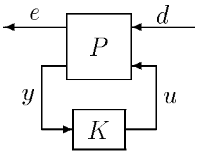
These are related through the linear state-space equations

$$P = \left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{array}\right] = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] = C(sI-A)^{-1}B + D$$

The H_∞ output feedback control design problem is: Does there exist a linear controller, K, with internal structure

$$K = \left[\begin{array}{c|c} A_K & B_K \\ \hline C_K & D_K \end{array}\right]$$

such that the closed-loop system $e = F_L(P, K)d$,



is stable and the ∞-norm of $F_L(P, K)$ is less than γ? Note that the above block diagram represents a linear fractional transformation (LFT). LFTs are described in more detail in the “Representing Uncertainty” section in Chapter 4. The LFT equation $F_L(P,K)$ is given by $P_{11} + P_{12}K(I - P_{22}K)^{-1}P_{21}$.

The standard state-space technique to calculate H_∞ output feedback controllers is to select a value of γ and determine if there exists a controller K such that $\|F_L(P,K)\|_\infty < \gamma$. This value of γ is updated based on a modified bisection algorithm, called γ *iteration*. This iteration procedure continues until the magnitude of the difference between the smallest γ value that has passed and the largest γ value that has failed is small.

Two functions are included in μ-Tools to synthesize continuous-time H_∞ controllers. `hinfsyn` and `hinfsyne` calculate the H_∞ output feedback (sub)optimal controller. The `hinfsyn` command constructs the standard centralized H_∞ controller whereas the `hinfsyne` command constructs the H_∞ (sub)optimal controller that minimizes the entropy integral at a specific frequency. The default in `hinfsyne` is to minimize the entropy integral at ∞. Both `hinfsyn` and `hinfsyne` commands implement the full general output feedback equations based on the interconnection structure P. The complete set of equations and results for the H_∞ control problem can be found in the "H• Output Feedback" section of this chapter and in references [DoyGKF] and [GloD].

The following assumptions are made about the open-loop system P in the `hinfsyn` and `hinfsyne` commands:

(A1) (A,B_2) is stabilizable and (C_2,A) is detectable.

(A2) D_{12} is full column rank and D_{21} is full row rank.

(A3) $\begin{bmatrix} A - j\omega I & B_2 \\ C_1 & D_{12} \end{bmatrix}$ has full column rank for all ω.

(A4) $\begin{bmatrix} A - j\omega I & B_1 \\ C_2 & D_{21} \end{bmatrix}$ has full row rank for all ω.

`hinfsyn` and `hinfsyne` return the H_∞ controller, the closed-loop system, and the γ level achieved.

The `hinfsyn` and `hinfsyne` programs provide a γ iteration using a modified bisection method. You select a high and low value of γ, `gamma_max` and `gamma_min`. The bisection method iterates on the value of γ in an effort to approach the optimal H_∞ control design. If the value `gamma_max` equals `gamma_min`, only one γ value is tested. The bisection algorithm stops when the difference between the smallest value of γ that has passed and the largest value of γ that has failed is less than `tol`.

The calling sequence for the `hinfsyn` command is

```
[k,clp,gfin] =
   hinfsyn(p,nmeas,ncon,gamma_min,gamma_max,tol)
```

`p` is the SYSTEM interconnection structure, `nmeas` is the number of measurements, `ncon` is the number of control inputs, `gamma_min` and `gamma_max` are the minimum and maximum γ values, and `tol` is the difference between final γ values. The output argument `k` is an H_∞ (sub)optimal controller, `clp` is the closed-loop system with H_∞ controller implemented (`clp = starp(p,k)`), and `gfin` is the final γ value associated with `k` and `clp`. See the manual pages for `hinfsyn` in Chapter 8, “” for more information. The calling sequence for the command is identical to.

H_∞ Design Example

The objective is to design an H_∞ (sub)optimal control law for SYSTEM interconnection structure given by the block diagram in Figure 3-6. The HIMAT plant model and weightings are described in more detail in the “HIMAT Robust Performance Design Example” section in Chapter 7.

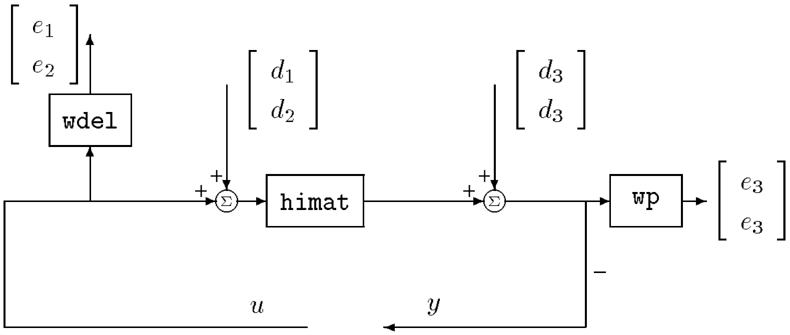

$\begin{bmatrix} e_1 \\ e_2 \end{bmatrix}$
wdel
$\begin{bmatrix} d_1 \\ d_2 \end{bmatrix}$
$\begin{bmatrix} d_3 \\ d_3 \end{bmatrix}$
+
+
Σ
himat
+
+
Σ
wp
$\begin{bmatrix} e_3 \\ e_3 \end{bmatrix}$
−
u
y

Figure 3-6: H_∞ Design Example Interconnection Structure

The SYSTEM interconnection structure is located in `himat_ic`. It consists of two sensor measurements, two error signals, two actuator inputs, two disturbance inputs, and eight states. The range of γ is selected to be between 1.0 and 10.0 with a tolerance, `tol`, on the relative closeness of the final γ solution of 0.1. For each iteration, the program prints the current γ value being tested, and the results of five tests for the existence of a controller achieving the closed-loop norm objective. At the end of each iteration a (`p`) or (`f`) is displayed denoting that the γ value either passed or failed. Upon finishing, `hinfsyn` and `hinfsyne` print out the lowest γ value achieved. A # sign is used in the printout to denote which of the five conditions for the existence of an H_∞ (sub)optimal controller failed.

```
   nmeas = 2;
   ncont = 2;
   gmn = 1;
   gmx = 10;
   tol = 0.1;
   mkhimat
   himatic
   minfo(himat_ic)
   system: 8 states   6 outputs   6 inputs
   p = himat_ic;
   [k,clp] = hinfsyn(p,ncont,nmeas,gmn,gmx,tol);
   Test bounds: 1.0000 < gamma<=10.0000

gamma    hamx_eig  xinf_eig  hamy–eig  yinf–eig  nrho–xy   p/f

   10.000   2.3e–02   1.2e–07   2.3e–02  –1.5e–11    0.0223     p

   5.5000   2.3e–02   1.3e–07   2.3e–02   0+0e+00    0.0747     p

    3.250   2.3e–02   1.3e–07   2.3e–02   0+0e+00    0.2222     p

    2.125   2.3e–02   1.3e–07   2.3e–02   0+0e+00    0.5642     p

    1.562   2.3e–02   1.4e–07   2.3e–02   0+0e+00   1.1977#     f

    1.711   2.3e–02   1.4e–07   2.3e–02   0+0e+00    0.9474     p

    1.677   2.3e–02   1.4e–07   2.3e–02  –3.0e–14    0.9973     p

    1.654   2.3e–02   1.4e–07   2.3e–02   0+0e+00   1.0328#     f

   Gamma value achieved:1.6770
```

We can verify that the closed-loop system achieved an H_∞ norm of 1.6770 by calculating the H_∞ norm using the `hinfnorm` command.

```
   hinfnorm(clp)
   norm between 1.667 and 1.679
   achieved near 3.814
```

H_∞ Optimal Control Theory

This part of the H_∞ chapter presents simple state-space formulae for all controllers solving a standard H_∞ problem: For a given number $\gamma \geq 0$, find all stabilizing controllers such that the H_∞ norm of the closed-loop transfer function is less than γ. Under these conditions, a parametrization of all controllers solving the problem is given as a linear fractional transformation (LFT) on a contractive, stable free parameter. The state dimension of the coefficient matrix for the LFT equals that of the plant, and has a separation structure reminiscent of classical LQG (i.*e*., H_2) theory. The results are essentially from reference [GloD2] with proofs removed. This directly generalizes the results in reference [DoyGKF] and [GloD]. It is assumed that the reader has at least read [DoyGKF] before attempting to read the material to follow.

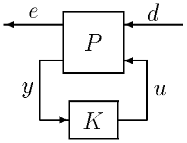
Two popular performance measures in optimal control theory are H_2 and H_∞ norms. Recall that they defined in the frequency domain for a stable transfer matrix $P(s)$ as

$$\|P\|_2 := \left(\frac{1}{2\pi}\int_{-\infty}^{\infty} \mathrm{trace}[P(j\omega)^*P(j\omega)]\,d\omega\right)^{1/2}$$

$$\|P\|_\infty := \sup_{\omega} \bar{\sigma}([)P(j\omega)] \quad (\bar{\sigma}(:) = \text{maximum singular value})$$

The former arises when the exogenous signals either are fixed or have a fixed power spectrum; the latter arises from (weighted) balls of exogenous signals. H_2-optimal control theory was heavily studied in the 1960's as the Linear Quadratic Gaussian (LQG) optimal control problem; H_∞-optimal control theory is continuing to be developed. We assume the reader either is familiar with the engineering motivation for these problems, or is interested in the results of this chapter for some other reason.

The basic block diagram used in this chapter is



where P is the generalized plant and K is the controller. Only finite dimensional linear time-invariant (LTI) systems and controllers will be considered. The generalized plant P contains what is usually called the plant in a control problem plus all weighting functions. The signal d contains all external inputs, including disturbances, sensor noise, and commands, the output e is an error signal, y is the measured variables, and u is the control input. The diagram is also referred to as a linear fractional transformation (LFT) on K and P is called the coefficient matrix for the LFT. The resulting closed loop transfer function from d to e is denoted by $T_{ed} = F_L(P,K)$.

The main H_∞ output feedback results are presented in the “H• Output Feedback” section. The proofs of these results exploit the *separation* structure of the controller. If perfect measurements of the states (x) and the disturbances (d) are available (this is defined as the Full Information problem), then the central controller is simply a gain matrix F_∞, obtained through finding a certain stable invariant subspace of a Hamiltonian matrix. Also, the optimal output estimator is an observer whose gain is obtained in a similar way from a dual Hamiltonian matrix. These special cases are described in the “H• Full Information and Full Control Problems” section. In the general output feedback case the controller can be interpreted as an optimal estimator for $F_\infty x$. Furthermore, the two Hamiltonians involved in this solution can be associated with full information and output estimation problems.

As mentioned, this material is taken primarily from [GloD2], which is a direct generalization of [DoyGKF], and contains a substantial repetition of material. Roughly speaking, [GloD2] proves those results in [GloD] which were stated without proof, using [DoyGKF] machinery, which considered a less general problem. An alternative approach in relaxing some of the assumptions in [DoyGKF] is to use loop-shifting techniques as in [ZhouK], [GloD], and more completely in [SafLC]. We also consider some aspects of generalizations to the ≤ case, primarily to indicate the problems encountered in the optimal case. A detailed derivation of the necessity of the generalized conditions for the Full Information problem is given. In keeping with the style of [GloD] and [DoyGKF], we don’t present a complete treatment of the ≤ case. Complete derivations of the optimal output feedback case can be found in [GlovM] using different techniques.

Historical Perspective

This section is not intended as a review of the literature in H_∞ theory, but rather an attempt to outline some of the work that led up to and most closely touches on [DoyGKF], [GloD], and [GloD2]. Control, history For a more extensive bibliography and review of earlier literature, the interested reader might see [Fran1] and [FranD].

Zames' [Zame] original formulation of H_∞ optimal control theory was in an input-output setting. Most solution techniques available at that time involved analytic functions (Nevanlinna-Pick interpolation) or operator-theoretic methods [Sara], [AdAK], and [BallH]. An earlier state-space solution was presented in [Doy1], in which the steps were as follows: parametrize all internally stabilizing controllers via Youla [YouJB]; obtain realizations of the closed-loop transfer matrix; convert the resulting model-matching problem into the equivalent 2 × 2-block general distance or best approximation problem involving mixed Hankel-Toeplitz operators; reduce to the Nehari problem (Hankel only); and solve the Nehari problem by the procedure of [Glo1]. Both [Fran1] and [FranD] give expositions of this approach, which will be referred to as the "1984" approach.

In a mathematical sense, the 1984 procedure solved the general rational H_∞ optimal control problem and much of the subsequent work in H_∞ control theory focused on the 2 × 2-block problems, either in the model-matching or general distance forms. Unfortunately, the associated complexity of computation was substantial, involving several Riccati equations of increasing dimension, and formulae for the resulting controllers tended to be very complicated and have high state dimension. Encouragement came from [LimH] who showed, for problems transformable to 2 × 1-block problems, that a subsequent minimal realization of the controller has state dimension no greater than that of the generalized plant G. This suggested the likely existence of similarly low dimension optimal controllers in the general 2 × 2 case.

Additional progress on the 2 × 2-block problems came from [BallC], who gave a state-space solution involving three Riccati equations. [JonJ] showed a connection between the 2 × 1-block problem. [FoisT] developed an interesting class of operators called skew Toeplitz to study the 2 × 2-block problem. Other approaches have been derived by [Hung] using an interpolation theory approach, [Kwak] using a polynomial approach, and [Kim] using a method based on conjugation.

In addition to providing controller formulae that are simple and expressed in terms of plant data, the methods in the present approach are a fundamental departure from the earlier work described above. In particular, the Youla parametrization and the resulting 2×2-block model-matching problem of the 1984 solution are avoided entirely; replaced by a pair of 2×1-block problems and a separation argument. The entire development uses simple and familiar tools, in the style of [Will1] relying on state feedback and observer-based control methods and more straightforward and elegant use of operator theory. Another strong influence on this work is Redheffer's work [Red2] on linear fractional transformations.

Independent encouragement for a simpler approach to the H_∞ problem came from papers by [KhaPZ] and [ZhoK]. They showed that for the state-feedback H_∞ problem one can choose a constant gain as a (sub)optimal controller. In addition, a formula for the state-feedback gain matrix was given in terms of an algebraic Riccati equation. These results are similar to those in the "H• Full Information and Full Control Problems" section, though the proof techniques are entirely different. Also, these papers established connections between H_∞ -optimal control, quadratic stabilization, and linear-quadratic differential games.

As expected, the results and techniques in [DoyGKF] and [GloD2] have encouraged greater interest in applications of H_∞ methods, in alternative developments of the theory using other techniques, and in extensions to more general problems. The state-space theory of H_∞ can be carried much further, by generalizing time-invariant to time-varying, infinite horizon to finite horizon, and finite dimensional to infinite dimensional. A flourish of activity has begun on these problems and the already numerous results indicate, not surprisingly, that many of the results generalize *mutatis mutandis*, to these cases.

Notation

The notation is fairly standard. The Hardy spaces H_2 and $H_2^\perp$ consist of square-integrable functions on the imaginary axis with analytic continuation into, respectively, the right and left half-plane. The Hardy space H_∞ consists of bounded functions with analytic continuation into the right half-plane. The Lebesgue spaces $L_2 = L_2(-\infty,\infty)$, $L_{2+} = L_2[0,\infty)$, and $L_{2-} = L_2(-\infty,0]$ consist respectively of square-integrable functions on $(-\infty,\infty)$, $[0,\infty)$, and $(-\infty,0]$, and L_∞ consists of bounded functions on $(-\infty,\infty)$. As interpreted in this chapter, L_∞

will consist of functions of frequency, L_{2+} and L_{2-} functions of time, and L_2 will be used for both.

We will make liberal use of the Hilbert space isomorphism, via the Laplace transform and the Paley-Wiener theorem, of $L_2 = L_{2+} \oplus L_{2-}$ in the time-domain with $L_2 = H_2 \oplus H_2^{\perp}$ in the frequency-domain and of L_{2+} with H_2 and L_{2-} with $H_2^{\perp}$. In fact, we will normally not make any distinction between a time-domain signal and its transform. Thus we may write $d \in L_{2+}$ and then treat d as if $d \in H_2$. This style streamlines the development, as well as the notation, but when any possibility of confusion could arise, we will make it clear whether we are working in the time- or frequency-domain.

All matrices and vectors will be assumed to be complex. A transfer matrix in terms of state-space data is denoted

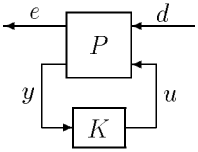
$$\left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] := C(sI - A)^{-1}B + D$$

For a matrix $M \in \mathbf{C}^{p \times r}$, M' denotes its conjugate transpose, $\bar{\sigma}(M) = \rho(M'M)^{1/2}$ denotes its maximum singular value, $\rho(M)$denotes its spectral radius (if $p = r$), and $M^{\dagger}$ denotes the Moore-Penrose pseudo-inverse of M. *Im* denotes image, *ker* denotes kernel, and $P^{\sim}(s) := P(-\bar{s})'$. For operators, Γ^* denotes the adjoint of Γ. The prefix B denotes the open unit ball and the prefix R_c denotes complex-rational.

The orthogonal projections P_+ and P_- map L_2 to, respectively, H_2 and$H_2^{\perp}$ (or L_{2+} and L_{2-}). For $P \in L_{\infty}$, the Laurent or multiplication operator $M_P : L_2 \rightarrow L_2$ for frequency-domain $d \in L_2$ is defined by $M_P d = Pd$. The norms on L_∞ and L_2 in the frequency-domain were defined in the “Performance as Generalized Disturbance Rejection” section. Note that both norms apply to matrix or vector-valued functions. The unsubscripted norm $\|\bullet\|$ will denote the standard Euclidean norm on vectors. We will omit all vector and matrix dimensions throughout, and assume that all quantities have compatible dimensions.

Problem Statement

Consider the system described by the block diagram



Both P and K are complex-rational and proper, and K is constrained to provide internal stability. We will denote the transfer functions from d to e as T_{ed} in general and for a linear fractional transformation feedback connection as above we also write $T_{ed} = F_L(P,K)$. This section discusses the assumptions on P that will be used. In our application we have state models of P and K. Then *internal stability* will mean that the states of P and K go to zero from all initial values when $d = 0$.

Since we will restrict our attention exclusively to proper, complex-rational controllers that are stabilizable and detectable, these properties will be assumed throughout. Thus the term controller will be taken to mean a controller that satisfies these properties. Controllers that have the additional property of being internally stabilizing will be said to be *admissible*. Although we are taking everything to be complex, in the special case where the original data is real (e.g., P is real-rational) then all of the results (such as K) will also be real.

The problem to be considered is to find all admissible $K(s)$ such that $\|T_{ed}\|_\infty < \gamma(\le \gamma)$. The realization of the transfer matrix P is taken to be of the form

$$P(s) = \left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{array}\right] = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right]$$

compatible with the dimensions $e(t) \in \mathbf{C}^{p_1}$, $y(t) \in \mathbf{C}^{p_2}$, $d(t) \in \mathbf{C}^{m_1}$, $u(t) \in \mathbf{C}^{m_2}$, and the state $x(t) \in \mathbf{C}^n$. The following assumptions are made:

(A1) (A,B_2) is stabilizable and (C_2,A) is detectable.

(A2) D_{12} is full column rank with $\begin{bmatrix} D_{12} & D_{\perp} \end{bmatrix}$ unitary and D_{21} is full row rank with $\begin{bmatrix} D_{21} \\ \tilde{D}_{\perp} \end{bmatrix}$ unitary.

(A3) $\begin{bmatrix} A - j\omega I & B_2 \\ C_1 & D_{12} \end{bmatrix}$ has full column rank for all ω.

(A4) $\begin{bmatrix} A - j\omega I & B_1 \\ C_2 & D_{21} \end{bmatrix}$ has full row rank for all ω.

Assumption (A1) is necessary for the existence of stabilizing controllers. The assumptions in (A2) mean that the penalty on $e = C_1x + D_{12}u$ includes a nonsingular, normalized penalty on the control u, and that the exogenous signal ω includes both plant disturbance and sensor noise, and the sensor noise weighting is normalized and nonsingular. Relaxation of (A2) leads to singular control problems.

Assumption (A3) relaxes the [DoyGKF] assumptions that (C_1,A) is detectable and $D'_{12}C_1 = 0$, and (A4) relaxes (A,B_1) stabilizable and $B'_1D_{21} = 0$. Assumptions (A3) and (A4) are made for a technical reason: together with (A1) it guarantees that the two Hamiltonian matrices in the corresponding H_2 problem belong to $dom(Ric)$. It is tempting to suggest that (A3) and (A4) can be dropped, but they are, in some sense, necessary for the methods in this chapter to be applicable. A further discussion of the assumptions and their possible relaxation will be discussed in the “Relaxing Assumptions A1–A4” section.

It will be assumed in the rest of this chapter that $D_{22} = 0$. To see how to handle the general case for $D_{22} \neq 0$, suppose K is a stabilizing controller for D_{22} set to zero, and satisfies

$$\left\| F_L\left(P - \begin{bmatrix} 0 & 0 \\ 0 & D_{22} \end{bmatrix}, K\right) \right\|_{\infty} < \gamma$$

then

$$F_L(P,K(I + D_{22}K)^{-1}) = P_{11} + P_{12}K(I + D_{22}K - P_{22}K)^{-1}P_{21}$$

$$= F_L\left(P - \begin{bmatrix} 0 & 0 \\ 0 & D_{22} \end{bmatrix}, K\right).$$

Hence a controller K for

$$P - \begin{pmatrix} 0 & 0 \\ 0 & D_{22} \end{pmatrix}$$

yields a controller $\tilde{K} = K(I + D_{22}K)^{-1}$ for P. The μ-Tools commands `hinfsyn` and `hinfsyne` handle the nonzero D_{22} case.

When $D_{22} \neq 0$ there is a possibility of the feedback system becoming ill-posed due to $\det(I + D_{22}\tilde{K}(\infty)) = 0$ (or more stringent conditions if we require well-posedness in the face of infinitesimal time delays [Will1]). Such possibilities need to be excluded.

It can be assumed, without loss of generality, that $\gamma = 1$ since this is achieved by the scalings $\gamma^{-1}D_{11}$, $\gamma^{-1/2}B_1$, $\gamma^{-1/2}C_1$, $\gamma^{1/2}B2$, $\gamma^{1/2}C_2$, and $\gamma^{-1}K$. This will be done implicitly for many of statements of this chapter.

Preliminaries

This section reviews some mathematical preliminaries, in particular the computation of the various norms of a transfer matrix P. Consider the transfer matrix

$$P(s) = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] \tag{3-4}$$

with A stable (i.*e*., all eigenvalues in the left half-plane).

The norm $\|P\|_\infty$ arises in a number of ways. Suppose that we apply an input $d \in L_2$ and consider the output $e \in L_2$. Then a standard result is that

$\|P\|_\infty$ is the induced norm of the multiplication operator M_P, as well as the Toeplitz operator $P_+M_P : H_2 \to H_2$.

$$\|P\|_\infty = \sup_{d \in BL_2} \|e\|_2 = \sup_{d \in BL_{2+}} \|P_+ e\|_2 = \sup_{d \in BH_2} \|P_+ M_P d\|_2$$

The rest of this section involves additional characterizations of the norms in terms of state-space descriptions. “The Riccati Operator” section collects some basic material on the Riccati equation and the Riccati operator, which play an essential role in the development of both theories.

The Riccati Operator

Let A, Q, R be complex $n \times n$ matrices with Q and R Hermitian. Define the $2n \times 2n$ Hamiltonian matrix

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

If we begin by assuming H has no eigenvalues on the imaginary axis, then it must have n eigenvalues in $Re\ s < 0$ and n in $Re\ s > 0$. Consider the two n-dimensional spectral subspaces $\chi_-(H)$ and $\chi_+(H)$: the former is the invariant subspace corresponding to eigenvalues in $Re\ s < 0$; the latter, to eigenvalues in $Re\ s > 0$. Finding a basis for $\chi_-(H)$, stacking the basis vectors up to form a matrix, and partitioning the matrix, we get

$$\chi_-(H) = Im \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} \tag{3-5}$$

where $X_1, X_2 \in \mathbf{C}^{n \times n}$, and

$$H \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} T_X, \ \ \text{Re}\ \lambda_i(Tx) < 0 \forall i \tag{3-6}$$

If X_1 is nonsingular, or equivalently, if the two subspaces

$$\chi_{-}(H),\ \mathrm{Im}\begin{bmatrix} 0 \\ I \end{bmatrix} \tag{3-7}$$

are complementary, we can set $X := X_2 X_1^{-1}$. Then X is uniquely determined by H, i.*e*., H a X is a function, which will be denoted *Ric*; thus, $X = Ric(H)$. We will take the domain of *Ric*, denoted *dom*(*Ric*), to consist of Hamiltonian matrices H with two properties, namely, H has no eigenvalues on the imaginary axis and the two subspaces in equation (3-7) are complementary. For ease of reference, these will be called the stability property and the complementary property, respectively. The following well-known results give some properties of X as well as verifiable conditions under which H belongs to *dom*(*Ric*). See, for example, Section 7.2 in [Fran1], Theorem 12.2 in [Wonh], and [Kuc1].

Lemma 3.1. Suppose *H Œ dom(Ric)* and *X = Ric(H)*. Then

a X is Hermitian

b satisfies the algebraic Riccati equation

$$A'X + XA + XRX - Q = 0$$

c $A + RX$ is stable

Lemma 3.2. Suppose H has no imaginary eigenvalues, R is either positive semidefinite or negative semidefinite, and (A,R) is stabilizable. Then *H Œ dom(Ric)*.

Lemma 3.3. Suppose H has the form

$$H = \begin{bmatrix} A & -BB' \\ -C'C & -A' \end{bmatrix}$$

with *(A,B)* stabilizable and rank $\begin{bmatrix} A + j\omega I & C' \end{bmatrix} = n \forall\omega$. Then $H \in dom(Ric)$, $X = Ric(H) \geq 0$, and $\ker(X) \subset \chi :=$ stable unobservable subspace.

By stable unobservable subspace we mean the intersection of the stable invariant subspace of A with the unobservable subspace of (A,C). Note that if $(C,-A)$ is detectable, then $Ric(H) \geq 0$. Also, note that $\ker(X) \subset \chi \subset \ker(C)$, so that the equation $XM = C'$ always has a solution for M, for example the least-squares solution given by $X^{\dagger}C'$.

Computing the H_∞ Norm

For the transfer matrix $P(s)$ in equation (3-4), with A stable, define the Hamiltonian matrix

$$H := \begin{bmatrix} A + BR^{-1}D'C & BR^{-1}B' \\ -C'(I - DD')^{-1}C & -(A + BR^{-1}D'C)' \end{bmatrix} \tag{3-8}$$

$$= \begin{bmatrix} A & 0 \\ -C'C & -A' \end{bmatrix} + \begin{bmatrix} B \\ -C'D \end{bmatrix} R^{-1} \begin{bmatrix} D'C & B' \end{bmatrix} \tag{3-9}$$

where $R = I - D'D$. The following lemma is essentially from [And], [Will1], and [BoyBK].

Lemma 3.4. Let $\bar{\sigma}(D) < 1$, then the following conditions are equivalent:

a $\|P\|_\infty < 1$

b H has no eigenvalues on the imaginary axis

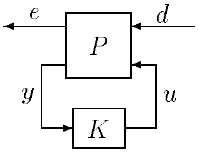
c $H \in dom(Ric)$

d $H \in dom(Ric)$ *and* $Ric(H) \geq 0$ ($Ric(H) > 0$ *if* (C,A) is observable)

Lemma 3.4 suggests the following way to compute an H_∞ norm: select a positive number γ; test if $\|P\|_\infty < \gamma$ by calculating the eigenvalues of H; increase or decrease γ accordingly; repeat. Thus H_∞ norm computation requires a search, over either γ or ω. We should not be surprised by similar characteristics of the H_∞-optimal control problem. A somewhat analogous situation occurs for matrices with the norms $\|M\|_2^2 = \text{trace}(M^*M)$ and $\|M\|_\infty = \bar{\sigma}[M]$. In principle, $\|M\|_2^2$ can be computed exactly with a finite number of operations, as can the test for whether $\bar{\sigma}(M) < \gamma$ (e.g., $\gamma^2 I - M^*M > 0$), but the value of $\bar{\sigma}(M)$ cannot. To compute $\bar{\sigma}(M)$ we must use some type of iterative algorithm.

H_∞ Full Information and Full Control Problems

In this section we discuss four problems from which the output feedback solutions will be constructed via a separation argument. These special problems are central to the whole approach taken in this chapter, and as we shall see, they are also important in their own right. All pertain to the standard block diagram,



but with different structures for P. The problems are labeled

FI	Full information
FC	Full control
DF	Disturbance feedforward (to be considered in the “Disturbance Feedforward and Output Estimation” section)
OE	Output estimation (to be considered in the “Disturbance Feedforward and Output Estimation” section)

FC and OE are natural duals of FI and DF, respectively. The DF solution can be easily obtained from the FI solution, as shown in the “Disturbance Feedforward and Output Estimation” section. The output feedback solutions will be constructed out of the FI and OE results. A dual derivation could use the FC and DF results.

The FI and FC problems are not, strictly speaking, special cases of the output feedback problem, as they do not satisfy all of the assumptions. Each of the four problems inherit certain of the assumptions A1–A4 from the “Problem Statement” section as appropriate. The terminology and assumptions will be discussed in the subsections for each problem. In each of the four cases, the results are necessary and sufficient conditions for the existence of a controller such that $\|T_{ed}\|_\infty < \gamma$ and the family of all controllers such that $\|T_{ed}\|_\infty < \gamma$. In all cases, K must be admissible.

The $H\infty$ solution involves two Hamiltonian matrices, $H\infty$ and J_∞, which are defined as follows:

$$R := D'_{1\bullet}D_{1\bullet} - \begin{bmatrix} \gamma^2 I_{m_1} & 0 \\ 0 & 0 \end{bmatrix}, \text{ where } D_{1\bullet} := \begin{bmatrix} D_{11} & D_{12} \end{bmatrix}$$

$$\tilde{R} := D_{\bullet 1}D'_{\bullet 1} - \begin{bmatrix} \gamma^2 I_{p_1} & 0 \\ 0 & 0 \end{bmatrix}, \text{ where } D_{\bullet 1} := \begin{bmatrix} D_{11} \\ D_{21} \end{bmatrix}$$

$$H_\infty := \begin{bmatrix} A & 0 \\ -C'_1C_1 & A' \end{bmatrix} - \begin{bmatrix} B \\ -C'_1D_{1\bullet} \end{bmatrix} R^{-1} \begin{bmatrix} D'_{1\bullet}C_1 & B' \end{bmatrix} \tag{3-10}$$

$$J_\infty := \begin{bmatrix} A' & 0 \\ -B_1B'_1 & -A \end{bmatrix} - \begin{bmatrix} C' \\ -B_1D'_{\bullet 1} \end{bmatrix} \tilde{R}^{-1} \begin{bmatrix} D_{\bullet 1}B'_1 & C \end{bmatrix} \tag{3-11}$$

If $H_\infty \in$ *dom(Ric)* then let X_1, X_2 be any matrices such that

$$H_\infty \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} = \begin{bmatrix} X_1 \\ X_2 \end{bmatrix} T_X, \quad X'_1X_2 = X'_2X_1, \quad \operatorname{Re}\lambda_i(T_X) \text{ ð } 0\forall i \tag{3-12}$$

Similarly if $J_\infty \in$ *dom(Ric)* then let Y_1, Y_2 be any matrices such that

$$J_\infty \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} = \begin{bmatrix} Y_1 \\ Y_2 \end{bmatrix} T_Y, \quad Y'_1Y_2 = Y'_2Y_1 \quad \operatorname{Re}\lambda_i(T_Y) \text{ ð } 0\forall i \tag{3-13}$$

Also define

$$X_\infty := X_2X_1^{-1}, \quad Y_\infty := Y_2Y_1^{-1} \tag{3-14}$$

Finally define the *state feedback* and *output injection* matrices as

$$F := \begin{bmatrix} F_1 \\ F_2 \end{bmatrix} := -R^{-1}[D'_{1\bullet}C_1 + B'X_\infty] \tag{3-15}$$

$$L := \begin{bmatrix} L_1 & L_2 \end{bmatrix} := -[B_1 D'_{\bullet 1} + Y_\infty C']\tilde{R}^{-1} \tag{3-16}$$

Problem FI: Full Information

In the full information (FI) special problem P has the following form:

$$P_{(s)} = \left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ \begin{bmatrix} I \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ I \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} \end{array}\right] \tag{3-17}$$

It is seen that the controller is provided with full information since $y = \begin{pmatrix} x \\ d \end{pmatrix}$. In some cases, a suboptimal controller may exist which uses just the state feedback x, but this will not always be possible. While the state feedback problem is more traditional, we believe that the Full Information problem is more fundamental and more natural than the state feedback problem, once you get outside the pure H_2 setting.

The assumptions relevant to the FI problem, which are inherited from the output feedback problem, are

(A1) (A,B_2) is stabilizable.

(A2) D_{12} is full column rank with $\begin{bmatrix} D_{12} & D_\perp \end{bmatrix}$ unitary.

(A3) $\begin{bmatrix} A - j\omega I & B_2 \\ C_1 & D_{12} \end{bmatrix}$ has full column rank for all ω.

The results for the Full Information case are as follows:

Theorem 3.5. Suppose P is given by (3-17) and satisfies A1–A3. Then,

- **a** $\exists K$ such that $\|T_{ed}\|_\infty < 1$ ⌠ $H_\infty \in dom(Ric)$, $Ric(H_\infty) \geq 0$
- **b** All admissible $K(s)$ such that $\|T_{ed}\|_\infty < 1$ are given by

$$K(s) = \begin{bmatrix} -Q(s) & I \end{bmatrix} \begin{bmatrix} T_1 & 0 \\ T_2 & I \end{bmatrix} \begin{bmatrix} F_1 & -I \\ F_2 & 0 \end{bmatrix}$$

for $Q \in R_c H_\infty$, $\|Q\|_\infty < 1$.

Problem FC: Full Control

The full control (FC) problem has P given by,

$$P(s) = \left[\begin{array}{c|c|c} A & B_1 & \begin{bmatrix} I & 0 \end{bmatrix} \\ \hline C_1 & D_{11} & \begin{bmatrix} 0 & I \end{bmatrix} \\ C_2 & D_{21} & \begin{bmatrix} 0 & 0 \end{bmatrix} \end{array}\right]$$

(3-18)

and is the dual of the Full Information case: the P for the FC problem has the same form as the transpose of P for the FI problem. The term *Full Control* is used because the controller has full access to both the state through output injection and to the output e. The only restriction on the controller is that it must work with the measurement y. The assumptions that the FC problem inherits from the output feedback problem are just the dual of those in the FI problem:

(A1) (C_2, A) is detectable.

(A2) D_{21} is full row rank with $\begin{bmatrix} D_{21} \\ \tilde{D}_\perp \end{bmatrix}$ unitary.

(A4) $\begin{bmatrix} A - j\omega I & B_1 \\ C_2 & D_{21} \end{bmatrix}$ has full row rank for all ω.

Necessary and sufficient conditions for the FC case are given in the following corollary. The family of all controllers can be obtained from the dual of Theorem 3.5 but these controllers will not be required in the sequel and are hence omitted.

Corollary 3.6. Suppose P is given by (3-18) and satisfies A1, A2 and A4. Then,

$$\exists K \text{ such that } \|T_{ed}\|_\infty < 1 \Leftrightarrow \mathrm{J}\infty \in dom(Ric),\ Ric(\mathrm{J}_\infty) \geq 0$$

H_∞ Output Feedback

The solution to the Full Information problem of the “H• Full Information and Full Control Problems” section is used in this section to solve the output feedback problem. First in Theorem 3.8 a so-called disturbance feedforward problem is solved. In this problem one component of the disturbance, d_2, can be estimated exactly from y using an observer, and the other component of the disturbance, d_1, does not affect the state or the output. The conditions for the existence of a controller satisfying a closed-loop H_∞-norm constraint is then identical to the FI case.

The solution to the general output feedback problem can then be derived from the transpose of Theorem 3.7 (Corollary 3.9) by a suitable change of variables which is based on X_∞ and the completion of the squares argument (see [GloD2]).

The main result is now stated in terms of the matrices defined in the “H• Full Information and Full Control Problems” section involving the solutions of the X_∞ and Y_∞ Riccati equations together with the *state feedback* and *output injection* matrices F and L. Assume that unitary changes of coordinates on ω and z have been carried out to give the following partitions of D, F_1 and L_1.

$$\left[\begin{array}{c|c} & F' \\ \hline L' & D \end{array}\right] = \left[\begin{array}{c|ccc} & F'_{11} & F'_{12} & F'_{2} \\ \hline L'_{11} & D_{1111} & D_{1112} & 0 \\ L'_{12} & D_{1121} & D_{1122} & I \\ L'_{2} & 0 & I & 0 \end{array}\right] \tag{3-19}$$

Theorem 3.7. *Suppose P satisfies the assumptions A1–A4 of the* “Problem Statement” *section.*

a There exists an admissible controller $K(s)$ such that $\|F_L(P,K)\|_\infty < \gamma$ *(i.e.,* $\|T_{ed}\|_\infty < \gamma$*)* if and only if

i $\gamma > max\ (\bar{\sigma}[D_{1111}, D_{1112}], \bar{\sigma}[D'_{1111}, D'_{1121}])$

ii $H_\infty \in dom(Ric)\ with\ X_\infty = Ric(H_\infty) \geq 0$

iii $J_\infty \in dom(Ric)\ with\ Y_\infty = Ric(J_\infty) \geq 0$

iv $\rho(X_\infty Y_\infty) < \gamma^2$.

e Given that the conditions of part *(a)* are satisfied, then all rational internally stabilizing controllers $K(s)$ satisfying $\|F_L(P,K)\|_\infty < \gamma$ are given by

$K = F_L(K_a, \Phi)$ *for arbitrary* $\Phi \in R_c H_\infty$ *such that* $\|\Phi\|_\infty < \gamma$

where

$$K_a = \left[\begin{array}{c|cc} \hat{A} & \hat{B}_1 & \hat{B}_2 \\ \hline \hat{C}_1 & \hat{D}_{11} & \hat{D}_{12} \\ \hat{C}_2 & \hat{D}_{21} & 0 \end{array}\right]$$

$$\hat{D}_{11} = -D_{1121}D'_{1111}(\gamma^2 I - D_{1111}D'_{1111})^{-1}D_{1112} - D_{1122},$$

$\hat{D}_{12} \in \mathbf{C}^{m_2 \times m_2}$ and $\hat{D}_{21} \in \mathbf{C}^{p_2 \times p_2}$ are any matrices (*e*.g., Cholesky factors) satisfying

$$\hat{D}_{12} \quad \hat{D}_{12} = I - D_{1121}(\gamma^2 I - D'_{1111}D_{1111})^{-1}D_{1121},$$
$$\hat{D}_{21} \quad \hat{D}_{21} = I - D'_{1112}(\gamma^2 I - D_{1111}D'_{1111})^{-1}D_{1112},$$

and

$$\begin{aligned} \hat{B}_2 &= Z_\infty^{-1}(B_2 + L_{12}) \quad \hat{D}_{12}, \\ \hat{C}_2 &= -\hat{D}_{21}(C_2 + F_{12}), \\ \hat{B}_1 &= -Z_\infty^{-1}L_2 + \hat{B}_2\hat{D}_{12}^{-1}\hat{D}_{11}, \\ \hat{C}_1 &= F_2 + \hat{D}_{11}\hat{D}_{21}^{-1}\hat{C}_2, \\ \hat{A} &= A + BF + \hat{B}_1\hat{D}_{21}^{-1}\hat{C}_2, \end{aligned}$$

where

$$Z_\infty = (I - \gamma^{-2}Y_\infty X_\infty).$$

(Note that if $D_{11} = 0$ then the formulae are considerably simplified.)

The proof of this main result is via some special problems that are simpler special cases of the general problem and can be derived from the FI and FC problems. A separation type argument can then give the solution to the general problem from these special problems. It can be assumed, without loss of generality, that $\gamma = 1$ since this is achieved by the scalings $\gamma^{-1}D_{11}$, $\gamma^{-1/2}B_1$, $\gamma^{-1/2}C_1$, $\gamma^{1/2}B_2$, $\gamma^{1/2}C_2$, $\gamma^{-1}X_\infty$, $\gamma^{-1}Y_\infty$ and $\gamma^{-1}K$. All the proofs will be given for the case $\gamma = 1$.

Disturbance Feedforward and Output Estimation

In the Disturbance Feedforward problem one component of the disturbance, d_1, does not affect the state or the output. The other component of the disturbance, d_2 (and hence the state x), can be estimated exactly from y using an observer. The conditions for the existence of a controller satisfying a closed-loop H_∞-norm constraint is then identical to the Full Information case.

Theorem 3.8. (Disturbance Feedforward)

Theorem 3.7 is true under the additional assumptions that

$$B_1\tilde{D}'_\perp = 0, \quad A - B_1D'_{21}C_2 \text{ is stable.} \tag{3-20}$$

In this case,

$$Y_\infty = 0, \; Z = I, \; L = -\begin{bmatrix} 0 & B_1D'_{21} \end{bmatrix}$$

The transpose of Theorem 3.8 can now be stated to obtain another special case of Theorem 3.7.

Corollary 3.9. (Output Estimation)

Theorem 3.7 is true under the additional assumptions that

$$D'_\perp C_1 = 0, \quad A - B_2D'_{12}C_1 \text{ is stable.}$$

In this case

$$X_\infty = 0, \; Z = I, \; F = -\begin{bmatrix} 0 \\ D'_{12}C_1 \end{bmatrix}$$

Converting Output Feedback to Output Estimation

The output feedback case when the disturbance, d, cannot be estimated from the output is reduced to the case of Corollary 3.9 by the change of variables

$$\|e\|_2^2 - \|d\|_2^2 = \|v\|_2^2 - \|r\|_2^2$$

where

$$\begin{aligned} v &= u + T_2 d - \left[T_2,\ I\right] F_x \\ r &= T_{1(2)}(d - F_1 x) \end{aligned}$$

We will perform the change of variables with v replacing e and r replacing d. Hence

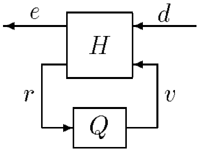
$$\begin{aligned} \dot{x} &= (A + B_1 F_1)x + B_1 T_1^{-1} r + B_2 u \\ v &= u + T_2 T_1^{-1} r - F_2 x \\ y &= C_2 x + D_{21} T_1^{-1} r + D_{21} F_1 x \end{aligned}$$

and the transfer matrix from $\begin{pmatrix} r \\ u \end{pmatrix}$ to $\begin{pmatrix} v \\ y \end{pmatrix}$ is

$$P_{vyru}(s) := \left[\begin{array}{c|cc} A + B_1 F_1 & B_1 T_1^{-1} & B_2 \\ \hline -F_2 & T_2 T_1^{-1} & I \\ C_2 + D_{21} F_1 & D_{21} T_1^{-1} & 0 \end{array}\right]$$

(3-21)

Similarly substituting v for u in the equation for P gives that the transfer function from $\begin{pmatrix} d \\ v \end{pmatrix}$ to $\begin{pmatrix} e \\ r \end{pmatrix}$ is H as defined by

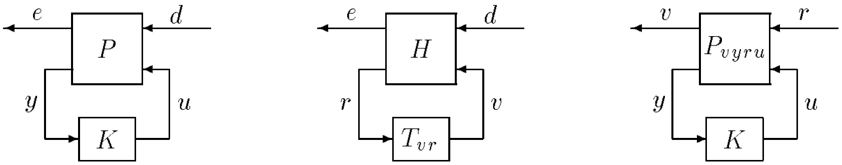


$$H = \left[\begin{array}{c|cc} A_F & B_1 - B_2T_2 & B_2 \\ \hline C_{1F_\infty} & D_\perp D'_\perp D_{11} & D_{12} \\ -T_1F_1 & T_1 & 0 \end{array}\right]$$

(3-22)

It can be shown that $H\sim H = I$ (since $\|e\|_2^2 - \|d\|_2^2 = \|v\|_2^2 - \|r\|_2^2$) and that A_F is stable.

We can show with a little algebra the equivalence of the first two of the following block diagrams, with $T_{vr} = F_L(P_{vyru},K)$ given by the third one.



Lemma 3.10. Let P satisfy A1–A4, and assume that X_∞ exists and $X_\infty \geq 0$. Then the following are equivalent:

- **a** K internally stabilizes P and $\|F_L(P,K)\|_\infty < 1$,
- **b** K internally stabilizes P_{vyru} and $\|F_L(P_{vyru},K)\|_\infty < 1$
- **c** K internally stabilizes P_{tmp} and $\|F_L(P_{\text{tmp}},K)\|_\infty < 1$,

where P_{vyru} is given by (3-21) and

$$P_{\text{tmp}} := \left[\begin{array}{c|cc} A + B_1F_1 & B_1 & B_2 \\ \hline -D_{12}F_2 & D_{11} & D_{12} \\ C_2 + D_{21}F_1 & D_{21} & 0 \end{array}\right].$$

The importance of the above constructions for P_{vyru} and P_{tmp} is that they satisfy the assumptions for the output estimation problem (Corollary 3.9) since $A + BF$ is stable.

Relaxing Assumptions A1–A4

In this section we indicate how the results of the "H• Output Feedback" section can be extended to more general cases by the relaxation of assumptions A1–A4. The optimal case is not considered, but the interested reader may refer to [GloD2].

Relaxing A3 and A4

Suppose that,

$$P = \left[\begin{array}{c|cc} 0 & 0 & 1 \\ \hline 0 & 0 & 1 \\ 1 & 1 & 0 \end{array}\right]$$

which violates both A3 and A4 and corresponds to the robust stabilization of an integrator. If the controller $u = -\varepsilon x$, for $\varepsilon > 0$ is used then

$$T_{ed} = \frac{-\varepsilon s}{s + \varepsilon}, \text{ with } \|T_{ed}\|_\infty = \varepsilon$$

Hence the norm can be made arbitrarily small as $\varepsilon \to 0$, but $\varepsilon = 0$ is not admissible since it is not stabilizing. This may be thought of as a case where the H_∞-optimum is not achieved on the set of admissible controllers. Of course, for this system, H_∞ *optimal* control is a silly problem, although the suboptimal case is not obviously so.

If you simply drop the requirement that controllers be admissible and remove assumptions A3 and A4, then the formulae in this chapter will yield $u = 0$ for both the optimal controller and the suboptimal controller with $\Phi = 0$. This illustrates that assumptions A3 and A4 are necessary for the techniques in this chapter to be directly applicable. An alternative is to develop a theory that maintains the same notion of admissibility, but relaxes A3 and A4. The easiest way to do this would be to pursue the suboptimal case introducing ε perturbations so that A3 and A4 are satisfied.

Relaxing A1

If assumption A1 is violated, then it is obvious that no admissible controllers exist. Suppose A1 is relaxed to allow unstabilizable and/or undetectable modes on the $j\omega$ axis, and internal stability is also relaxed to also allow closed-loop $j\omega$ axis poles, but A2–A4 is still satisfied. It can be easily shown that under these conditions the closed-loop H_∞ norm cannot be made finite, and in particular, that the unstabilizable and/or undetectable modes on the $j\omega$ axis must show up as poles in the closed-loop system.

Violating A1 and Either or Both of A3 and A4

Sensible control problems can be posed that violate A1 *and* either or both of A3 and A4. For example, cases when A has modes at $s = 0$, which are unstabilizable through B_2 and/or undetectable through C_2, arise when an integrator is included in a weight on a disturbance input or an error term. In these cases, either A3 or A4 is also violated, or the closed-loop H_∞ norm cannot be made finite. In many applications such problems can be reformulated so that the integrator occurs inside the loop (essentially using the internal model principle), and is hence detectable and stabilizable.

An alternative approach to such problems, which could potentially avoid the problem reformulation, would be to pursue the techniques in [GloD2], but relax internal stability to the requirement that all closed-loop modes be in the closed left half plane. Clearly, to have finite H_∞ norm these closed-loop modes could not appear as poles in T_{ed}. The formulae given in this chapter will often yield controllers compatible with these assumptions. You would then have to decide whether closed-loop poles on the imaginary axis were due to weights and hence acceptable or due to the problem being poorly posed as in the above example.

A third alternative is to again introduce ε perturbations so that A1, A3 and A4 are satisfied. Roughly speaking, this would produce sensible answers for sensible problems, but the behavior as $\varepsilon \rightarrow 0$ could be problematic.

Relaxing A2

In the case that either D_{12} is not full column rank or D_{21} is not full row rank, then improper controllers can give bounded H_∞-norm for T_{ed}, although will not be admissible as defined in the “Problem Statement” section. Such singular filtering and control problems have been well-studied in H_2 theory and many of the same techniques go over to the H_∞ -case (e.g., [Will2], [WilKS] and [HauS]). In particular the structure algorithm of [Silv] could be used to make the terms D_{12} and D_{21} full rank by the introduction of suitable differentiators in the controller.

Discrete-Time and Sampled-Data H_∞ Control

Discrete-Time Systems

The control of a discrete-time system described by the state difference equation

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ y_k &= Cx_k + Du_k \end{aligned}$$

is investigated in this section. The state difference equation corresponds to the transfer function,

$$P(z) = D + C(zI - A)^{-1}B$$

As in the continuous-time case, a controller with transfer function $K_d(z)$ can be synthesized to make the closed-loop, $H(z) = F_L(P(z),K_d(z))$ internally stable and

$$\|H\|_\infty := \sup_\theta \bar{\sigma}(H(e^{j\theta})) < \gamma$$

for any γ sufficiently large. This can be accomplished either directly in terms of the original data (A,B,C,D) or via the bilinear transformation,

$$z = \frac{1 + \frac{1}{2}sh}{1 - \frac{1}{2}sh},$$

which maps the unit disk in the z-plane into the left half of the s-plane for any $h > 0$. The μ-Tools command to synthesize discrete-time H_∞ controllers, `dhinfsyn`, uses this bilinear transformation and the corresponding continuous-time μ-Tools commands.

An additional consideration is which of the controllers K_d that make $\|H\|_\infty < \gamma$ should be chosen. The controller that maximizes the entropy integral,

$$I = \frac{\gamma^2}{2\pi}\int_{-\pi}^{\pi} \log\det(I - \gamma^{-2}H(e^{j\theta})^*H(e^{j\theta}))\frac{1 - \left|z_o^{-2}\right|}{\left|e^{j\theta} - z_o^{-1}\right|^2}d\theta$$

for any $|z_o| > 1$ can be calculated. The usual *central* controller, the default for `hinfsyne`, is taken as the one corresponding to $z_o = \infty$ and gives a measure of how far $H(e^{j\theta})$ is less than γ."

Sampled-Data Systems

Continuous-time systems where measurements are sampled and then the control signal calculated by a discrete-time controller followed by a hold are termed *sampled-data* systems. Two possible H_∞-type approaches to sampled data control law design are available in μ-Tools software.

Consider the system in Figure 3-7, where P is the continuous-time generalized plant, d is a continuous-time disturbance signal, e is a continuous-time error signal, y is the measurement to be sampled by the sampler S, with sampling period h, and u is the control signal, which is the output of the hold device, H, and is constant between sampling points.

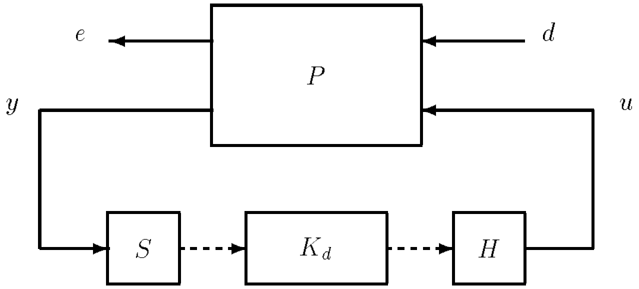


Figure 3-7: Sampled Data System Block Diagram

If it is assumed that the input d is in fact piecewise constant (synchronized with u), and that only the sampled values of e are of interest, then there is a discrete-time equivalent system for P that relates the discrete-time inputs and outputs. The controller can then be designed using discrete-time H_∞ techniques.

This approach has two potential problems; one is that the intersample behavior of the outputs is ignored and the other is that the inputs are assumed piecewise constant. An alternative approach is to require that the induced norm between the inputs d and outputs e be less than γ,

$$\sup_{w} \frac{\|e\|_{L_2}}{\|d\|_{L_2}} < \gamma$$

This will handle both the above difficulties and has been studied in detail by Bamieh and Pearson [BamP] along with a number of other researchers

[HarK,KabH,Toi,Yam]. It turns out that if γ is achievable then it can be achieved by a linear time-invariant controller of the same degree as the generalized plant, P. The computations for this controller involve calculating an equivalent discrete-time system and then using discrete-time H_∞ methods.

Discrete-Time and Sampled-Data Example

The following example is meant to illustrate the discrete-time and sampled-data norms involved rather than be representative of controller design. Consider the system in Figure 3-8 where,

$$F_1(s) = \frac{1}{(1+\tau_1 s)}$$

$$F_2(s) = \frac{\omega_o^2}{(s^2 + 2c\omega_o s + \omega_o^2)}$$

$$F_3(s) = \frac{1}{(1+\tau_3 s)}$$

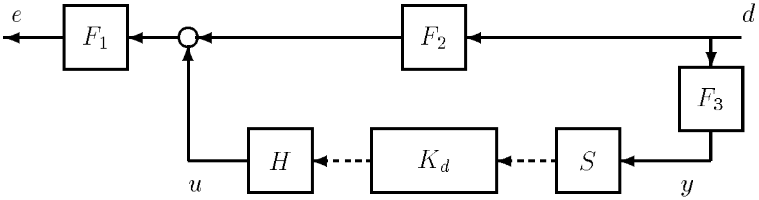


Figure 3-8: Sampled Data System Block Diagram for Simple Example

This gives the generalized plant,

$$P = \begin{bmatrix} P_{11}(s) & P_{12}(s) \\ P_{21}(s) & P_{22}(s) \end{bmatrix}, = \begin{bmatrix} F_1F_2 & F_1 \\ F_3 & 0 \end{bmatrix}$$

and the closed-loop is trying to match the output of F_2 by the controller output based on the sampled input to F_2 filtered by F_3. P can be defined as follows.

```
h = 0.1;
tau1 = 0.001;
om_o = 2*pi; c = 0.05;
tau3 = 0.01;
F1 = nd2sys(1,[tau1 1]);
F2 = nd2sys(1,[omo_o(-2) 2*c/omo_o 1]);
F3 = nd2sys(1,[tau3 1]);
p_ic = abv(mmult(F1,sbs(F2,1)),sbs(F3,0));
ncon = 1; nmeas = 1;
minfo(p_ic)
system: 4 states 2 outputs 2 inputs
```

The zero controller will result in a purely continuous-time system with induced norm given by $\|P_{11}(s)\|_\infty$.

```
hinfnorm(sel(p_ic,1,1))
norm between 10.01 and 10.02
achieved near 6.267
```

Now let us design a controller for this sampled-data system using the corresponding sample and hold discrete-time system. The variable delay corresponds to the number of sample delays in the controller.

```
gmin =.001; delay = 0;
gmax = 1;
tol = 0.001; tol2 = 0.001;
p_ic_sh = samhld(p_ic,h);
if delay>0,
 p_ic_sh=mmult(daug(1,nd2sys(1,eye(1,delay+1))),p_ic_sh;
end
[k_d,g_d,gfin_d] = ...
 dhfsyn(p_ic_sh,nmeas,ncon,gmin,gmax,tol,h,inf,-1,-2);
 Test bounds:0.0010 < gamma <=1.0000

gamma    hamx_eig   xinf_eig  hamy–eig  yinf–eig   nrho–xy   p/f
1.000     3.5e–01    0.0e+00   3.5e–01  –3.9e–16    0.0337    p
0.500     3.5e–01    0.0e+00   3.5e–01  –1.1e–15    0.1345    p
0.251     3.5e–01   –8.4e–14   3.5e–01  –7.1e–14    0.5357    p
0.126     3.5e–01    0.0e+00   3.5e–01  –4.1e–16    2.1257#   f
0.188     3.5e–01   –7.5e–14   3.5e–01   0.0e+00    0.9498    p
0.157     3.5e–01    0.0e+00   3.5e–01  –3.6e–16    1.3648#   f
0.173     3.5e–01   –1.1e–15   3.5e–01  –8.7e–14    1.1292#   f
0.181     3.5e–01    0.0e+00   3.5e–01   0.0e+00    1.0337#   f
0.184     3.5e–01    0.0e+00   3.5e–01   0.0e+00    0.9904    p
0.182     3.5e–01   –3.5e–14   3.5e–01  –3.8e–16    1.0117#   f
0.183     3.5e–01   –3.0e–16   3.5e–01   0+0e+00    1.0010#   f

 Gamma value achieved: 0.1844

 dhfnorm(g_d,tol2,h)
 norm between 0.1835 and 0.1837
 achieved near 31.42
```

The induced norm of the sampled-data system from continuous-time inputs to continuous-time outputs can now be calculated using the command sdhfnorm.

```
[gaml_d,gamu_d] = sdhfnorm(p_ic,k_d,h,delay,tol2)
gaml_d =
   2.0250e+01
gamu_d =
   2.0267e+01
```

The discrete-time system with piecewise constant inputs and ignoring intersample behavior has an induced norm of only 0.1837 whereas if the input is allowed to vary during the sampling period the gain can be made 100 times larger at 20.2. A suboptimal controller for the sampled-data system can also be calculated using sdhfsyn.

```
 [k_sd,gfin_sd]=sdhfsyn(p_ic,1,1,gmin,gamu_d,tol,h,delay,-2);
 Test bounds:0.0010 < gamma <=20.2670

gamma   hamx_eig  xinf_eig  hamy–eig  yinf–eig   nrho_xy  p/f
20.267   3.5e–01  –1.6e–30 3.le–01     1.0e–09    0.0016  p
10.134   3.4e–01   1.3e–31 1.6e–01     1 0e–09    0.0087  p
 5.067   3.3e–01  –2.7e–20 8.3e–17#    *******   *******  f
 7.601   3.4e–01   2.3e–20 3.8e–15#    *******   *******  f
 8.867   3.4e–01   3.7e–21 1.2e–14#    *******   *******  f
 9.501   3.4e–01   l.le–l9 1.2e–01     1.0e–09    0.0109  p
 9.184   3.4e–01  –5.3e–20 7.6e–02     1.0e–09    0.0128  p
 9.026   3.4e–01   5.5e–21 4.2e–02     1.0e–09    0.0145  p
 8.947   3.4e–01   1.3e–31 9.4e–14#    *******   *******  f
 8.986   3.4e–01  –7.7e–20 2.7e–02     1.0e–09    0.0153  p
 8.966   3.4e–01  –1.5e–31 1.3e–02     1 0e–09    0.0160  p
 8.956   3.4e–01   4.1e–20 3.9e–13#    *******   *******  f
 8.961   3.4e–01   4.9e-20 6.6e-03     1 0e–09    0.0163  p
 8.959   3.4e–01   7.6e-20 8.7e-14#    *******   *******  f
 8.960   3.4e–01   3.5e-03 3.2e-03     1 0e–09    0.0165  p
 8.960   3.4e–01  -2.3e-20 9.0e-14#    *******   *******  f

 Gamma value achieved:8.9601
```

and the norm checked by

```
[gaml_sd,gamu_sd] = sdhfnorm(p_ic,k_sd,h,delay,tol2)
gaml_sd =
   8.9445e+00
gamu_sd =
   9.0028e+00
```

The initial design gave very low estimates of the possible gain in the system. The latter design indicates that no controller can give a low gain with this sampled-data problem. The main difficulty with this particular problem is that the filter $F_3(s)$ has too high a bandwidth and this gives a high potential gain for F_3 followed by the sampler. In contrast to the continuous-time case, the calculation of a worst-case disturbance in the sampled-data case is not straightforward. However the time domain simulation of the system is now performed to illustrate the reason for its high gain.

```
tfinal = 10;
t1 = (0:h/100:tfinal)';
[nr,nc] = size(t1);
w1 = zeros(nr,nc);
for i = 1:length(t1)/200,
w1(100*i-20:100*i) = ...
cos(i*pi/5-pi/6)*exp(h*(-20:1:0)/(100*tau3))';...
end
w = vpck(w1,t1);
[z_d,y_d,u_d] = sdtrsp(p_ic,k-d,w,h,tfinal,h/100);
vplot(z_d,'-',y_d,'.')
gain_d = norm(vunpck(z-d),2)/vnorm(vunpck(w),2)
gain_d =
   1.2924e+01
```

The time responses are given in Figure 3-9 and the high gain achieved by the disturbance being large just before the sampling instant and zero elsewhere, hence having a relatively low total energy.

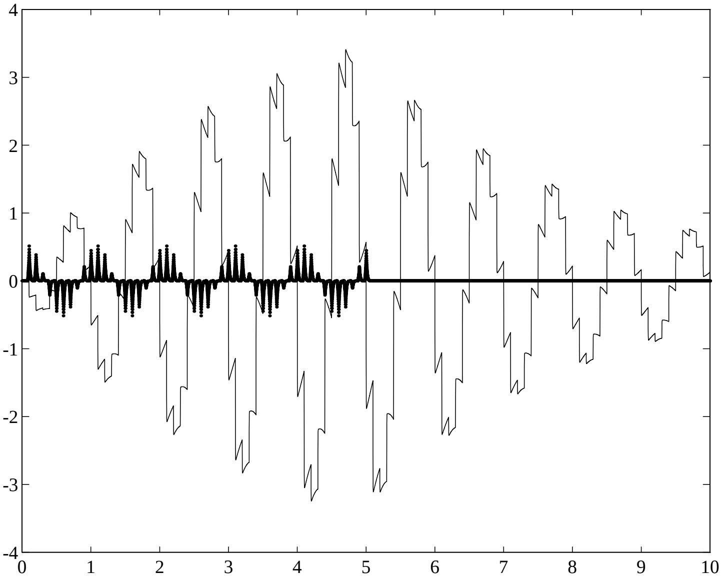


Figure 3-9: Discrete-Time Controller Time Response

The suboptimal sampled-data controller can be simulated with the same input as follows:

```
[z_sd,y_sd,u_sd] = sdtrsp(p_ic,k_sd,w,h,tfinal,h/100);
vplot(z_sd,'-',y_sd,'.')
gain_sd = norm(vunpck(z_sd),2)/vnorm(vunpck(w),2)
gain_sd =
   6.3332e-01
```

The time responses are given in Figure 3-10. The gain in this instance is much lower for this particular input. However other inputs can give a gain of close to nine.

The same example can be repeated for different values of the sampling period, *h*, and the controller delay, and for variations in the time constants. The two controllers often give very similar results, however, the discrete-time results obtained from `samhld` and `dhfsyn` can give optimistic gain estimates when compared with those obtained by `sdhfsyn`.

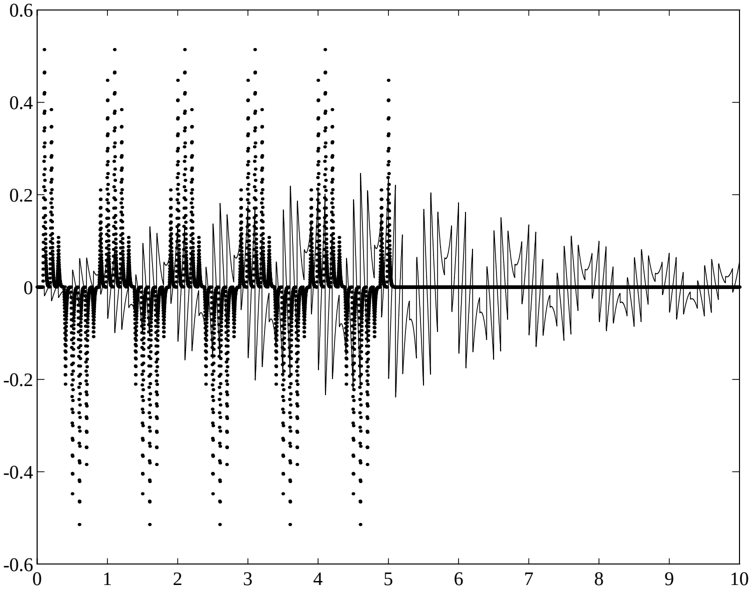


Figure 3-10: Sampled Data Controller Time Response

Loop Shaping Using H_∞ Synthesis

A particularly straightforward method of designing controllers is to use a combination of loop shaping and robust stabilization as proposed in McFarlane and Glover [McFG_1, McFG2]. Given a system with transfer function $G(s)$ the problem set up is given in Figure 3-11.

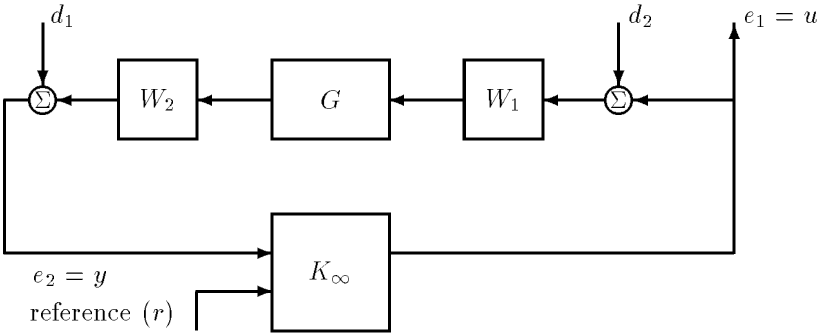


Figure 3-11: H_∞ Loop Shaping Standard Block Diagram

The first step is to design a pre-compensator $W_1(s)$, so that the gain of $W_2(s)G(s)W_1(s)$ is sufficiently high at frequencies where good disturbance attenuation is required and is sufficiently low at frequencies where good robust stability is required. The second step is to design a feedback controller, K_∞, so that

$$\frac{1}{b(W_2GW_1,K_\infty)} := \left\| \begin{bmatrix} I \\ K_\infty \end{bmatrix} (I - W_2GW_1,K_\infty)^{-1}[W_2GW_1, I] \right\|_\infty \eth \frac{1}{\varepsilon}$$

which will also give robust stability of the perturbed weighted plant

$$(N + \Delta_1)(M + \Delta_2)^{-1} \text{ for } \left\| \begin{bmatrix} \Delta_1 \\ \Delta_2 \end{bmatrix} \right\|_\infty < b(W_2GW_1,K_\infty)$$

where $NM^{-1} = W_2GW_1$ is a normalized coprime factorization satisfying $N(j\omega)^*N(j\omega) + M(j\omega)^*M(j\omega) = I$. This stability margin is always less than 1 and gives a good indication of robust stability to a wide class unstructured perturbations. Stability margin values of $\varepsilon > 0.2 – 0.3$ are generally considered satisfactory. The closed-loop H_∞-norm performance objective has the standard signal gain interpretation. Finally it can be shown that the controller, K_∞, does not substantially affect the loop shape in frequencies where the gain of W_2GW_1 is either high or low, and will guarantee satisfactory stability margins in the frequency region of gain crossover. In the regulator setup, the final controller to be implemented is $W_1K_\infty W_2$ in the feedback configuration shown in Figure 3-12.

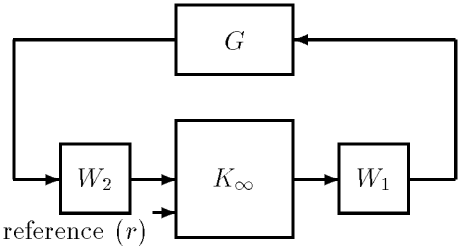


Figure 3-12: H_∞ Loop Shaping Controller Implementation

Closely related to the above problem is the approach to uncertainty using the gap family of metrics. These metrics, $\delta(G_0,G_1)$, give a numerical value between 0 and 1 for the distance between any two linear systems $G_0(s)$ and $G_1(s)$ that have the same number of inputs and outputs. The gap metric was introduced into the control literature by Zames and El-Sakkary [ZamE] and exploited by Georgiou and Smith [GeoSm]. It can be computed using an H_∞ optimization problem [Geo]. An interesting new metric, ν-gap (*nugap*), was derived by Vinnicombe [Vin] and has a frequency response interpretation. For both of these metrics, the following robust performance result holds [QiuD, Vin].

$$\arcsin b(G_1, K_1) \geq \arcsin b(G_0, K_0) - \arcsin\delta_\nu(G_0, G_1) - \arcsin\delta_\nu(K_0, K_1) \qquad \textbf{(3-23)}$$

where

$$b(G, K) = \left\| \begin{bmatrix} I \\ K \end{bmatrix} (I - GK)^{-1} \begin{bmatrix} G & I \end{bmatrix} \right\|_\infty^{-1}.$$

The nugap is always less than or equal to the gap, so its predictions using the above robustness results are tighter. To make use of the gap metrics in robust design, weighting functions need to be introduced. In the above robustness result G needs to be replaced by W_2GW_1 and K by $W_1^{-1}KW_2^{-1}$ (similarly for G_0, G_1, K_0, and K1). This makes the weighting functions compatible with the weighting structure in the loop shaping H_∞ synthesis.

The interpretation of this result is that if a nominal plant G_0 is stabilized by controller K_0, with *stability margin* b(G_0,K_0), then the stability margin when G_0 is perturbed to G_1 and K_0 is perturbed to K_1 is degraded by no more than the above formula. Note that $1/b(G,K)$ is also the signal gain from disturbances on the plant input and output to the input and output of the controller. Model reduction of the system model and controller can be performed by using balanced truncations or Hankel norm approximation of normalized coprime factor representation.

If a reference signal (r) is available, then there is a variety of methods to incorporate the tracking error into the objective. One effective procedure is to choose a coprime factorization of $K_\infty = UV^{-1}$ and let $M = V^{-1}(r + U_y)$ where r is the reference and y the measurement. U and V can be obtained from a particular observer form of K_∞ in such a way that with closed-loop $y = Nr$ and $G = NM^{-1}$ is a normalized coprime factorization [Vin]. The DC. gain of this controller could then be adjusted to give zero steady state tracking error. Alternatively, this nominal command response could be diagonalized and a further command compensation performed. These ideas are applied to the "HIMAT Robust Performance Design Example" section in Chapter 7.

Model Reduction

It is often desirable to approximate a state-space representation of a system with a lower order state-space representation. This procedure is referred to as a *model reduction*. The μ-Analysis and Synthesis Toolbox (μ-Tools) provides several commands to aid in reducing the order of a system. These are discussed in this section along with an example to illustrate their use.

Given a SYSTEM matrix `[A B; C D]`, the simplest method of model reduction is to truncate a part of the SYSTEM `A` matrix and remove the corresponding columns and rows of the `B` and `C` matrices. The command `strunc` performs this function. You should be careful to order the modes of the `A` matrix and truncate modes that do not significantly affect the system response. The command `strans` is useful in this context as it transforms the `A` matrix into block diagonal form with 1 × 1 or 2 × 2 blocks corresponding to the respectively real and complex poles in order of increasing magnitude. This is often done prior to truncating high frequency modes.

Truncating high frequency modes will also affect the low frequency response of the various transfer functions. The command `sresid` can be used to residualize the truncated modes and compensate for the zero frequency contribution of each truncated mode with an additional `D` matrix term in the resulting reduced order SYSTEM matrix.

More advanced model reduction techniques for stable systems can be performed with the μ-Tools commands `sysbal` and `hankmr`. `sysbal` performs a balanced realization on the input SYSTEM matrix, which entails balancing the observability and controllability Grammians (for a more detailed discussion see [Enn], [Glo1] and [Moo]). In its simplest form, this command will remove all unobservable and/or uncontrollable modes. `sysbal` also returns a vector of the Hankel singular values of the system, which can be used to further truncate the modes of the SYSTEM.

The following example illustrates how `sysbal` can be used to remove unobservable and uncontrollable modes. Two systems, `P` and `C`, are created and then interconnected with unity gain negative feedback. The closed-loop system, `clp`, is given by

$$\mathrm{clp} = \frac{PC}{1 + PC}$$

Suppose that

```
P = nd2sys(1,[1,1,1]);
C = nd2sys(1,[1,1]);
```

The correct way to form this closed-loop transfer function is to use

```
clpc = starp(mmult([1;1],mmult(mmult(P,C),[1,-1])),1,1,1);
```

The closed-loop SYSTEM matrix `clp` has three states. However, suppose instead the closed-loop system is formed as follows.

```
clp = mmult(P,C,minv(madd(1,mmult(P,C))));
minfo(clp)
system: 6 states   1 outputs   1 inputs
rifd(spoles(clp))

         real         imaginary         frequency           damping
  –5.0000e–01      –8.6603e–01       1.0000e+00        5.0000e–01
  –5.0000e–01       8.6603e–01       1.0000e+00        5.0000e–01
  –1.0000e+00       0.0000e+00       1.0000e+00        1.0000e+00
  –2.2816e–01      –1.1151e+00       1.1382e+00        2.0045e–01
  –2.2816e–01       1.1151e+00       1.1382e+00        2.0045e–01
  –1.5437e+00       0.0000e+00       1.5437e+00        1.0000e+00
```

The closed-loop system, `clp`, contains the open-loop poles of p as well as the closed-loop poles. Interconnecting systems with the commands `mmult` and `madd` often lead to nonminimal realizations. You can see that the open-loop poles are unobservable and/or uncontrollable by using `sysbal` with its second input, the truncation tolerance, set to zero. The output gives the Hankel singular values that are strictly greater than this tolerance together with a truncated balanced realization of this order. (A strictly positive default tolerance is also available.) Note that only five values are returned, the sixth being calculated as being identically zero, and the fourth and fifth are both zero to machine accuracy. `strunc` is then run to remove these two modes. Finally the H_∞-norm of the error between the system with the nonminimal modes truncated and the system formed using starp is calculated as being essentially zero.

```
[clpr,hanksv] = sysbal(clp,0);
disp(hanksv')
6.7732e-014.7580e-014.8484e-028.0919e-17
1.5746e-18
clpr = strunc(clpr,3);
minfo(clpr)
system:    3 states    1 outputs   1 inputs
rifd(spoles(clpr))
```

```
          real        imaginary        frequency           damping

   –2.2816e–01      –1.1151e+00       1.1382e+00        2.0045e–01

   –2.2816e–01       1.1151e+00       1.1382e+00        2.0045e–01

   –1.5437e+00       0.0000e+00       1.5437e+00        1.0000e+00
```

```
hinfnorm(msub(clpc,clpr))
norm between 4.163e-16 and 4.167e-16
achieved near 0
```

A variety of norms is available to measure the error in reducing the model order. The Hankel-norm and H_∞-norms have relatively complete theories. Note that the Hankel norm is the maximum of the Hankel singular values and is hence available from sysbal. The command hankmr calculates the optimal Hankel norm approximation, of a specified order, to the input system. This method is presented in detail in [Glo1]. The calling sequence is

```
[sysb,sig] = sysbal(sys)
[sysh,sysu] = hankmr(sysb,sig,k,'d')
```

The sysh is stable with k poles, sysu is unstable (or anticausal), and $\|$sys - sysh - sysu$\|_\infty$ = sig(k + 1). This answer is optimal. If the 'd' option of hankmr is used then the following bound is guaranteed,

```
sig(k + 1) ≤ ||sys - sysh|| ∞≤ sig(k + 1) + sig(k + 2) + ...
```

The lower bound holds for any sysh of degree k and for truncated balanced realizations the upper bound needs to be doubled.

The following example creates a 15-state system and examines three, 7-state reduced-order models generated by sysbal, hankmr, and sresid. First generate the system, which consists of 10 real poles, a resonant pole pair, and first-order and second-order high frequency all-pass terms.

```
a = -diag([.03.05.1.2.3.4 1 3 5 10]);
b = [.03.05.1.2.3.4 1 3 5 10]'; c = ones(1,10);
d = 0.001;
sys1 = pck(a,b,c,d);
sys2 = nd2sys([1.1.4],[1.1.1]);
sys3 = nd2sys([1 -3 1000],[1 3 1000]);
sys4 = nd2sys([1 -20],[1 20]);
sys = mmult(sys1,sys2,sys3,sys4);
minfo(sys)
system:    15 states   1 outputs   1 inputs
```

The frequency response of sys is calculated and plotted in Figure 3-13.

```
omega = logspace(-1,3,60);
sys_g = frsp(sys,omega);
vplot('bode',sys_g);
title('Original system to be reduced')
```

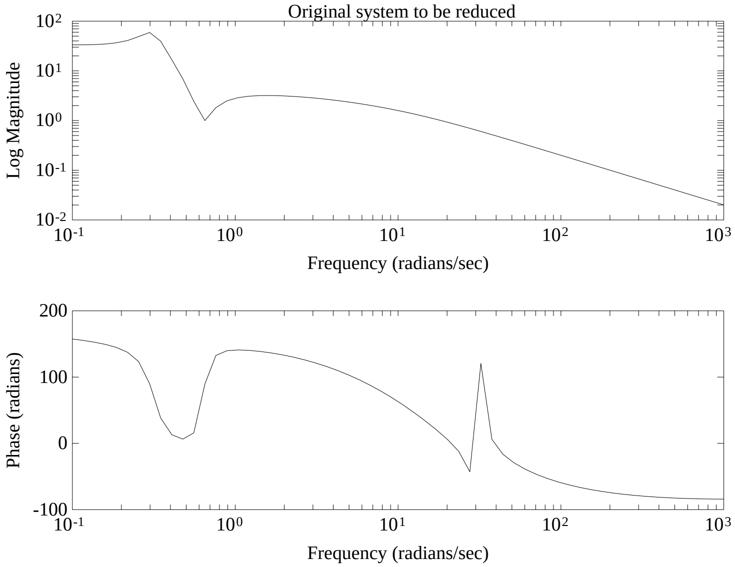


Figure 3-13: Original System to be Reduced via Model Reduction Techniques

Next the balanced realization is formed and truncated to seven states. Its H_∞-norm error is compared with the upper and lower bounds.

The Bode plot of the original system and the seven state, balanced realization model sysb7 is shown in Figure 3-14.

```
[sysb,sig] = sysbal(sys);
[mattype,p,m,n] = minfo(sysb);
disp(sig')
Columns 1 through 5
3.6422e+01 2.4906e+01  6.1381e+00  2.0261e+00  7.1689e-01
Columns 6 through 11
6.9421e-01 6.0281e-01  4.2291e-01  1.1884e-01  3.4276e-02
Columns 12 through 15
9.0070e-03 2.4834e-03  4.0909e-04  1.4544e-04  3.9050e-06
k = 7;
sysb7 = strunc(sysb,k);
sysb7_g = frsp(sysb7,omega);
vplot('bode',sys_g,sysb7_g) title(['Model reduction example:
  Frequency domain'])
tmp = hinfnorm(msub(sys_g,sysb7_g));
disp(['H-inf error = ' num2str(tmp)])
  H-inf error = 0.8553
disp(['lower bound = ' num2str(sig(k+1))])
  lower bound = 0.4229
disp(['upper bound = ' num2str(2*sum(sig(k+1:n)))])
  upper bound = 1.176
```

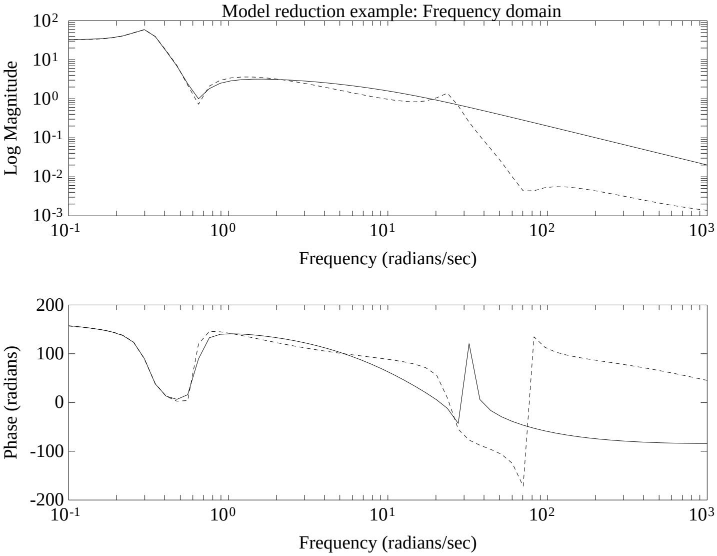


Figure 3-14: Balanced Realization Model Reduction: Original System (solid), Balanced (dashed)

The seven-state optimal Hankel norm approximation to sys is calculated with hankmr and placed in sysha7. Figure 3-15 contains the Bode plots of original system, the balanced realization model, and the seven-state Hankel norm model.

```
kh = 7;
sysha7 = hankmr(sysb,sig,kh,'d');
sysha7_g = frsp(sysha7,omega);
tmp = hinfnorm(msub(sys_g,sysha7_g));
disp(['H-inf error = ' num2str(tmp)])
H-inf error = 0.4351
disp(['lower bound = ' num2str(sig(kh+1))])
lower bound = 0.4229
disp(['upper bound = ' num2str(sum(sig(kh+1:n)))])
upper bound = 0.5881
vplot('bode',sys_g,sysb7_g,sysha7_g)
title(['Three different model reduction techniques'])
```

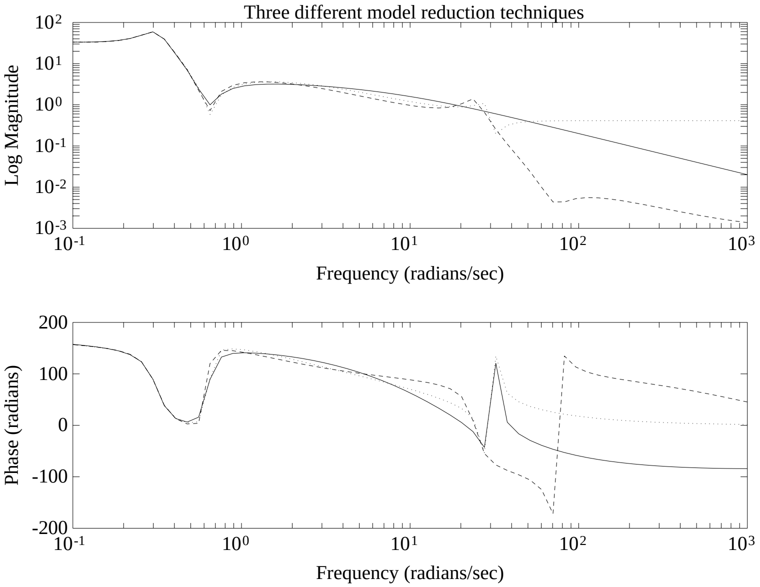


Figure 3-15: Model Reduction: Original System (solid), Balanced (dashed), Hankel (dotted)

Now obtain a truncated residualization. It turns out that the first seven poles with smallest modulus also have the largest H_∞-norms and hence no reordering of the poles after strans is required.

```
sysr = strans(sys);
sysrt7 = sresid(sysr,7);
sysrt7_g = frsp(sysrt7,omega);
tmp = hinfnorm(msub(sys_g,sysrt7_g));
disp(['H-inf error = ' num2str(tmp)])
H-inf error = 7.782
vplot('bode',sys_g,sysb7_g,sysha7_g,sysrt7_g)
title(['Four different model reduction techniques'])
```

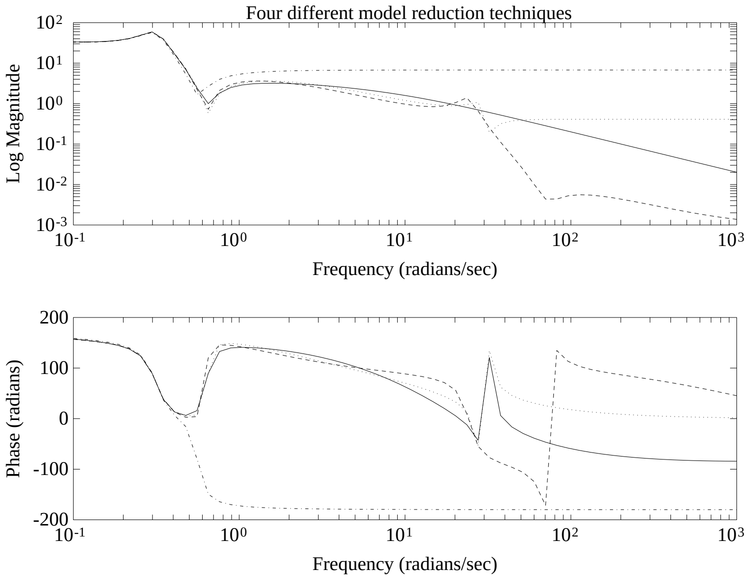


Figure 3-16: Model Reduction: Original System (solid), Balanced (dashed), Hankel (dotted), Residualized (dashed-dotted)

The four different model Bode plots are shown in Figure 3-16. The output from `hankmr` is nearly H_∞-optimal and that from `sysbal` has a similar error and does not change the D matrix. The frequency responses are plotted below with `sys`-solid, `sysbal`-dashed, `hankmr`-dotted, and `resid`-dash/dot. Note that as expected `sresid` matches well at low frequency but not at high frequency.

The time responses of the four systems, shown in Figure 3-17, are now compared in response to a 1 second pulse. The same line types are used for the display of the time domain responses. The response of `sysrt7` does not match `sys` well over the first 2 seconds but after 2 seconds the match is good.

```
pulse = siggen('t<1',[0:.1:10]);
ysys = trsp(sys,pulse);
integration step size: 0.003162
interpolating input vector (zero order hold)
ysysb7 = trsp(sysb7,pulse);
integration step size: 0.004127
interpolating input vector (zero order hold)
ysysha7 = trsp(sysha7,pulse);
integration step size: 0.003548
interpolating input vector (zero order hold)
ysysrt7 = trsp(sysrt7,pulse);
integration step size: 0.1
vplot(ysys,'-',ysysb7,':',ysysha7,'--',ysysrt7,'-.');
xlabel('Time: seconds')
title('Model reduction example: time domain')
```

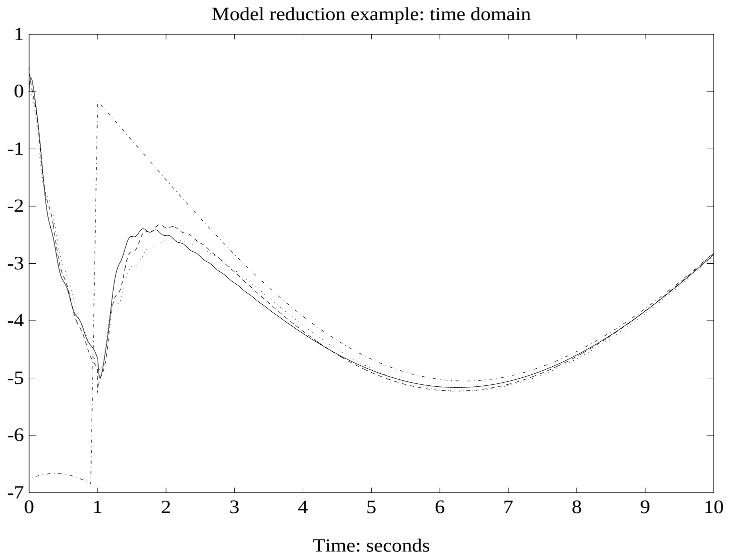


Figure 3-17: Time Response of the Original System and the Three Reduced Order Models

Note You should not draw general conclusions from this one example as to the relative merits of the different schemes.

The H_∞ norm of the error is not always appropriate, for example, in the system above none of the methods accurately matches the Bode diagram at high frequencies. It is therefore desirable to generate reduced-order models whose frequency weighted error is small. Two methods are available in μ-Tools to assist in this. The first method is based on frequency weighted Hankel norm approximation as proposed in Latham and Anderson [LatA] and finds $\hat{G}$ of degree k to minimize the Hankel norm of the stable part of $W_1^{\sim-1}(G-\hat{G})W_2^{\sim-1}$. Note that $W_1(s)^{\sim}$ is defined as. $W_1(-s)'$. This is implemented in the functions sfrwtbal and sfrwtbld. G is required to be stable and the weights need to be square, stable, and minimum phase. sfrwtbal then finds a balanced realization of the stable part of $W_1^{\sim-1}GW_2^{\sim-1}$ together with its Hankel singular values, which in this case also provide lower bounds on the achievable error. The resulting balanced system is approximated using hankmr (or another method if preferred) and $\hat{G}$ constructed using sfrwtbld. This is illustrated on the 15-state example with a sixth order approximation and with the relative error criterion, although it is not restricted to this case.

```
wt1 = mmult(sys1,sys2); wt2 = 1; k = 6; n = 15;
[sysfrwtbal,sigfrwt] = sfrwtbal(sys,wt1,wt2);
disp(sigfrwt')
Columns 1 through 5
1.0000e+00 1.0000e+00  1.0000e+00  9.9697e-01  9.3961e-01
Columns 6 through 11
9.1022e-01 2.2177e-01  5.9077e-02  1.7448e-02  4.5526e-03
Columns 12 through 15
1.2419e-03 2.1046e-04  8.1357e-05  1.2386e-05  2.9765e-07
sysfrwtk = hankmr(sysfrwtbal,sigfrwt,k,'d');
sysfrwthat = sfrwtbld(sysfrwtk,wt1,wt2);
sysfrwthat_g = frsp(sysfrwthat,omega);
disp(hinfnorm(msub(1,vrdiv(sysfrwthat_g,sys_g))))
3.2401e-01
vplot('bode',sys_g,'-',sysfrwthat_g,':')
title('sys_g (-) and sysfrwthat_g (:)')
```

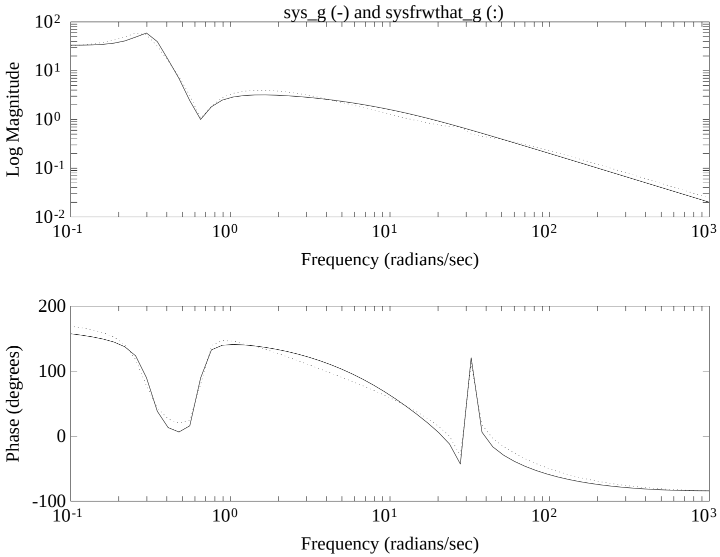


Figure 3-18: Frequency Response of the Original System and Frequency Weighted Balanced Reduction with sfrwtbld

The frequency weighted error is close to the unachievable lower bound of sigfrwt(k+1). A plot of this is shown in Figure 3-18. You can verify that this method is optimal by checking that the Hankel norm of the stable part of the weighted error is sigfrwt(k+1), using sfrwtbal as follows. (Note that these routines use sdecomp, which decomposes a system into the sum of two systems with poles to the left and right of a vertical line in the complex plane.)

```
[sysfrwterr,sigfrwterr] =...
sfrwtbal(msub(sys,sysfrwthat),wt1,wt2);
disp(max(sigfrwterr));
2.2177e-01
```

A method that is particularly appropriate for relative error model reduction is obtained using srelbal. The error criterion is the same as in the general frequency weighted case but with $W_1 = I$ and W_2 satisfying $W_2 W_2^{\sim} = G^{\sim} G$. This method is related to balanced stochastic truncation as introduced by Desai and Pal [DesP]. The lower bounds can be obtained as above and upper bounds have been derived by Green [Gre] and improved by Wang and Safonov [WanS]. There is no requirement that the system be square or minimum phase but it must not have fewer columns than rows. The resulting realization can be truncated as before.

```
[sysrelbal,sigrel] = srelbal(sys);
sysrelbalk = strunc(sysrelbal,k);
sysrelbalk_g = frsp(sysrelbalk,omega);
disp(hinfnorm(msub(1,vrdiv(sysrelbalk_g,sys_g))));
4.7480e-01
vplot('bode',sys_g,'-',sysrelbalk_g,'--',sysfrwthat_g,':')
title('model reduction example: relative error')
```

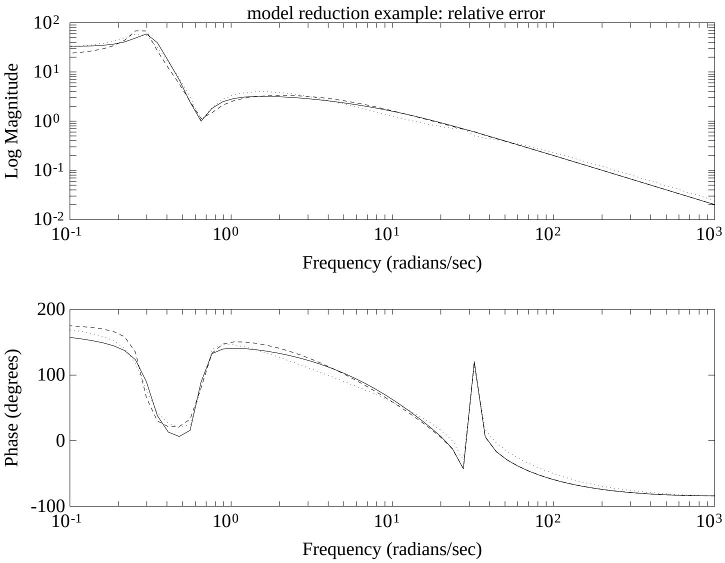


Figure 3-19: Frequency Response of the Original System and Reduced Order Models Using Relative Error Methods

It can be checked that sigrel equals sigfrwt above in this case. Both methods perform well with results close to the lower bound and similar frequency responses as seen in Figure 3-19. Glover [Glo2] suggests a combination of additive and relative error by performing relative error model reduction of the augmented system, $\begin{bmatrix} G \\ \alpha I \end{bmatrix}$. Relative error is recovered if $\alpha = 0$, and additive error as $\alpha \to \infty$. The resulting approximation can be thought of as satisfying

$\hat{G} = (I + \Delta_r)G + \alpha\Delta_a$ where $\left\| \begin{bmatrix} \Delta_r & \Delta_a \end{bmatrix} \right\|_\infty$ has been made small. This is easily implemented using srelbal as follows to give a seventh order fit with a performance between the two extremes.

```
k = 7; alpha = 15;
[sysrelbal,sigrel] = srelbal(abv(sys,alpha*eye(1)));
sysrelbalk = sel(strunc(sysrelbal,k),1,1);
```

You can evaluate the error in the same manner as was done in the frequency weighted balanced reduction case.

References

[AdAK:] Adamjan, V.M., D.Z. Arov, and M.G. Krein, “Infinite block Hankel matrices and related extension problems,” *AMS Transl.*, vol. 111, pp. 133-156, 1978.

[And:] Anderson, B.D.O., “An algebraic solution to the spectral factorization problem,” *IEEE Transactions on Automatic Control*, vol. AC-12, pp. 410-414, 1967.

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

[BallH:] Ball, J.A. and J.W. Helton, “A Beurling-Lax theorem for the Lie group U(m,n) which contains most classical interpolation theory,” *J. Op. Theory*, vol. 9, pp. 107-142, 1983.

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

[BoyBk:] Boyd, S., V. Balakrishnan, and P. Kabamba, “A bisection method for computing the H_∞ norm of a transfer matrix and related problems,” *Math. Control, Signals, and Systems*, vol. 2, no. 3, pp. 207-220, 1989.

[ClemG:] Clements, D.J. and *K*. Glover, “Spectral Factorization via Hermitian Pencils,” *Linear Algebra and its Applications*, Linear Systems Special Issue, pp. 797-846, Sept. 1989.

[DesP:] Desai, U.B. and D. Pal, “A transformation approach to stochastic model reduction,” *IEEE Transactions on Automatic Control*, vol. AC-29, pp. 1097-1100, 1984.

[Doy1:] Doyle, J.C., “Lecture notes in advances in multivariable control,” *ONR/Honeywell Workshop*, Minneapolis, 1984.

[DoyFT:] Doyle, J.C. , B.A. Francis, and A.R. Tannenbaum, *Feedback Control Theory*, Macmillian Publishing Company, New York, Toronto, 1992.

[DoyGKF:] Doyle, J.C., K. Glover, P. Khargonekar, and B. Francis, “State-space solutions to standard H_2 and H_∞ control problems,” *IEEE Transactions on Automatic Control*, vol. AC-34, no. 8, pp. 831-847, August 1989.

[Enn:] Enns, D., “Model reduction for control system design,” Ph.D. dissertation, Stanford University, June, 1984.

[FoisT:] Foias, C., and A. Tannenbaum, “On the four-block problem, I,” in *Operator Theory: Advances and Applications*, vol. 32, pp. 93-122, 1988 and “On the four-block problem, II: The singular system,” *Int. Equations Operator Theory*, vol. 11, pp. 726-767, 1988.

[Fran1:] Francis, B.A., *A course in H_∞ control theory*, Lecture Notes in Control and Information Sciences, vol. 88, Springer-Verlag, Berlin, 1987.

[FranD:] Francis, B.A. and J.C. Doyle, “Linear control theory with an H_∞ optimality criterion,” *SIAM J. Control and Opt.*, vol. 25, pp. 815-844, 1987.

[GeoS:] T. Georgiou and M. Smith, “Robust stabilization in the gap metric: Controller design for distributed plants,” *IEEE Transactions on Automatic Control*, pp. 1133-1143, vol. AC-37., no. 8, Aug. 1992.

[Glo1:] Glover, K., “All optimal Hankel-norm approximations of linear multivariable systems and their L_∞ error bounds,” *International Journal of Control*, vol. 39, pp. 1115-1193, 1984.

[Glo2:] K. Glover, “Multiplicative approximation of linear multivariable systems with L_∞ error bounds,” *Proceedings of the American Control Conference*, Seattle, pp. 1705-1709, 1986.

[Glo3:] Glover, K., “Tutorial on Hankel-norm approximation,” in *From Data to Model* (J.C. Willems ed.), Springer-Verlag, pp. 26-48, 1989.

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

[GloD2:] Glover, K. and J. Doyle, “A state-space approach to H_∞ optimal control,” in *Three Decades of Mathematical Systems Theory: A Collection of Surveys at the Occasion of the 50th Birthday of Jan C. Willems*, H. Nijmeijer and J.M. Schumacher (Eds.), Springer-Verlag Lecture Notes in Control and Information Sciences vol. 135, 1989.

[GloM:] Glover, K. and D. Mustafa, “Derivation of the Maximum Entropy H_∞-controller and a State-space formula for its Entropy,” *International Journal of Control*, vol. 50, no. 3, pp. 899-916, Sept. 1989.

[GloLDKS:] Glover, K., D.J.N. Limebeer, J.C. Doyle, E.M. Kasenally, and M.G. Safonov, “A characterization of all solutions to the four block general distance problem,” *SIAM J. Control and Opt*., vol. 29, no. 2, pp. 283-324, March, 1991.

[GohLR:] Gohberg, I., P. Lancaster and L. Rodman, “On the Hermitian solutions of the symmetric algebraic Riccati equation,” *SIAM J. Control and Opt*., vol. 24, no. 6, pp. 1323-1334, 1986.

[Gre:] Green, M., “A relative error bound for balanced stochastic truncation,” *IEEE Transactions on Automatic Control*, vol. AC-33, pp. 961-965, 1988.

[GGLD:] Green, M., K. Glover, D. Limebeer, and J.C. Doyle, “A J-spectral factorization approach to H_∞ control,” *SIAM J. Control and Opt*., Vol. 28(6), pp. 1350-1371, 1990.

[HarK:] Hara, S. and P.T. Kabamba, “On computing the induced norm of sampled data feedback systems,” *Proceedings of the American Control Conference*, Green Valley, AZ, pp. 319-320, 1990.

[HauS:] Hautus, M.L.J. and L.M. Silverman, “System structure and singular control.” *Linear Algebra Applications*, vol. 50, pp. 369-402, 1983.

[Hung:] Hung, Y.S., “RH_∞-optimal control-Part I: Model matching, Part II: Solution for controllers,” *International Journal of Control*, vol. 49, pp. 675-684, 1998.

[JonJ:] Jonckheere, E.A., and J.C. Juang, “Fast computation of achievable performance in mixed sensitivity H_∞ design,” *IEEE Transactions on Automatic Control*, vol. AC-32, pp. 896-906, 1987.

[KabH:] Kabamba, P.T. and S. Hara, “Worst case analysis and design of sampled data control systems,” *IEEE Transactions on Automatic Control*, vol. AC-38, no. 9, pp. 1337-1357, Sept. 1993.

[KhaPZ:] Kargoneckar, P.P., I.R. Peterson, and M.A. Rotea, “H_∞ optimal control with state-feedback,” *IEEE Transactions on Automatic Control*, vol. AC-33, pp. 786-788, 1988.

[Kim:] Kimura, H., “Synthesis of H_∞ controllers based on conjugation,” *Proceedings of IEEE Conference on Decision and Control*, Austin, TX., pp. 1207-1211, 1988.

[Kwak:] Kwakernaak, H., “A polynomial approach to minimax frequency domain optimization of mutlivariable feedback systems,” *International Journal of Control*, pp. 117-156, 1986.

[Kuc1:] Kucera V., “A contribution to matrix quadratic equations,” *IEEE Transactions on Automatic Control*, vol. AC-17, no. 3, pp. 344-347, 1972.

[LatA:] Latham, G.A. and B.D.O. Anderson, “Frequency-weighted optimal Hankel norm approximation of state transfer functions,” *Systems and Control Letters*, vol. 5, pp. 229-236, 1985.

[LimH:] Limebeer, D.J.N. and G.D. Halikias, “A controller degree bound for H_∞-optimal control problems of the second kind,” *SIAM J. Control and Opt.*, vol. 26, no. 3, pp. 646-677, 1988.

McFG1:] McFarlane, D.C. and K. Glover, *Robust Controller Design using Normalised Coprime Factor Plant Descriptions*, Springer-Verlag Lecture Notes in Control and Information Sciences, vol. 138, 1989.

[McFG2:] McFarlane, D.C. and K. Glover, “A Loop Shaping Design Procedure using H_∞ Synthesis,” *IEEE Transactions on Automatic Control*, vol. AC-37, no. 6, pp. 759-769, June 1992.

[Moo:] Moore, B.C., “Principal components analysis in linear systems: controllability, observability and model reduction,” *IEEE Transactions on Automatic Control*, vol. AC-26, pp. 17-31, 1981.

[MusG:] Mustafa, D. and K. Glover, “Controllers which satisfy a closed-loop H_∞ norm bound and maximize an entropy integral,” *Proceedings of the 27th IEEE Conference on Decision and Control*, Austin, Texas, 1988, pp. 1609-1613.

[QiuD:] Qiu, L. and E.J. Davison, “Feedback stability under simultaneous gap metric uncertainties in plant and controller,” *Systems & Control Letters*, vol. 18(1), pp. 9-22, 1992.

[Red:] Redheffer, R.M., “Inequalities for a matrix Riccati equation,” *Journal of Mathematics and Mechanics*, vol. 8, no. 3, pp. 349-367, 1959.

[Red2:] Redheffer, R.M., “On a certain linear fractional transformation,” *Journal of Mathematics and Physics*, vol. 39, pp. 269-286, 1960.

[SafCL:] Safonov, M.G., R.Y. Chiang and D.J.N. Limebeer, “Hankel model reduction without balancing: a descriptor approach,” *Proceedings of the 26th IEEE Conference on Decision and Control*, Los Angeles, 1987.

[SafLC:] Safonov, M.G., D.J.N. Limebeer, and R.Y. Chiang, “Simplifying the H_∞ theory via loop shifting,” *Proceedings of the 27th IEEE Conference on Decision and Control*, vol. 2, pp. 1399-1404, 1989.

[Sara:] Sarason, D., “Generalized interpolation in H_∞,” *AMS. Trans.*, vol. 127, pp. 179-203, 1967.

[Silv:] Silverman, L.M., “Inversion of multivariable linear systems,” *IEEE Transactions on Automatic Control*, vol. AC-14, pp. 270-276, 1969.

[Toi:] Toivonen, H.T., “Sampled-data control of continuous-time systems with an H_∞ optimality criterion,” *Report 90-1*, Dept. of Chemical Engineering, Abo Akademi, Finland, Jan., 1990.

[VanD:] Van Dooren, P., “A generalized eigenvalue approach for solving Riccati equations,” *SIAM J. Sci. Comput.*, vol. 2, pp. 121-135, 1981.

[Vin:] Vinnicombe, G., “Measuring Robustness of Feedback Systems,” PhD dissertation, Department of Engineering, University of Cambridge, 1993.

[WanS:] Wang, W. and M.G. Safonov, “A tighter relative error bound for balanced stochastic truncation,” *Systems and Control Letters* , vol. 14, pp. 307-317, 1990.

[Will1:] Willems, J.C., “Least-squares stationary optimal control and the algebraic Riccati equation,” *IEEE Transactions on Automatic Control*, vol. AC-16, pp. 621-634, 1971.

[Will2:] Willems, J.C., “Almost invariant subspaces: an approach to high gain feedback design – Part I: almost controlled invariant subspaces.” *IEEE Transactions on Automatic Control*, vol. AC-26, pp. 235-252, 1981.

[WilKS:] Willems, J.C., A. Kitapci and L.M. Silverman, “Singular optimal control: a geometric approach.” *SIAM J. Control and Opt.*, vol. 24, pp. 323-337, 1986.

[Wonh:] Wonham, W.M., *Linear Multivariable Control: A Geometric Approach*, third edition, Springer-Verlag, New York, 1985.

[YouJB:] Youla, D.C., H.A. Jabr, and J.J. Bongiorno, “Modern Wiener-Hopf design of optimal controllers: part II,” *IEEE Transactions on Automatic Control*, vol. AC-21, pp. 319-338, 1976.

[Yam:] Yamamamoto, Y., “A new approach to sampled-data control systems—A function space viewpoint with applications to tracking problems,” *Proceedings of the 29th Conference on Decision and Control*, pp. 1882-1887, 1990.

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

[ZamE:] Zames, G. and El-Sakkary, “Unstable systems and feedback: The gap metric,” *Proceedings of the Allerton Conference*, pp. 380-385, Oct., 1980.

[ZhoK:] Zhou, K., and P.P. Khargonekar, “An algebraic Riccati equation approach to H_∞ optimization,” *Systems and Control Letters*, vol. 11, pp. 85-92, 1988.

4

Modeling and Analysis of Uncertain Systems

The advanced features of the μ-Analysis and Synthesis Toolbox (μ-Tools) are aimed at

- Analyzing the effect of uncertain models on achievable closed-loop performance
- Designing controllers to provide optimal worst-case performance in the face of the plant uncertainty

Hence, it is imperative that you understand

- How model uncertainty is represented in this framework
- The technical tools available to answer questions about the robustness of a given closed-loop system to certain forms of model uncertainty

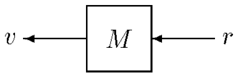
In this chapter, we concentrate on these ideas, through concepts of linear fractional transformations and the structured singular value (μ). We begin in the next section with linear fractional transformations (LFTs) and their role in modeling uncertainty in matrices and systems.

Representing Uncertainty

In this section, we describe the linear fractional representation of uncertainty that is used in μ-Tools. The basic idea in modeling an uncertain system is to separate what is known from what is unknown in a feedback-like connection, and bound the possible values of the unknown elements. This is a direct generalization of the notion of a state-space realization, where a linear dynamical system is written as a feedback interconnection of a constant matrix and a very simple dynamic element made up of a diagonal matrix of delays or integrators. This realization greatly facilitates manipulation and computation of linear systems, and linear fractional transformations provide the same capability for uncertain systems.

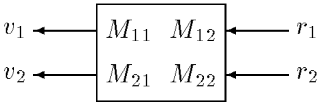
Linear Fractional Transformations (LFTs)

Linear Fractional Transformations (LFTs) are a powerful and flexible approach to represent uncertainty in matrices and systems. Consider first a complex matrix *M*, relating vectors *r* and *v*.



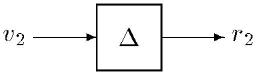
$$v = Mr$$

If *r* and *v* are partitioned into a top part and bottom part, then we can draw the relationship in more detail, explicitly showing the partitioned matrix *M*.



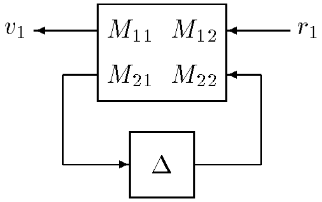
$$\begin{aligned} v_1 &= M_{11}r_1 + M_{12}r_2 \\ v_2 &= M_{21}r_1 + M_{22}r_2 \end{aligned}$$

Suppose a matrix Δ relates v_2 to r_2, as



$$r_2 = \Delta v_2$$

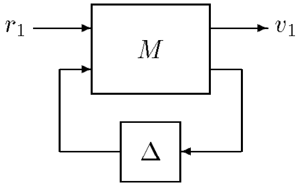
The linear fractional transformation of *M* by Δ interconnects these two elements, as follows,



Eliminate v_2 and r_2, leaving the relationship between r_1 and v_1

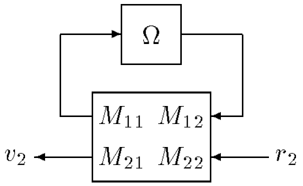
$$\begin{aligned} v_1 &= \underbrace{[M_{11} + M_{12}\Delta(I - M_{22}\Delta)^{-1}M_{21}]}_{F_L(M,\Delta)}r_1 \\ &= F_L(M,\Delta)r_1 \end{aligned}$$

The notation F_L indicates that the *lower* loop of M is closed with Δ. It is more traditional to write a block diagram with the arrows reversed, as in



This still represents the same formula, $v_1 = F_L(M,\Delta)r_1$, and the choice of directions is a matter of taste. We prefer to write as much as is convenient of a block diagram with the arrows going right to left to be consistent with matrix and operator composition, which goes the same way. This simple convention reduces the confusion in going between block diagrams and equations, particularly when blocks have multiple inputs.

If the *upper* loop of M is closed with Ω, then we have



$$v_2 = F_U\left(M,\Omega\right)r_2$$

where

$$F_U(M,\Omega) := [M_{22} + M_{21}\Omega(I - M_{11}\Omega)^{-1}M_{12}]$$

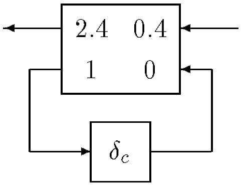
Parametric Uncertainty

How do we use LFTs to represent an uncertain parameter? Suppose c is a parameter, and it is known to take on values

Write this as $c = 2.4{+}0.4\delta_c$ where $\delta_c \in [-1,1]$. This is a linear fractional transformation. Indeed, check that

$$c = F_L\left(\begin{bmatrix} 2.4 & 0.4 \\ 1 & 0 \end{bmatrix}, \delta_c\right)$$

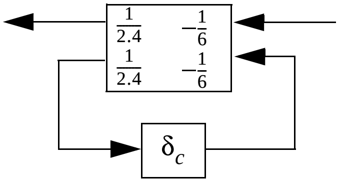
So, everywhere $\leftarrow\boxed{c}\leftarrow$ appears in a block diagram, simply replace it with



If the gain c^{-1} also appears, the LFT representation can still be used, because inverses of LFTs are LFTs (on the same δ). Note that

$$\begin{aligned} c^{-1} &= \frac{1}{2.4 + 0.4\delta_c} \\ &= \frac{1}{2.4}\left(1 + \frac{-\frac{1}{6}\delta_c}{1 - (-\frac{1}{6})\delta_c}\right) \\ &= F_L\left(\begin{bmatrix} \frac{1}{2.4} & -\frac{1}{6} \\ \frac{1}{2.4} & -\frac{1}{6} \end{bmatrix}, \delta_c\right) \end{aligned}$$

So, everywhere $\leftarrow\boxed{\frac{1}{c}}\leftarrow$ appears in a block diagram, replace it with



The general case for inverses can be solved with the matrix inversion lemma. Specifically, given a matrix H, there exists matrices H_{LI} and H_{UI} such that for all Δ and Ω

$$[F_L(\mathrm{H},\Delta)]^{-1} = F_L(H_{LI},\Delta),\ [F_U(H,\Omega)]^{-1} = F_U(H_{UI},\Omega)$$

In fact, with $H = \begin{bmatrix} H_{11} & H_{12} \\ H_{21} & H_{22} \end{bmatrix}$, the formulas for H_{LI} and H_{UI} are just

$$H_{LI} = \begin{bmatrix} H_{11}^{-1} & -H_{11}^{-1}H_{12} \\ H_{21}H_{11}^{-1} & H_{22} - H_{21}H_{11}^{-1}H_{12} \end{bmatrix}$$

$$H_{LI} = \begin{bmatrix} H_{11} - H_{12}H_{22}^{-1}H_{21} & H_{12}H_{22}^{-1} \\ -H_{22}^{-1}H_{21} & H_{22}^{-1} \end{bmatrix}$$

Consider a second order system, representing a single degree-of-freedom mass/damper/spring system with uncertain elements

$$m\ddot{x} + c\dot{x} + kx = u$$

The coefficients are assumed to be uncertain, with a nominal value, and a range of possible variation

$$m = \overline{m}(1 + 0.5\delta_m),\quad c = \bar{c}(1 + 0.3\delta_c),\quad k = \bar{k}(1 + 0.4\delta_k)$$

with $-1 \le \delta_m, \delta_c, \delta_k \le 1$. Note that this represents 50% uncertainty in m, 30% uncertainty in c, 40% uncertainty in k. A block diagram is shown in Figure 4-1.

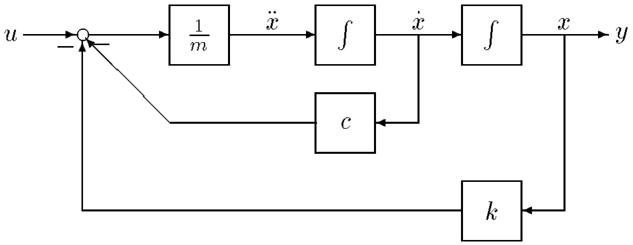


Figure 4-1 Second Order Mass/Damper/Spring System

Define matrices

$$M_{mi} := \begin{bmatrix} -0.5 & \frac{1}{\bar{m}} \\ -0.5 & \frac{1}{\bar{m}} \end{bmatrix}, \qquad M_c := \begin{bmatrix} \bar{c} & 0.3\bar{c} \\ 1 & 0 \end{bmatrix}, \qquad M_k := \begin{bmatrix} \bar{k} & 0.4\bar{k} \\ 1 & 0 \end{bmatrix}$$

Now, replace $\rightarrow\boxed{\frac{1}{m}}\rightarrow$, $\leftarrow\boxed{c}\leftarrow$, and $\leftarrow\boxed{k}\leftarrow$, respectively, with the block diagrams in Figure 4-2.

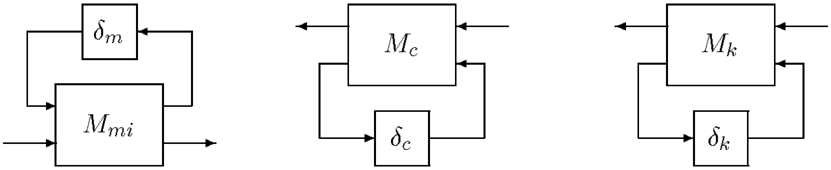


Figure 4-2: Uncertain Elements as LFTs

Since we will eventually separate what is known (M_{mi}, M_c, M_K, and integrators) from what is unknown (δ_m, δ_c, δ_k), redraw the original block diagram with the LFT representation of the uncertain elements, leaving out the δs, but label the signals which go to and from the δs. This is shown in Figure 4-3.

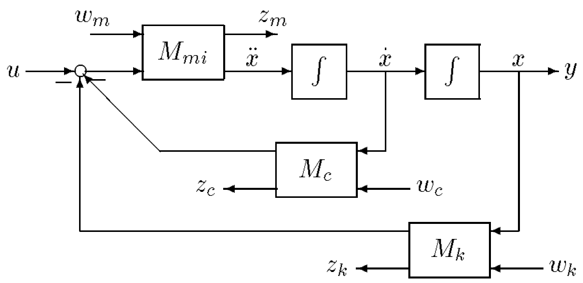


Figure 4-3: Known Part of Uncertain System

Let G_{mck} be the four-input (w_m, w_c, w_k, u), four-output (z_m, z_c, z_k, y), two-state system shown in Figure 4-3 and depicted in Figure 4-4.

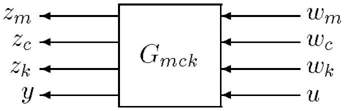


Figure 4-4: Macro View of Known System

Note that G_{mck} only depends on $\overline{m}$, $\bar{c}$, $\bar{k}$, , 0.5, 0.4, and 0.3 and the original differential equation which relates u to y. Hence, G_{mck} is known. Also, the uncertain behavior of the original system is characterized by an upper linear fractional transformation, F_U, of G_{mck} with a diagonal uncertainty matrix as shown in Figure 4-5.

$$y = F_U\left(Gmck, \underbrace{\begin{bmatrix} \delta_m & 0 & 0 \\ 0 & \delta_c & 0 \\ 0 & 0 & \delta_k \end{bmatrix}}_{\Delta}\right) u$$

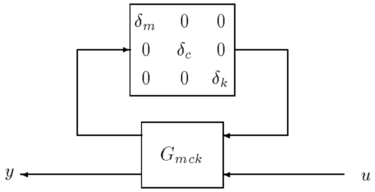


Figure 4-5: Uncertain Mass/Damper/Spring System as LFT

The unknown matrix Δ, referred to as the perturbation, is *structured*. It has a block-diagonal structure, and affects the input/output relationship between u and y in an LFT (feedback) manner. During the course of this chapter, and beyond, many models of uncertainty will involve structured perturbation matrices and LFTs. Using sysic, we can easily compute G_{mck}.

```
mbar = 3; cbar = 1; kbar = 2;
matmi = [-0.5 1/mbar ; -0.5 1/mbar];
matc = [cbar 0.3*cbar;1 0];
matk = [kbar .4*kbar;1 0];
int1 = nd2sys([1],[1 0]);
int2 = nd2sys([1],[1 0]);
systemnames = 'matmi matc matk int1 int2';
sysoutname = 'Gmck';
inputvar = '[wm;wc;wk;u]';
input_to_matmi = '[wm;u-matc(1)-matk(1)]';
input_to_matc = '[int1;wc]';
input_to_matk = '[int2;wk]';
input_to_int1 = '[matmi(1)]';
input_to_int2 = '[int1]';
outputvar = '[matmi(1);matc(2);matk(2);int2]';
sysic;
```

μ-Tools Commands for LFTs

The lower and upper LFT formulas are implemented in μ-Tools with the command `starp`. The name `starp` comes from the *star product* operation defined and developed in [Red$_1$]. The star product is a generalization of the LFT, and includes both the lower and upper LFTs as special cases.

Suppose that T and B are two matrices (CONSTANT, VARYING or SYSTEM) partitioned as below.

$$T = \begin{bmatrix} T_{11} & T_{12} \\ T_{21} & T_{22} \end{bmatrix}, \qquad B = \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

such that the matrix product $T_{22}B_{11}$is well defined, and in fact, square.

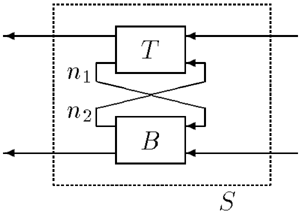
If $I - T_{22}B_{11}$ is invertible, define the star product of T and B to be

$$S(T, B) := \begin{bmatrix} F_L(T, B_{11}) & T_{12}(I - B_{11}T_{22})^{-1}B_{12} \\ B_{21}(I - T_{22}B_{11})^{-1}T_{21} & F_U(B, T_{22}) \end{bmatrix}$$

where F_L and F_U are defined as earlier,

$$\begin{aligned} F_L(T, B_{11}) &= T_{11} + T_{12}B_{11}(I - T_{22}B_{11})^{-1}T_{21} \\ F_U(B, T_{22}) &= B_{22} + B_{21}T_{22}(I - B_{11}T_{22})^{-1}B_{12} \end{aligned}$$

In a block diagram, the star product appears as



and would be computed

```
S = starp(T,B,n1,n2)
```

Here n_1 is the row dimension of $[T_{21}\ T_{22}]$ (and the column dimension of $\begin{bmatrix} B_{11} \\ B_{21} \end{bmatrix}$), which is the number of signals which are fed from T to B. Similarly, n_2 is the row dimension of $[B_{11}\ B_{12}]$ (and the column dimension of $\begin{bmatrix} T_{12} \\ T_{22} \end{bmatrix}$), which is the number of signals which are fed from B to T. The remaining inputs and outputs appear in the output matrix S in the same order as they are in T and B. If the dimension arguments n1 and n2 are omitted, then the following takes place:

- n1 = min([ynum(T) unum(B)])
- n2 = min([unum(T) ynum(B)])

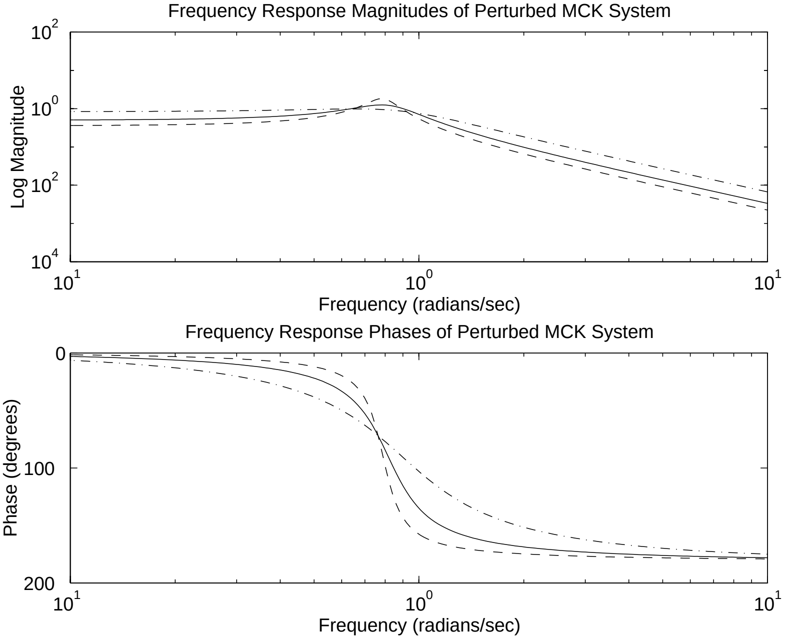
so all possible loop closures are made. Hence, LFTs (which are special cases of star products) are easily computed using starp without dimension arguments.

```
M = crandn(10,8);
Delta = crandn(3,4);
Omega = crandn(6,3);
flmd = starp(M,Delta);
fumo = starp(Omega,M);
```

It is possible to form the star product of two dynamical systems, and it is possible to form the star product of their frequency responses. Hence, starp works with all μ-Tools data types.

We can illustrate this by considering the frequency response of the uncertain mass/damper/spring system considered in the previous section entitled "Parametric Uncertainty". Compute the frequency response of the four-input, four-output system G_{mck}. Then, the frequency response of the uncertain system is simply the linear fractional transformation of the frequency response of G_{mck} with the perturbation matrix (δ_m, δ_c, δ_k).

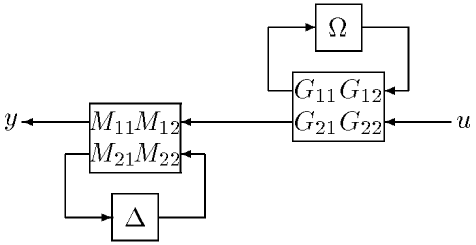
```
Gmck_g = frsp(Gmck,logspace(-1,1,100));
delnom = diag([0;0;0]);
rifd(spoles(starp(delnom,Gmck)))
delnpn = diag([-1;1;-1]);
rifd(spoles(starp(delnpn,Gmck)))
delpnp = diag([1;-1;1]);
rifd(spoles(starp(delpnp,Gmck)))
vplot('bode',starp(delnom,Gmck_g),'-',...
   starp(delnpn,Gmck_g),'.-',starp(delpnp,Gmck_g),'--')
```



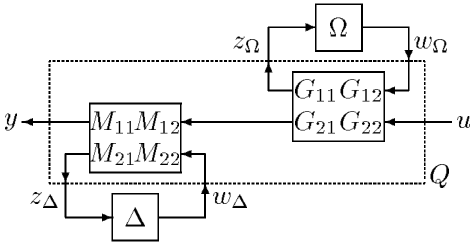
Interconnections of LFTs

By now, you probably have noticed an extremely important property of LFTs — typical algebraic operations such as frequency response, cascade connections, parallel connections, and feedback connections preserve the LFT structure. This means that normal interconnections of LFTs are still in the form of an LFT. Hence, the LFT is an excellent choice for a general hierarchical representation of uncertainty. For illustrative purposes, we consider a few additional examples in this section.

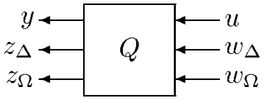
Consider a cascade connection of $F_L(M,\Delta)$ with $F_U(G,\Omega)$, so that $y = F_L(M,\Delta)F_U(G,\Omega)u$. This is shown below.



Draw a box around M and G, isolating them from Δ and Ω, calling the boxed items Q.



Q is made up of the elements of M and G, and relates the variables $(u, w\Delta, w_\Omega)$ to (y, z_Δ, z_Ω)



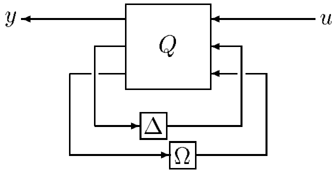
From the diagram, Q is easily calculated as

$$\begin{bmatrix} y \\ z_\Delta \\ z_\Omega \end{bmatrix} = \underbrace{\begin{bmatrix} M_{11}G_{22} & M_{12} & M_{11}G_{21} \\ M_{21}G_{22} & M_{22} & M_{21}G_{21} \\ G_{12} & 0 & G_{11} \end{bmatrix}}_{Q} \begin{bmatrix} u \\ w_\Delta \\ w_\Omega \end{bmatrix}$$

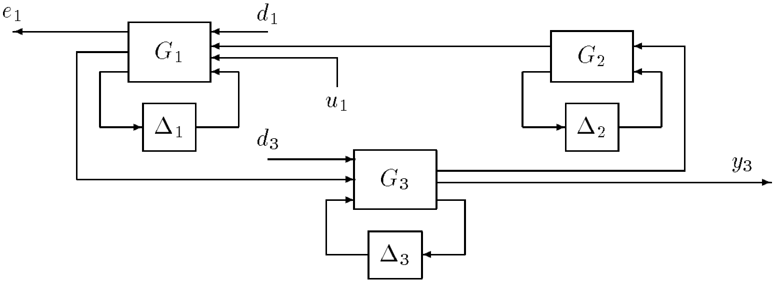
Since $\begin{bmatrix} \Delta & 0 \\ 0 & \Omega \end{bmatrix}$ is the matrix that relates $\begin{bmatrix} z_\Delta \\ z_\Omega \end{bmatrix} \to \begin{bmatrix} w_\Delta \\ w_\Omega \end{bmatrix}$, it is clear that the cascade connection of $F_L(M,\Delta)$ and $F_U(G,\Omega)$ is yet another LFT, namely

$$F_L(M,\Delta)F_U(G,\Omega) = F_L\left(Q, \begin{bmatrix} \Delta & 0 \\ 0 & \Omega \end{bmatrix}\right)$$

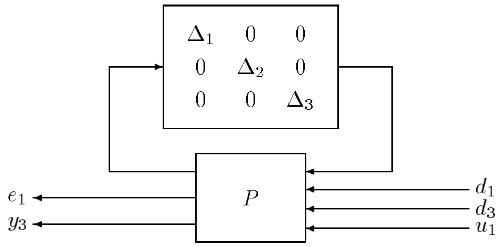
as shown below



Similar manipulations can be carried out for parallel connections, as well as feedback connections, and arbitrary combinations of these. For instance, a complicated feedback connection with three LFTs



can be drawn as a single LFT on the diagonal matrix containing the three individual perturbations,



Here, P depends only on G_1, G_2, G_3, and the interconnection diagram and is easy to calculate with the interconnection program sysic. For instance, if every line in the diagram represents a scalar signal, then correct sysic commands to create P are as follows:

```
G1 = sysrand(8,3,4);
G2 = sysrand(7,2,2);
G3 = sysrand(6,3,3);
systemnames = 'G1 G2 G3';
sysoutname = 'P';
inputvar = '[w1;w2;w3;d1;d3;u]';
input_to_G1 = '[d1;G2(1);u1;w1]';
input_to_G2 = '[G3(1);w2]';
input_to_G3 = '[d3;G1(2);w3]';
outputvar = '[G1(3);G2(2);G3(3);G1(1);G3(2)]';
cleanupsysic = 'yes';
sysic
```

Note The uncertainty matrix affecting P is structured, with a block-diagonal structure. Many elements of the uncertainty matrix are known to be zero. This is an extremely important observation. In other words, general uncertainty at component level becomes structured uncertainty at the interconnection level.

Parameter Uncertainty in Transfer Functions

Suppose that we have a process with an uncertain gain, first-order lag with uncertain time constant, and uncertain delay (modeled with a Pade approximation). The transfer function for the process is

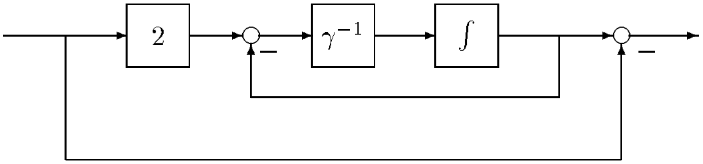
$$y(s) = \underbrace{K}_{\text{gain}} \underbrace{\frac{1}{\tau s+1}}_{\text{lag}} \underbrace{\frac{-\gamma s+1}{\gamma s+1}}_{\text{Pade}} u(s) \tag{4-1}$$

Assume that each of the terms K, γ, and τ is uncertain, with $K \in [1\ 3]$, $\gamma \in [0.05\ 0.15]$, and $\tau \in [1\ 2]$. Further assume that K and γ are linearly related, so that as K takes on values from $1 \rightarrow 3$, γ simultaneously takes on values from $0.05 \rightarrow 0.15$. Represent these variations with two uncertainties, δ_1 and δ_2, with

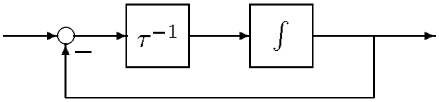
$$K = 2 + \delta_1,\ \gamma = 0.1 + 0.05\delta_1,\ \tau = 1.5 + 0.5\delta_2$$

where $-1 \le \delta_1, \delta_2 \le 1$.

A block diagram of $\frac{-\gamma s+1}{\gamma s+1}$ is



Similarly, a block diagram of $\frac{1}{\tau s+1}$ is

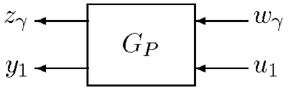


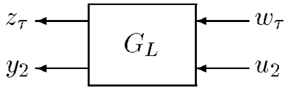
Use upper-loop LFTs to model K, γ^{-1} and τ^{-1}, define matrices

$$M_k := \begin{bmatrix} 0 & 1 \\ 1 & 2 \end{bmatrix},\quad M_{\gamma I} := \begin{bmatrix} -\frac{1}{2} & 10 \\ -\frac{1}{2} & 10 \end{bmatrix},\quad M_{\tau I} := \begin{bmatrix} -\frac{1}{3} & \frac{2}{3} \\ -\frac{1}{3} & \frac{2}{3} \end{bmatrix}$$

so that $K = F_U(M_K,\delta_1)$, $\gamma^{-1} = F_U(M\gamma_I,\delta_1)$, $\tau^{-1} = F_U(M_{\tau I},\delta_2)$.

The first-order Pade system is of the form $F_U(G_P,\delta_1)$, and the first-order lag is of the form $F_U(G_L,\delta_2)$, where G_P and G_L are known, two-input, two-output, one-state systems





G_P is shown in detail in Figure 4-6 and can be built easily using `sysic`.

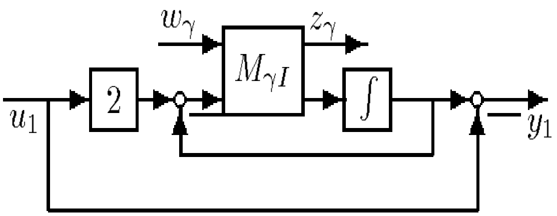


Figure 4-6: LFT System Gp for Uncertain Pade

For instance

```
mgammai = [-0.5 10; -0.5 10];
int = nd2sys([1],[1 0]);
systemnames = 'mgammai int';
sysoutname = 'GP';
inputvar = '[w_gamma;u1]';
input_to_mgammai = '[w_gamma;2*u1-int]';
input_to_int = '[mgammai(2)]';
outputvar = '[mgammai(1);int-u1]';
sysic;
```

Similar calculations are necessary for G_L, whose internal structure is shown in Figure 4-7.

```
mtaui = [-1/3 2/3; -1/3 2/3];
int = nd2sys([1],[1 0]);
systemnames = 'mtaui int';
sysoutname = 'GL';
inputvar = '[w_tau;u2]';
input_to_mtaui = '[w_tau;u2-int]';
input_to_int = '[mtaui(2)]';
outputvar = '[mtaui(1);int]';
sysic;
```

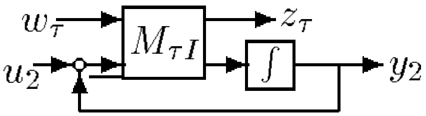


Figure 4-7: LFT System G_L for Uncertain Lag

The uncertain K is directly represented using $F_U(M_K,\delta_1)$. The known part of the uncertain gain/lag/Pade system is simply a four-input, four-output, two-state system G_{proc} shown in Figure 4-8 with internal structure shown in Figure 4-9.

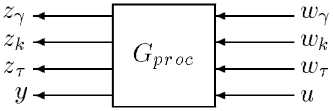


Figure 4-8: Known Part of Uncertain Gain/Lag/Pade

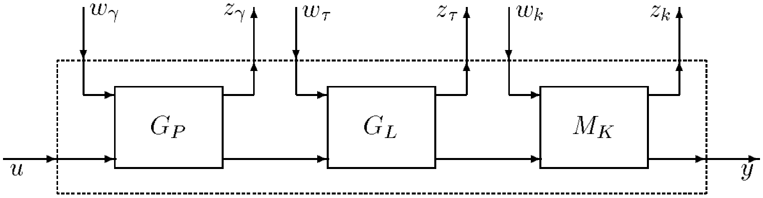


Figure 4-9: Cascade of Pade, Lag, and Gain LFTs

The uncertain system's behavior, equation Figure 4-1, is an LFT,

$$y = F_U\left(G_{proc,}\begin{bmatrix}\delta_1 I_2 & 0\\ 0 & \delta_2\end{bmatrix}\right)u$$

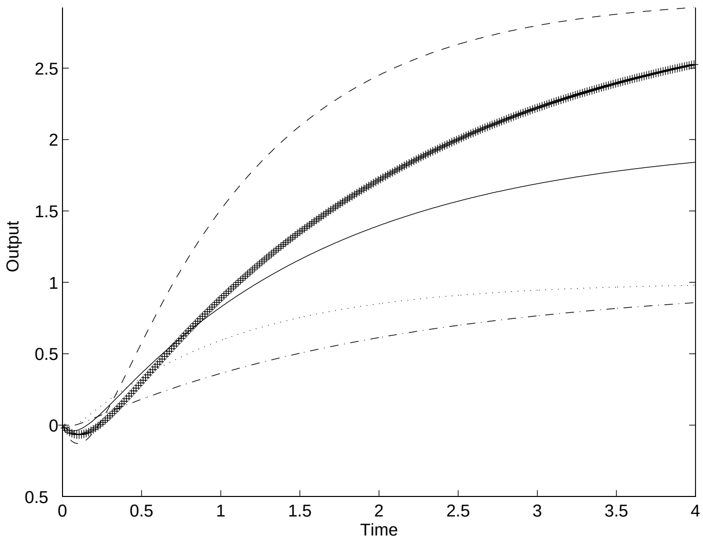
The perturbation δ_1 is repeated twice, due to the coupled variation in K and γ.

The SYSTEM matrix G_{proc} can be constructed with `sysic` as follows:

```
MK = [0 1;1 2];
systemnames = 'GP GL MK';
sysoutname = 'Gproc';
inputvar = '[w_gamma;w_k;w_tau;u]';
input_to_GP = '[w_gamma;u]';
input_to_GL = '[w_tau;GP(2)]';
input_to_MK = '[w_k;GL(2)]';
outputvar = '[GP(1);MK(1);GL(1);MK(2)]';
sysic;
```

We can do some time-domain simulations to verify that this model is correct. Recall that $\delta_1 = -1$ translates into $K = 1$, $\gamma = 0.05$ while $\delta_1 = 1$ means $K = 3$, $\gamma = 0.15$. Also, $\delta_2 = -1$ corresponds to $\tau = 1$, and $\delta_2 = 1$ corresponds to $\tau = 2$.

```
% pertnom ---> nominal values
pertnom = diag([0,0,0]);
% pert1 ---> low gain, short delay, slow lag
pert1 = diag([-1,-1,-1]);
% pert2 ---> low gain, short delay, fast lag
pert2 = diag([-1,-1,1]);
% pert3 ---> high gain, long delay, slow lag
pert3 = diag([1,1,-1]);
% pert4 ---> high gain, long delay, fast lag
pert4 = diag([1,1,1]);
tfinal = 4;
sysnom = starp(pertnom,Gproc);
ynom = trsp(sysnom,1,tfinal);
y1 = trsp(starp(pert1,Gproc),1,tfinal);
y2 = trsp(starp(pert2,Gproc),1,tfinal);
y3 = trsp(starp(pert3,Gproc),1,tfinal);
y4 = trsp(starp(pert4,Gproc),1,tfinal);
vplot(ynom,'-',y1,':',y2,'-.',y3'--',y4,'+')
```



Linear State-Space Uncertainty

For general parametric uncertainty in state-space or transfer function models, the methods outlined in the previous section are used. In the special case of linear uncertainty in a state-space model, the uncertainty description can be built up even more easily. Consider an uncertain state-space model,

$$\begin{bmatrix} \dot{x}(t) \\ y(t) \end{bmatrix} = \begin{bmatrix} A_0 + \sum_{i=1}^{m} \delta_i A_i & B_0 + \sum_{i=1}^{m} \delta_i B_i \\ C_0 + \sum_{i=1}^{m} \delta_i C_i & D_0 + \sum_{i=1}^{m} \delta_i D_i \end{bmatrix} \begin{bmatrix} x(t) \\ u(t) \end{bmatrix}$$

$$= \left(\begin{bmatrix} A_0 & B_0 \\ C_0 & D_0 \end{bmatrix} + \sum_i^m \delta_i \begin{bmatrix} A_i & B_i \\ C_i & D_i \end{bmatrix} \right) \begin{bmatrix} x(t) \\ u(t) \end{bmatrix} \tag{4-2}$$

where for each $i = 1,2,. . .,m$

$$\begin{bmatrix} A_i & B_i \\ C_i & D_i \end{bmatrix} \in \mathbf{R}^{(n+n_u)\times(n+n_y)}$$

Let

$$r_i := \operatorname{rank}\begin{bmatrix} A_i & B_i \\ C_i & D_i \end{bmatrix}$$

and factor each matrix (using `svd`, for instance) as

$$\begin{bmatrix} A_i & B_i \\ C_i & D_i \end{bmatrix} = \begin{bmatrix} E_i \\ F_i \end{bmatrix}\begin{bmatrix} G_i & H_i \end{bmatrix}$$

where

$$\begin{bmatrix} E_i \\ F_i \end{bmatrix} \in \mathbf{R}^{(n+n_y)\times r_i} \quad , \qquad \begin{bmatrix} G_i & H_i \end{bmatrix} \in \mathbf{R}^{r_i\times(n+n_y)}$$

Now, define a linear system G_{ss}, with extra inputs and outputs via the state equations as shown in Figure 4-10.

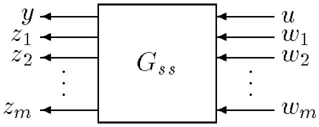


Figure 4-10: Linear System: G_{ss}

The uncertain system in equation Figure 4-2 is represented as an LFT around G_{ss}, namely

$$y = F_L(G_{ss},\Delta)u$$

where Δ maps $z \rightarrow w$, and has the structure given as

$$\begin{bmatrix} \dot{x} \\ u \\ z_1 \\ \vdots \\ z_m \end{bmatrix} = \begin{bmatrix} A_0 & B_0 & E_1 & \dots & E_m \\ C_0 & D_0 & F_1 & \dots & F_m \\ G_1 & H_1 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ G_m & H_m & 0 & \dots & 0 \end{bmatrix} \begin{bmatrix} x \\ u \\ w_1 \\ \vdots \\ w_m \end{bmatrix}$$

$$\Delta = \{\mathrm{diag}[\delta_1 I_{r_1}, \ldots, \delta_m I_{r_m}] : \delta_i \in \mathbf{R}\}$$

This approach, developed in [MorM], has its roots in the *Gilbert realization*, which is discussed in [Kai].

As an example, consider a two-state, single-input, single-output system with a single parameter dependence.

$$\begin{bmatrix} A(\delta) & B(\delta) \\ C(\delta) & D(\delta) \end{bmatrix} := \begin{bmatrix} 0 & 1 & 0 \\ -16 & -0.16 & 1 \\ 16 & 0 & 0 \end{bmatrix} + \delta \begin{bmatrix} 0 & 0 & 0 \\ 6.4 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

The matrix multiplying δ has rank 1, and factors simply as

$$\begin{bmatrix} 0 & 0 & 0 \\ 6.4 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 6.4 \\ 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$$

so the state equations for G_{ss} are

$$\begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \\ u \\ z_1 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -16 & -.16 & 1 & 6.4 \\ 16 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ u \\ w_1 \end{bmatrix}$$

Unmodeled Dynamics

Models of uncertainty are not limited to parametric uncertainty. Often, a low order, nominal model, which suitably describes the low-mid frequency range behavior of the plant is available, but the high-frequency plant behavior is uncertain. In this situation, even the dynamic order of the actual plant is not known, and something richer than parametric uncertainty is needed to represent this uncertainty. One common approach for this type of uncertainty is to use a *multiplicative uncertainty model*. Roughly, this allows you to specify a frequency-dependent percentage uncertainty in the actual plant behavior.

In order to specify the uncertainty set, we need to choose two things:

- A nominal model, $G(s)$
- A multiplicative uncertainty weighting function, $W_u(s)$

Given these, the precise definition of the multiplicative uncertainty set is

$$\mathcal{M}(G, W_u) := \left\{ \tilde{G} : \left| \frac{\tilde{G}(j\omega) - G(j\omega)}{G(j\omega)} \right| \le |W_u(j\omega)| \right\}$$

with the additional restriction that the number of right-half plane (RHP) poles of $\tilde{G}$ be equal to the number of right-half plane poles of G. At each frequency, $|W_u(j\omega)|$ represents the maximum potential percentage difference between all of the plants represented by $\mathcal{M}(G,W_u)$ and the nominal plant model G. In that sense, $\mathcal{M}(G,W_u)$ represents a ball of possible plants, centered at G. On a Nyquist plot, a disk of radius $|W_u(j\omega)G(j\omega)|$, centered at $G(j\omega)$ is the set of possible values that $\tilde{G}(j\omega)$ can take on, due to the uncertainty description.

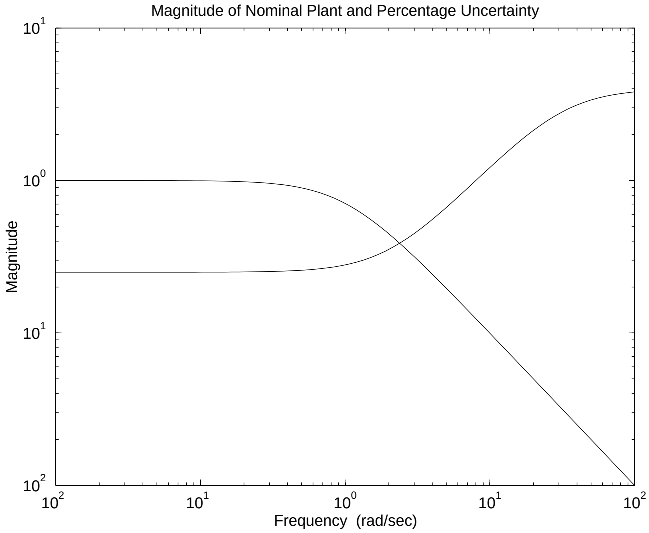
As an example, suppose that the nominal model of the plant is

$$G := \frac{1}{s-1}$$

and the uncertainty in the plant model is parametrized by the multiplicative uncertainty, with uncertainty weight

$$W_u := \frac{\frac{1}{4}\left(\frac{1}{2}s+1\right)}{\frac{1}{32}s+1}$$

```
G = nd2sys([1],[1 -1]);
Wu = nd2sys([0.5 1],[0.03125 1],0.25);
omega = logspace(-2,2,80);
vplot('liv,lm',frsp(G,omega),frsp(wu,omega))
```



It is instructive to consider sets of models that are similar to the nominal model G, and see to what extent the sets are contained in $M(G,W_u)$. Consider the following problem. Determine the smallest $\bar{\beta}$ such that

$$\left\{G\frac{\beta}{s+\beta} : \beta > \bar{\beta}\right\} \subset M(G,\ W_u)$$

The easiest approach using μ-Tools is just to compute, and plot

$$\left|\frac{\tilde{G}(j\omega) - G(j\omega)}{G(j\omega)W_u(j\omega)}\right|$$

for various values of β, and determine the lower limit $\bar{\beta}$ by comparing the plot's magnitude relative to 1. Using the command ex_unc, it is easy to carry out this procedure. For instance,

```
beta = 1;
Gtilde = mmult(nd2sys(beta,[1 beta]),G)
ex_unc(G,Gtilde,wu,omega); % above 1
beta = 10;
Gtilde = mmult(nd2sys(beta,[1 beta]),G)
ex_unc(G,Gtilde,wu,omega); % below 1
```

file: ex_unc.m

```
function ex_unc(G,Gtilde,Wu,omega)
Gg = frsp(G,omega);
Gtildeg = frsp(Gtilde,omega);
Wug = frsp(Wu,omega);
percdiff = vabs(vrdiv(msub(Gtildeg,Gg),mmult(Gg,Wug)));
vplot('liv,m',percdiff,1);
```

A few more trials reveal $\bar{\beta} \approx 6.1$. As an exercise, carry out similar calculations on the following examples:

- Define $\underline{\delta} > 0$ and $\bar{\delta} > 0$ as the smallest and largest numbers such that

$$\left\{\frac{1+\delta}{s-1-\delta} : \underline{\delta} \le \delta \le \bar{\delta}\right\} \subset M(G, W_u)$$

  Exercise: Using μ-Tools, show that $\underline{\delta} \approx 0.33$ and $\bar{\delta} \approx 0.425$.

- Define $\bar{\tau} > 0$ as the largest number such that

$$\left\{G\frac{-\tau s+1}{\tau s+1} : \underline{\tau} \le \bar{\tau}\right\} \subset M(G, W_u)$$

  Exercise: Using μ-Tools, show $\bar{\tau} \approx 0.07$.

- Define $\underline{r} > 0$ and $\bar{r} > 0$ as the smallest and largest numbers such that

$$\left\{G\frac{70^2}{s^2+2\xi 70 s+70^2} : \underline{\xi} \le \xi \le \bar{\xi}\right\} \subset M(G, W_u)$$

  Exercise: Using μ-Tools, show that $\underline{\xi} \approx 0.145$ and $\bar{\xi} \approx 5.7$.

- Define $\overline{m} > 0$ as the largest integer such that

$$\left\{G\left(\frac{50}{s+50}\right)^m : m = 0,1,\ldots,\overline{m}\right\} \subset M(G, W_u)$$

  Exercise: Using μ-Tools, show $\overline{m} = 6$.

We return to this example, and these specific *extreme* plants later in the "Analysis of Controllers for an Unstable System" section in Chapter 7.

Now, by defining $\Delta := \frac{\tilde{G}-G}{GW_u}$, each $\tilde{G}$ can be drawn as in Figure 4-11.

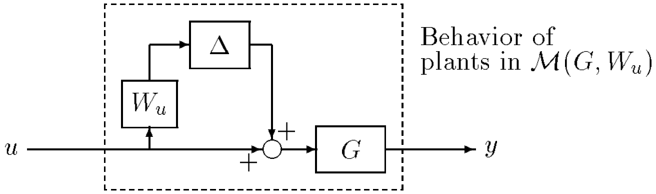


Figure 4-11: Multiplicative Uncertainty

In order to satisfy the constraint $\tilde{G} \in M(G, W_u)$, Δ must be a transfer function that satisfies $\max_{\omega} |\Delta(j\omega)| \leq 1$, and

$$\#\text{RHP poles } [G(1 + W_u\Delta)] = \#\text{RHP poles } [G]$$

Note that Δ itself may be unstable — reconsider the previous example involving the parameter δ. There, the unstable pole of $\tilde{G}$ changes location. It is easy to verify that

$$\Delta := \frac{(\frac{1}{32}s + 1)\delta s}{\frac{1}{4}(\frac{1}{2}s + 1)(s - 1 - \delta)} \tag{4-3}$$

gives (by simple algebraic manipulation)

$$\begin{aligned} G(1 + W_u\Delta) &= \frac{1}{s-1}\left(1 + \frac{\frac{1}{4}(\frac{1}{2}s + 1)}{(\frac{1}{32}s + 1)}\Delta\right) \\ &= \frac{(s-1)(1+\delta)}{(s-1)(s-1-\delta)} \\ &= \frac{1+\delta}{s-1-\delta} \end{aligned}$$

as desired. Moreover, the Δ's that actually yield

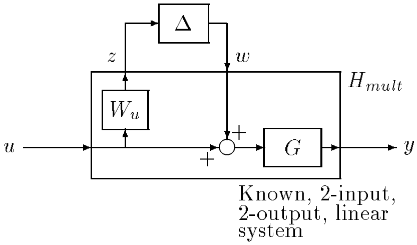
$$G(1 + W_u\Delta) \in M(G, W_u)$$

are those in equation (4-3) which also satisfy

$$\max_{\omega \in \mathbf{R}} |\Delta(j\omega)| \leq 1$$

It is important to understand that in this situation, Δ, G, and W_u are not viewed as three separate physical devices which are interconnected to form $\tilde{G}$. If they were, then the interconnection in Figure 4-11 would actually have two unstable modes, at (1) and $(1 + \delta)$, but the mode at 1 would be uncontrollable from u. Obviously, no controller could internally stabilize the system. Instead, Δ, G, and W_u are just combined to yield a useful manner (through the definition of $\mathcal{M}(G,W_u)$) to parametrize a set of plants that are similar (in a precise way) to G.

As usual, we represent the uncertain plant set $\mathcal{M}(G,W_u)$ as an LFT by separating the known elements (G and W_u) from the unknown elements (Δ) as shown in the following figure.



The uncertain component is now represented as $y = F_U(H_{mult},\Delta)u$, with H_{mult} having the value

$$\begin{bmatrix} z \\ y \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & W_u \\ G & G \end{bmatrix}}_{H_{mult}} \begin{bmatrix} w \\ u \end{bmatrix}$$

H_{mult} is easily calculated using sysic or simpler manipulations.

```
Hmult = madd(mmult([1;0],wu,[0 1]),mmult([0;1],G,[1 1]));
% or
systemnames = 'wu G';
sysoutname = 'Hmult';
inputvar = '[w;u]';
input_to_wu = '[u]';
input_to_G = '[w+u]';
outputvar = '[wu;G]';
sysic
```

In summary, the multiplicative uncertainty model captures a wide variety of plant variations, and represents not only parameter variations, but also unmodeled dynamics. In terms of Nyquist plots, it represents disk-uncertainty at each frequency. It is a coarse, yet simple, approach to putting uncertainty into models.

Additive (as opposed to multiplicative) uncertainty may also be used. Given a nominal model G and an additive uncertainty weighting function W_u, the additive uncertainty set is

$$A(G, W_u) := \{\tilde{G} : |\tilde{G}(j\omega) - G(j\omega)| \leq |W_u(j\omega)|\}$$

with the additional restriction that the number of right-half-plane poles of $\tilde{G}$ be equal to the number of right-half-plane poles of G. The block diagram to represent this form of uncertainty is shown in Figure 4-12. It can be expressed as an LFT in the usual manner.

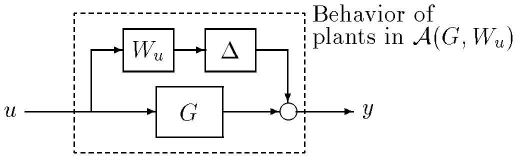
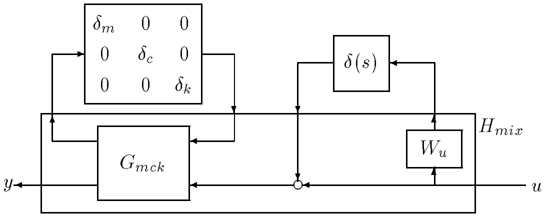


Figure 4-12: Additive Uncertainty

Mixed Uncertainty

Uncertainty may be mixed, including both parametric uncertainty and unmodeled dynamics. For an example, take the uncertain second order system from “Parametric Uncertainty” section, with parametric uncertainty in the parameters m, c, and k and additional high-frequency unmodeled dynamics using the multiplicative uncertainty model. The block diagram is



The possible behavior is represented as an LFT

$$y = F_U(H_{mix}, \Delta)u$$

where the perturbation matrix Δ has the structure

$$\Delta = \text{diag}\{\delta_m,\delta_c,\delta_k,\delta(s)\}$$

and H_{mix} is the five-input, five-output system defined by the following sysic commands.

```
systemnames = 'Gmck wu';
sysoutname = 'Hmix';
inputvar = '[parmpertin{3} ; unmodpertin ; u]';
input_to_Gmck = '[parmpertin; unmodpertin + u]';
input_to_wu = '[u]';
outputvar = '[Gmck(1:3) ; wu ; Gmck(4)]';
sysic
```

Analyzing the Effect of LFT Uncertainty

Given uncertain models, the structured singular value (μ) can be used to *analyze the robustness* of the system to the structured uncertainty that enters in the feedback form.

Using μ to Analyze Robust Stability

At this point, we have learned how to *represent* uncertain systems as LFTs on unknown, structured uncertainty matrices. However, there has been no discussion on how to analyze the robustness of these uncertain systems. This is done with the structured singular value, μ. In this section, we perform an analysis using the command `mu`, which computes upper and lower bounds for μ. We concentrate on the mechanics of using the command, and the conclusions that you reach with a μ-analysis.

Every μ-analysis consists of the following steps:

1. Recast (using the interconnection program `sysic` or Simulink®) the problem into the familiar feedback loop diagram of Figure 4-13, where M is a known linear system, and Δ is a structured perturbation.

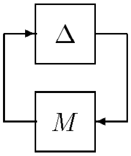


Figure 4-13: General Diagram for Robust Stability Analysis

2. Calculate a frequency response of M.
3. Describe the structure of the perturbations Δ.
4. Run the command `mu` on the frequency response.
5. Plot the bounds obtained from the μ calculation.

Usually M arises by connecting an uncertain system with a feedback controller. Consider the mass/damper/spring system in the "Mixed Uncertainty" section, with both parametric uncertainty and unmodeled dynamics. Suppose that a feedback controller K has been designed to improve the damping, by feedback from y to u, namely u = Ky. Then, the uncertain closed-loop system appears in Figure 4-14.

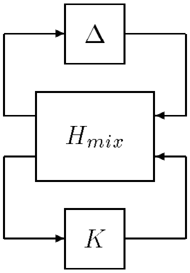


Figure 4-14: Uncertain Closed-Loop System

By grouping H_{mix} and K together, namely, $M := F_L(H_{mix},K)$, Figure 4-15 shows the transformation of Figure 4-14 to Figure 4-13.

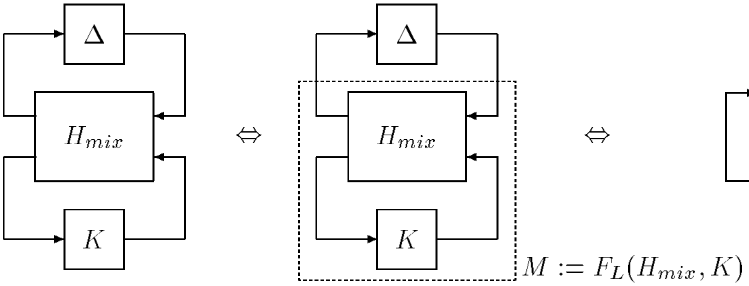


Figure 4-15: Transforming to General Form

With M constructed, next we must clearly describe the structure of the uncertain element Δ. The structure of the perturbation matrices must be passed to the `mu` command. There are three different aspects to consider about each block of the perturbation matrices when specifying the uncertainty structure:

- The type (real parameter vs. unmodeled dynamics) of the perturbation
- The dimension of the perturbation
- The number of independent locations that the particular uncertainty occurs (that is, does it affect the system in two or more places, as δ_1 does in the Gain and Pade parts of the example from the “Parametric Uncertainty” section?)

As for the software, this uncertainty structure information is stored as an $n \times 2$ array (called the block structure array) where n is the number of different perturbation elements in the uncertainty matrix. Each row of this block structure array has information to describe the uncertain block. In μ-Tools, these conventions are:

- A scalar *real* parameter is denoted `[-1 1]` (or `[-1 0]`).
- A repeated (f times) real parameter is denoted `[-f 0]`.
- A 1×1 (ie., scalar) *unmodeled dynamics* perturbation (later referred to as *complex* in the “Structured Singular Value Theory” section), is denoted `[1 1]` or (`[1 0]`).
- An $r \times c$ (ie., full) *unmodeled dynamics* block is denoted `[r c]` See the “HIMAT Robust Performance Design Example” section in Chapter 7 for an example.

Hence, in the“Parametric Uncertainty” section, the uncertainty set for G_{mck} is denoted

```
deltaset = [-1 1;-1 1;-1 1]; % m/c/k
```

The uncertainty set for G_{proc} in the “Parameter Uncertainty in Transfer Functions” section is denoted

```
deltaset = [-2 0;-1 0]; % Gain/Pade/Lag
```

The mixed uncertainty example in the “Mixed Uncertainty” section is represented as

```
deltaset = [-1 1;-1 1;-1 1;1 1]; % m/c/k/unmodeled
```

Note that the ordering of the uncertainty elements is (and must be) consistent with the ordering of the input/output channels of the known systems (G_{mck}, G_{proc}, and H_{mix}).

As a notational convention, once the uncertainty structure has been defined for any given problem, we will use the symbol Δ to represent all perturbation matrices with the appropriate structure. For example, in the mixed uncertainty example of the “Mixed Uncertainty” section, we have

$$\Delta := \left\{ \begin{bmatrix} \delta_1 & 0 & 0 & 0 \\ 0 & \delta_2 & 0 & 0 \\ 0 & 0 & \delta_3 & 0 \\ 0 & 0 & 0 & \delta_4 \end{bmatrix} : \delta_1 \in \mathbf{R}, \delta_2 \in \mathbf{R}, \delta_3 \in \mathbf{R}, \delta_4(s) \right\}$$

Now that the uncertainty structure has been represented, we can compute the size of perturbations to which the system is robustly stable. We need to calculate a frequency response of M, and then compute the structured singular value (μ) of M with respect to the uncertainty set Δ. At each frequency, the matrix $M(j\omega)$ is passed to the μ algorithm, and bounds for $\mu(M(j\omega))$ are computed, giving upper and lower bound functions of frequency, which are plotted. The notation for μ will be $\mu_{\Delta}(M(j\omega))$, to emphasize the dependency of the function not only on M, but also on the uncertainty set Δ.

Suppose the peak (across frequency ω) of the $\mu_{\Delta}(M(j\omega))$ is β. This means that for all perturbation matrices Δ with the appropriate structure (ie., any $\Delta \in \Delta$), and satisfying $\max_{\omega} \bar{\sigma}[\Delta(j\omega)] < \frac{1}{\beta}$, the perturbed system is stable. Moreover, there is a particular perturbation matrix $\Delta \in \Delta$ satisfying $\max_{\omega} \bar{\sigma}[\Delta(j\omega)] < \frac{1}{\beta}$ that causes instability. Hence, we think of

$$\frac{1}{\max_{\omega} \mu_{\Delta}(M(j\omega))}$$

as a stability margin with respect to the structured uncertainty set affecting M.

As mentioned, the software does not compute μ exactly, but bounds it from above and below by several optimization steps. Hence, the conclusions must be carefully stated in terms of upper and lower bounds. Let β_u be the peak (across frequency) of the upper bound for μ, and β_l be the peak of the lower bound for μ. Then:

- For all perturbation matrices $\Delta \in \boldsymbol{\Delta}$ satisfying

  $$\max_{\omega} \bar{\sigma}\,[\Delta(j\omega)] < \frac{1}{\beta_u}$$

  the perturbed system is stable.

- There is a particular perturbation matrix $\Delta \in \boldsymbol{\Delta}$ satisfying

  $$\max_{\omega} \bar{\sigma}\,[\Delta(j\omega)] = \frac{1}{\beta_l}$$

that causes instability.

Hence the gap between the upper and lower bounds translates into gaps between the conclusions *guaranteed robust stability* and *not robustly stable*.

The destabilizing perturbation matrix (of size $\frac{1}{\beta_l}$) can be constructed from the μ calculation using the command `dypert`.

It is instructive to carry out these steps on a simple example. Here, we analyze the robust stability of a simple single-loop feedback regulation system with two uncertainties. The plant is a lightly-damped, nominal two-state system with uncertainty in the (2,1) entry of the A matrix (the frequency-squared coefficient) and unmodeled dynamics (in the form of multiplicative uncertainty) at the control input. The overall block diagram of the uncertain closed-loop system is shown in Figure 4-16. The signals d, n, and e will be used in the next section when robust performance is discussed.

The two-state system with uncertainty in the A matrix is represented as an upper linear fractional transformation about a two-input, two-output, two-state system H, whose realization is

$$\left[\begin{array}{c|c} A_H & B_H \\ \hline C_H & D_H \end{array}\right] := \left[\begin{array}{rr|rr} 0 & 1 & 0 & 0 \\ -16 & -0.16 & 1 & 1 \\ \hline 6.4 & 0 & 0 & 0 \\ 16 & 0 & 0 & 0 \end{array}\right]$$

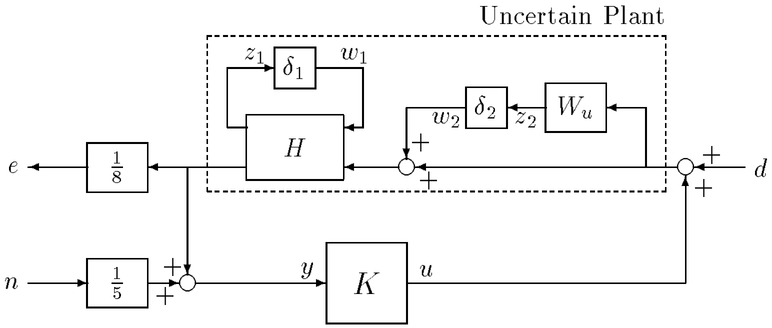


Figure 4-16: Robust Stability/Performance Example

The resulting second order system takes the form

$$F_U(H, \delta_1) = \frac{16}{s^2 + 0.16s + 16(1 + 0.4\delta_1)}$$

If we assume that δ_1 is unknown, but satisfies $|\delta_1| \le 1$, then we interpret the second order system to have 40% uncertainty in the denominator entry of the natural frequency-squared coefficient.

The plant is also assumed to have unmodeled dynamics at the input. This could arise from an unmodeled, or unreliable, actuator, for instance. The uncertainty is assumed to be about 20% at low frequency, rising to 100% at 6 radians/second. We model it using the multiplicative uncertainty model, using a first-order weight

$$W_u = \frac{7s + 8.5}{s + 42}$$

For this example, the controller is chosen as

$$K = \frac{-12.56s^2 + 17.32s + 67.28}{s^3 + 20.37s^2 + 136.74s + 179.46}.$$

Let $M(s)$ in Figure 4-17 denote the closed-loop transfer function matrix from Figure 4-16, after omitting δ_1 and δ_2. The dimensions of M are six states, four inputs and three outputs.

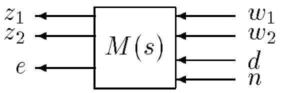


Figure 4-17: Closed-Loop Interconnection

In the *robust stability* analysis, we are only concerned about the stability of the perturbed closed-loop system, and in that case, only the transfer function that the perturbation matrix $\Delta := \begin{bmatrix} \delta_1 & 0 \\ 0 & \delta_2(s) \end{bmatrix}$ sees is important. For notational purposes, drop the s from $M(s)$, and partition M into

$$M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \tag{4-4}$$

where M_{11} is 2×2, and M_{22} is 1×2. The perturbation matrix Δ enters the problem as feedback around M_{11}, as shown in Figure 4-18.

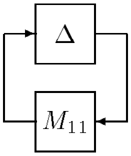


Figure 4-18: Example: Robust Stability

Hence, for robust stability calculations we only need the submatrix M_{11} (for *robust performance* calculations, presented in the “Robust Performance” section, we will need the entire matrix M).

The computational steps for analysis are given below:

1 Construct M and check dimensions.

```
rsexamp; % creates data
minfo(M)
```

2 Calculate frequency response, and select the first two input and output channels for the robust stability test.

```
omega = logspace(-2,2,200);
M_g = frsp(M,omega);
M11_g = sel(M_g,1:2,1:2);
```

3 Create matrix to describe the uncertainty structure (1 real parameter, 1 unmodeled dynamics uncertainty).

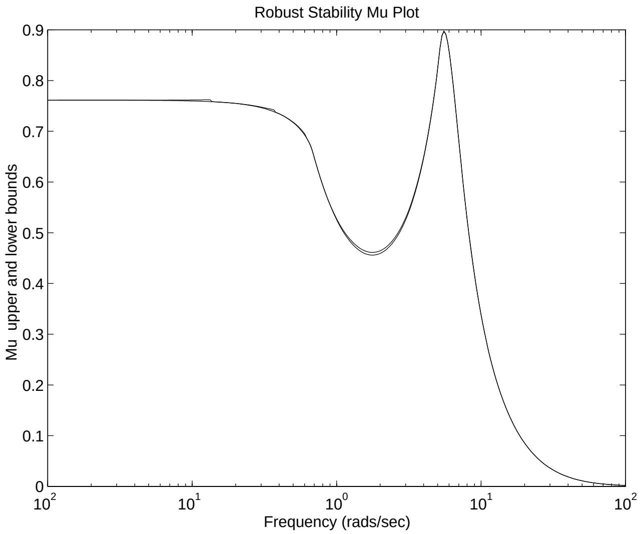
```
deltaset = [-1 0;1 1];
```

4 Compute $\mu_\Delta(M_{11}(j\omega))$ using `mu`.

```
[mubnds,dvec,sens,pvec,gvec] = mu(M11_g,deltaset);
```

5 Plot the bounds.

```
vplot('liv,m',mubnds)
pkvnorm(sel(mubnds,1,1))
[pklow,omegapklow] = pkvnorm(sel(mubnds,1,2))
```



The peak is about 0.89 (upper and lower bounds are very close in this example) and occurs at $\omega = 5.4$ radians/second. Hence, stability is guaranteed for all perturbations with appropriate structure, and $\max_{\omega} \bar{\sigma}\,[\Delta(j\omega)] < \frac{1}{0.89} \approx 1.12$.

6 Construct the smallest destabilizing perturbation. Check that the perturbation has the appropriate structure and that the size of the perturbation is equal to the reciprocal of the peak of the lower bound.

```
pert=dypert(pvec,deltaset,mubnds);
minfo(pert)
seesys(pert)
spoles(pert)
hinfnorm(sel(pert,1,1))
hinfnorm(sel(pert,2,1))
hinfnorm(sel(pert,1,2))
hinfnorm(sel(pert,2,2))
hinfnorm(pert)
```

7 Verify that the perturbed system is indeed unstable. Note the location of the perturbed closed-right-half-plane pole and its relationship with the peak of the μ-lower bound plot.

```
pertM = starp(pert,M);
spoles(pertM)
```

Using μ to Analyze Robust Performance

We have seen how μ can be used to analyze the robust stability of a LFT. In this section, we use μ to determine the robust performance of a LFT under perturbation.

Assume that the problem has been manipulated (using the program `sysic` or Simulink) into the familiar feedback loop diagram of Figure 4-19, where M is a known linear system, Δ is a structured perturbation from a problem-dependent allowable uncertainty set Δ, and d and e are the generalized disturbance and error that characterizes the performance objective.

The transformation from a system block diagram to the generic diagram in Figure 4-19 is analogous to the transformation described in this section and Figure 4-15, except that exogenous signals (disturbances and errors) are simply kept in the diagram.

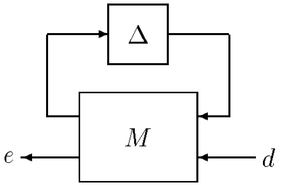


Figure 4-19: Robust Performance Formulation

Recall from Chapter 3 that the performance of MIMO control systems is characterized using H_∞ norms. Specifically, we assume that *good performance* is equivalent to

$$\|T\|_\infty := \max_{\omega \in \mathbf{R}} \bar{\sigma}(T(j\omega)) \le 1$$

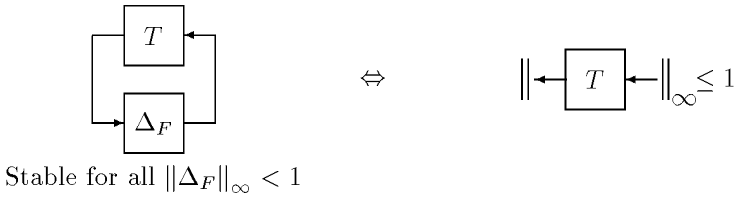
where T is some weighted, closed-loop transfer function matrix. In the case of robust performance of uncertain systems, we'll take T to be the uncertain transfer function from $d \to e$, so $T = F_U(M,\Delta)$.

Clearly, the transfer function from $d \to e$ is a function of Δ, through the elements of M, and the linear fractional formula for F_U. How large can the transfer function $F_U(M,\Delta)$ get as Δ takes on its allowed values? In view of this question, the robust performance condition is defined as The LFT in Figure

4-19 is said to achieve *robust performance* if it is stable for all perturbations $\Delta \in \Delta$ satisfying $\max_{\omega} \bar{\sigma}\,[\Delta(j\omega)] < 1$, and moreover, if $\|F_U(M,\Delta)\|_\infty \le 1$ for all such perturbations.

How can μ be used to assess robust performance? The main idea is very simple — you must first relate the size of a transfer function to a robust stability test. Suppose that T is a given stable system with input dimension n_d and output dimension n_e. By the Nyquist and small-gain theorem, we know that $\|T\|_\infty \le 1$ if and only if the feedback loop in Figure 4-20 is stable for every stable $\Delta_F(s)$ (of dimension $n_d \times n_e$) satisfying $\|\Delta_F\|_\infty < 1$.

Hence, a transfer function T is small ($\|T\|_\infty \le \beta$) if and only if T can tolerate all possible stable feedback perturbations Δ_F (with $\|\Delta_F\|_\infty < \frac{1}{\beta}$) without leading to instability. In other words, the size of a transfer function could be determined using a robust stability test. This ultimately allows us to pose the robust performance question as a robust stability question.



T
Δ_F
Stable for all $\|\Delta_F\|_\infty < 1$
$\Leftrightarrow$
$\|$ T $\|_\infty \le 1$

Figure 4-20: Performance as Robustness

Consider our situation, where $T = F_U(M,\Delta)$. Following the argument presented, we have that $\|F_U(M,\Delta)\|_\infty \le 1$ for all perturbations $\Delta \in \Delta$ satisfying $\max_{\omega} \bar{\sigma}\,[\Delta(j\omega)] < 1$ if and only if the LFT shown in Figure 4-21 is stable for all $\Delta \in \Delta$ and all stable Δ_F satisfying

$$\max_{\omega} \bar{\sigma}\,(\Delta(j\omega)) < 1 \quad \text{and} \quad \max_{\omega} \bar{\sigma}\,(\Delta_F(j\omega)) < 1.$$

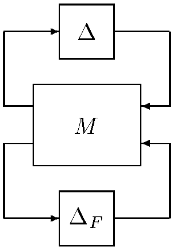


Figure 4-21: Robust Stability with Augmented Uncertainty

But this is exactly a *robust stability* problem for *M*, subjected to perturbation matrices of the form

$$\Delta_P = \begin{bmatrix} \Delta & 0 \\ \Delta & 0_F \end{bmatrix}$$

Hence, we use *robust stability* techniques — on a larger problem, computing $\mu_{\Delta_P}(M(j\omega))$ — to determine bounds on robust performance for our original problem. Specifically, we use an additional (fictitious) uncertainty element, and determine the robust stability of the extended system, and finally make conclusions about the robust performance of the original uncertain system, $F_U(M,\Delta)$.

Important Robust $\|\cdot\|_\infty$ Performance is characterized by introducing a fictitious uncertainty block across the disturbance/error channels and carrying out a Robust Stability Analysis.

In summary, each robust performance μ-analysis consists of the following steps:

1 Recast (using the interconnection program `sysic` or Simulink) the problem into the familiar feedback loop diagram of Figure 4-19, where *M* is a known linear system, and Δ is a structured perturbation, and *d* and *e* are the generalized disturbance and error that characterizes the performance objective.

2. Calculate a frequency response of M.
3. Describe the structure of the perturbation set $\mathbf{\Delta}$.
4. Use the dimensions of the disturbance/error channels to define the fictitiously uncertainty block, and augment this with the actual uncertainty structure of $\mathbf{\Delta}$ to obtain an extended uncertainty set $\mathbf{\Delta}_P$.
5. Compute $\mu_{\Delta_P}(M(j,\omega))$ on the frequency response of M, using the augmented uncertainty set $\mathbf{\Delta}_P$ (which contains the fictitious full, transfer function block).
6. Plot the bounds obtained from the μ calculation.

Suppose the peak of the μ-plot is β. This means that for all perturbation matrices $\Delta \in \mathbf{\Delta}$ satisfying $\max_{\omega} \bar{\sigma}[\Delta(j\omega)] < \frac{1}{\beta}$, the perturbed system is stable and $\|F_U(M,\Delta)\|_\infty \leq \beta$. Moreover, there is a particular perturbation $\Delta \in \mathbf{\Delta}$ satisfying $\max_{\omega} \bar{\sigma}[\Delta(j\omega)] = \frac{1}{\beta}$ that causes either $\|F_U(M,\Delta)\|_\infty = \beta$, or instability.

However, the software does not compute μ exactly, but upper and lower bounds. Let β_u be the peak (across frequency) of the upper bound for μ, and β_l be the peak (across frequency) of the lower bound for μ. Then:

- For all perturbation matrices $\Delta \in \mathbf{\Delta}$ satisfying

$$\max_{\omega} \bar{\sigma}[\Delta(j\omega)] < \frac{1}{\beta_u} ,$$

  the perturbed system is stable and $\|F_U(M,\Delta)\|_\infty \leq \beta_u$.
- There *is* a particular perturbation matrix $\Delta \in \mathbf{\Delta}$ satisfying

$$\max_{\omega} \bar{\sigma}[\Delta(j\omega)] = \frac{1}{\beta_l}$$

that causes either $\|F_U(M,\Delta)\|_\infty \geq \beta_l$, or instability.

Hence the gap between the upper and lower bounds leads to gaps in your inability to precisely determine the robust performance.

There are two exogenous disturbances: a force disturbance *d* at the input to the plant, and sensor noise *n* at the measurement. There is a single error, the output of the plant. The (weighted) open-loop transfer functions satisfy 2.6 ≤ $T_{ed} \leq 4.0$ depending on the value of $1 \geq \delta_1 \geq -1$, and of course $T_{en} = 0$. Hence the objective of control is to satisfy

$$\max_{\omega}[|T_{ed}(j\omega)|^2 + |T_{en}(j\omega)|^2]^{\frac{1}{2}} \leq 1$$

for all allowable perturbations.

In terms of M, we have

$$T = F_U\left(M, \begin{bmatrix} \delta_1 & 0 \\ 0 & \delta_{2(s)} \end{bmatrix}\right)$$

T is a 1-output, 2-input system, so the fictitious perturbation block, Δ_F, must be 2 × 1. Hence, the augmented uncertainty structure has three blocks, and they are:

- A scalar real parameter
- A scalar, transfer function uncertainty block
- A 2 × 1 full transfer function uncertainty block

```
deltaset = [-1 0;1 1];
fict_blk = [2 1];
aug_deltaset = [deltaset ; fict_blk];
```

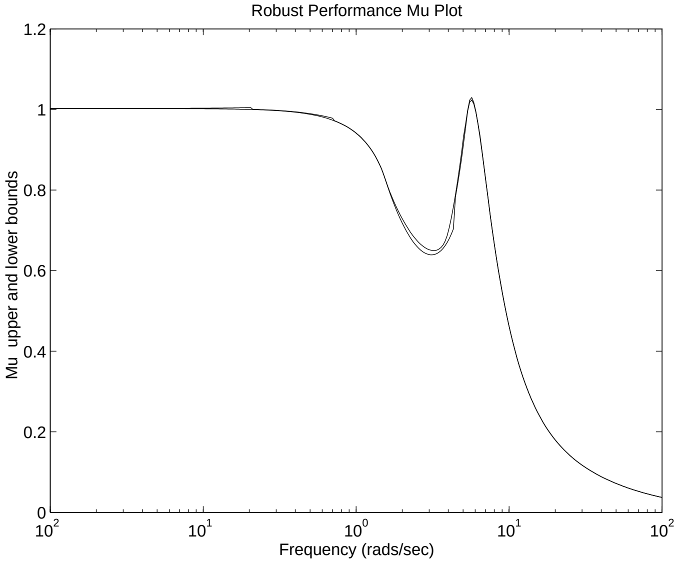
Now, the nominal value of T is simply $F_U(M,0_{2\times2})$, which is just M_{22}. Plot the magnitude (versus frequency) of these elements to see the nominal closed-loop transfer function. In the next command, we extract M_{22} in two different but equivalent manners.

```
vplot('liv,lm',sel(M_g,3,3:4),starp(zeros(2,2),M_g))
```

The robust performance μ calculation gives information about how much these transfer functions are affected by the linear fractional perturbations $\delta_1$and δ_2.

The calculation is carried out on the entire matrix *M*, using the augmented uncertainty structure.

```
[rpbnds,rpdvec,rpsens,rppvec,rpgvec] = mu(M_g,aug_deltaset);
vplot('liv,m',rpbnds);
```



The peak value (of both lower and upper bounds) is about 1.02. This implies that robust performance is not quite achieved. In other words, for every perturbation $\Delta = \mathrm{diag}\,[\delta_1,\delta_2(s)]$ satisfying $\max_{\omega} \bar{\sigma}\,[\Delta(j\omega)] < \frac{1}{1.02}$, we are guaranteed stability and $F_U(M,\Delta) \le 1.02$. Moreover, there is a perturbation $\Delta = \mathrm{diag}\,[\delta_1,\delta_2(s)]$ with $\max_{\omega} \bar{\sigma}\,[\Delta(j\omega)] \approx \frac{1}{1.02} < 1$ such that

$$\|F_U(M,\Delta)\|_\infty \approx 1.02 > 1.$$

This can be constructed with dypert, and put into the closed-loop system to verify the degradation of performance.

```
pert = dypert(rppvec,aug_deltaset,rpbnds,[1;2]);
spoles(pert)
seesys(pert)
hinfnorm(pert)
hinfnorm(starp(pert,M))
vplot('liv,lm',vnorm(starp(pert,M_g)))
vplot('liv,lm',starp(pert,M_g))
```

Using μ to Analyze Worst-Case Performance

In addition to determining if a system has robust performance to uncertainty, it is useful to get the worst-case perturbation of a given size. For instance, using perturbations of a particular structure Δ, and restricting to those of size $\leq \alpha$, what is the worst performance possible (as measured in $\|\cdot\|_\infty$ norm) and what is the perturbation that causes the largest degradation of performance? Precisely given $\alpha > 0$, the worst-case performance associated with structured perturbations of size α is defined as

$$W(M, \Delta, \alpha) := \max_{\substack{\Delta \in \Delta \\ \max_{\omega} \bar{\sigma}[\Delta(j\omega)] \leq \alpha}} \|F_U(M, \Delta)\|_\infty$$

The perturbation matrix which achieves the maximum is denoted $\Delta_{bad,\alpha}$.

The command wcperf assumes that the performance transfer function is an upper loop LFT

$F_U(M,\Delta)$

so that the uncertainty structure represented by deltaset is closed around the upper loops of M. The corresponding worst-case perturbation can be used in high-fidelity time-domain simulations to further understand the effect of uncertainty. Plotting $W(M,\alpha)$ versus α yields the *worst-case performance tradeoff curve*, which shows the tradeoff between size of uncertainty and worst-case performance. This can be used to assess the relative merits of two different controllers. The command wcperf computes the performance degradation curves. The syntax to compute the worst-case perturbation of size

0.8, and compute the performance degradation curves with at least 10 points is given below. The degradation curves are shown in Figure 4-22.

```
alpha = 0.8;
npts = 10;
[deltabad,lowbnd,uppbnd] = wcperf(M_g,deltaset,alpha,npts);
seesys(deltabad)
hinfnorm(deltabad)
hinfnorm(starp(deltabad,M))
vplot(lowbnd,uppbnd)
grid
```

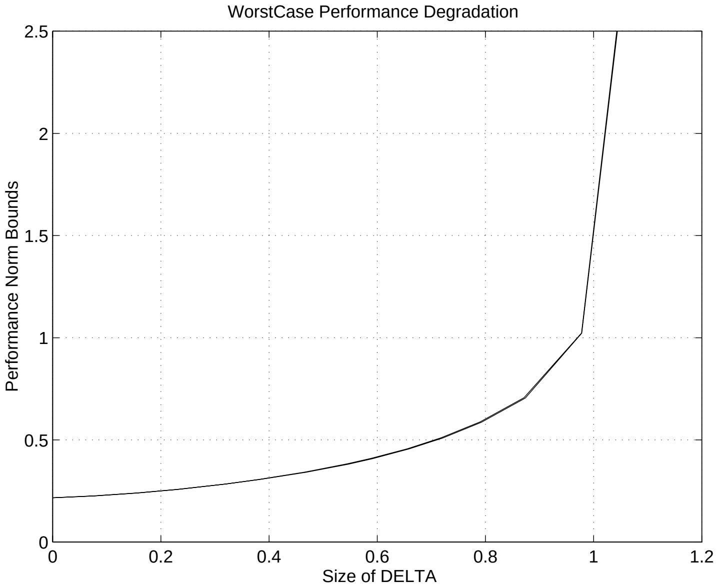


Figure 4-22: Worst-Case Performance Degradation Curves

Because μ can only be bounded above and below, the worst-case performance degradation can also only be bounded. The VARYING matrices `lowbnd` and `uppbnd` bound the worst-case performance degradation.

Summary

At this point, you should be well suited to explore the examples involving μ-analysis, namely the “Analysis of Controllers for an Unstable System”, “MIMO Margins Using m” and the “Space Shuttle Robustness Analysis” sections in Chapter 7. The rest of this chapter is a theoretical development of the properties of μ, and is not necessary reading at this point. Later, as you work with the μ software, and get comfortable with the interpretation of μ as a robust stability and robust performance measure, we encourage you to complete reading this chapter and learn more about this powerful framework.

Structured Singular Value Theory

This section covers a brief description of the mathematical properties of the structured singular value, μ. The material is mathematical, relying mostly on linear algebra concepts. Unfortunately, although many of the concepts are simple, the notation required to state things precisely gets messy and cumbersome. The notation used in this chapter is standard. It is listed in the "Notation" section of Chapter 3.

Complex Structured Singular Value

Definitions

This section is devoted to defining the structured singular value, a matrix function denoted by $\mu(\cdot)$. We consider matrices $M \in \mathbf{C}^{n \times n}$. In the definition of $\mu(M)$, there is an underlying structure Δ, (a prescribed set of block diagonal matrices) on which everything in the sequel depends. For each problem, this structure is in general different; it depends on the uncertainty and performance objectives of the problem. Defining the structure involves specifying three things; the type of each block, the total number of blocks, and their dimensions.

There are two types of blocks: *repeated scalar* and *full* blocks. Two nonnegative integers, S and F, represent the number of *repeated scalar* blocks and the number of *full* blocks, respectively. To keep tracking of their dimensions, we introduce positive integers $r_1,\ldots,r_S$; $m_1,\ldots,m_F$. The ith repeated scalar block is $r_i \times r_i$, while the jth full block is $m_j \times m_j$. With those integers given, we define $\Delta \subset \mathbf{C}^{n \times n}$ as

$$\Delta = \left\{ \operatorname{diag}\,[\delta_1 I_{r_1},\ldots,\delta_S I_{r_S},\Delta_1,\ldots,\Delta_F] : \delta_i \in \mathbf{C}, \Delta_j \in \mathbf{C}^{m_j \times m_j} \right\} \tag{4-5}$$

For consistency among all the dimensions, we must have

$$\sum_{i=1}^{S} r_i + \sum_{j=1}^{F} m_j = n.$$

Often, we will need norm bounded subsets of Δ, and we introduce the following notation

$$B_\Delta = \{\Delta \in \Delta : \bar{\sigma}(\Delta) \le 1\} \tag{4-6}$$

Note that in Figure 4-5 all of the repeated scalar blocks appear first. This is just to keep the notation as simple as possible, in fact they can come in any order. Also, the full blocks do not have to be square, but restricting them as such saves a great deal in terms of notation. The software will handle nonsquare full blocks, as well as any order to the blocks.

Definition 4.1: For $M \in \mathbf{C}^{n \times n}$, $\mu_\Delta(M)$ is defined

$$\mu_\Delta(M) := \frac{1}{\min\{\bar{\sigma}(\Delta) : \Delta \in \mathbf{\Delta}, \det(I - M\Delta) = 0\}} \tag{4-7}$$

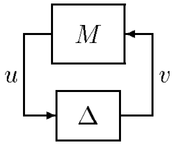
unless no $\Delta \in \mathbf{\Delta}$ makes $I - M\Delta$ singular, in which case $\mu_\Delta(M) := 0$.

Remark 1: Without loss in generality, the full blocks in the minimal norm Δ can each be chosen to be dyads (rank = 1). To see this, first consider the case of only 1 full block, $\mathbf{\Delta} = \mathbf{C}^{n \times n}$. Suppose that $I - M\Delta$ is singular. Then for some unit-norm vector x $\in \mathbf{C}^n M\Delta x = x$. Define $y := \Delta_x$. It follows that $y \neq 0$, and $\|y\| \leq \bar{\sigma}(\Delta)$. Hence, define a new perturbation, $\tilde{\Delta} \in \mathbf{C}^{n \times n}$ as

$$\tilde{\Delta} := yx^*$$

Obviously, $\bar{\sigma}(\tilde{\Delta}) = \|y\| \leq \bar{\sigma}(\Delta)$, and $y = \tilde{\Delta}x$, so that $I - M\tilde{\Delta}$ is also singular. Hence we have replaced a general perturbation Δ which satisfies the singularity condition, with a rank 1 perturbation that is no larger (in the $\bar{\sigma}(\cdot)$ sense), but still satisfies the singularity condition. Repeating this on a block-by-block basis allows for each full block to be a dyad. The software always carries out this particular construction.

Remark 2: It is instructive to consider a *feedback* interpretation of $\mu_\Delta(M)$at this point. Let $M \in \mathbf{C}^{n \times n}$ be given, and consider the loop shown below.



This picture is meant to represent the loop equations

$$u = Mv$$
$$v = \Delta u. \tag{4-8}$$

As long as $I - M\Delta$ is nonsingular, the only solutions u, v to the loop equations are $u = v = 0$. However, if $I - M\Delta$ is singular, then there are infinitely many solutions to Figure 4-8, and the norms $\|u\|$, $\|v\|$ of the solutions can be arbitrarily large. With an abuse of convention, we call this constant matrix feedback system unstable. Likewise, the term stable will describe the situation when the only solutions are identically zero. In this context, then, $\mu_\Delta(M)$ is a measure of the smallest structured Δ that causes *instability* of the constant matrix feedback loop shown above.

An alternative expression for $\mu_\Delta(M)$ follows from the definition.

Lemma 4.2:

$$\mu_\Delta(M) = \max_{\Delta \in \mathbf{B}_\Delta} \rho(M\Delta)$$

In view of this lemma, continuity of the function $\mu : \mathbf{C}^{n \times n} \rightarrow$ R is apparent. In general, though, the function $\mu : \mathbf{C}^{n \times n} \rightarrow$ R is not a norm, since it doesn't satisfy the triangle inequality. However, for any $\alpha \in \mathbf{C}$, $\mu(\alpha M) = |\alpha| \mu(M)$, so in some sense, it is related to how *big* the matrix is in a norm sense.

We can relate $\mu_\Delta(M)$ to familiar linear algebra quantities when Δ is one of two extreme sets:

- If $\Delta = \{\delta I : \delta \in \mathbf{C}\}$ ($S = 1$, $F = 0$, $r_1 = n$), then $\mu_\Delta(M) = \rho(M)$, the spectral radius of M.

  Proof: The only Δ's in Δ which satisfy the $\det(I - M\Delta) = 0$ constraint are reciprocals of nonzero eigenvalues of M. The smallest one of these is associated with the largest (magnitude) eigenvalue, so, $\mu_\Delta(M) = \rho(M)$.

- If $\Delta = \mathbf{C}^{n \times n}$ ($S = 0$, $F = 1$, $m_1 = n$), then $\mu_\Delta M = \bar{\sigma}(M)$.

Proof: If $\bar{\sigma}(\Delta) < \frac{1}{\bar{\sigma}(M)}$, then $\bar{\sigma}(M\Delta) < 1$, so $I - M\Delta$ is nonsingular. Applying equation Figure 4-7 implies $\mu_\Delta(M) \le \bar{\sigma}(M)$. On the other hand, let u and v be unit vectors satisfying $Mv = \bar{\sigma}(M)u$, and define $\Delta := \frac{1}{\bar{\sigma}(M)}vu^*$. Then $\bar{\sigma}(\Delta) = \frac{1}{\bar{\sigma}(M)}$ and $I - M\Delta$ is obviously singular. Hence, $\mu_\Delta(M) \geq \bar{\sigma}(M)$.

For a general Δ as in Figure 4-5 we must have

$$\{\delta I_n : \delta \in \mathbf{C}\} \subset \Delta \subset \mathbf{C}^{n \times n} \tag{4-9}$$

From the definition of μ, and the two special cases above, we conclude that

$$\rho(M) \leq \mu_\Delta(M) \leq \bar{\sigma}(M) \tag{4-10}$$

These bounds alone are not sufficient for our purposes, because the gap between ρ and $\bar{\sigma}$ can be arbitrarily large. They are refined by considering transformations on M that do not affect $\mu_\Delta(M)$, but affect ρ and $\bar{\sigma}$. To do this, define the following two subsets of $\mathbf{C}^{n\times n}$.

$$\mathbf{Q}_\Delta = \{Q \in \Delta : Q^*Q = I_n\} \tag{4-11}$$

$$\mathbf{D}_\Delta = \left\{ \begin{array}{l} \mathrm{diag}[D_1,\ldots,D_S,d_1 I_{m_1},\ldots,d_{F-1}I_{m_{F-1}},I_{m_F}] : \\ D_i \in \mathbf{C}^{r_i \times r_i}, D_i = D_i^* > 0, d_j \in \mathbf{R}, d_j > 0 \end{array} \right\}. \tag{4-12}$$

Note that for any $\Delta \in \Delta$, $Q \in \mathbf{Q}_\Delta$, and D $\in \mathbf{D}_\Delta$,

$$Q^* \in Q_\Delta \quad Q\Delta \in \Delta \quad \Delta Q \in \Delta \quad \bar{\sigma}(Q\Delta) = \bar{\sigma}(\Delta Q) = \bar{\sigma}(\Delta) \tag{4-13}$$

$$D\Delta = \Delta D. \tag{4-14}$$

Consequently,

Theorem 4.3: For all $Q \in \mathbf{Q}_\Delta$ and D $\in \mathbf{D}_\Delta$

$$\mu_\Delta(MQ) = \mu_\Delta(QM) = \mu_\Delta(M) = \mu_\Delta(DMD^{-1}) \tag{4-15}$$

Proof: For all D $\in \mathbf{D}_\Delta$ and $\Delta \in \Delta$,

$$\det(I - M\Delta) = \det(I - MD^{-1}\Delta D) = \det(I - DMD^{-1}\Delta)$$

since D commutes with Δ. Therefore, $\mu_\Delta(M) = \mu_\Delta(DMD^{-1})$. Also, for each $Q \in \mathbf{Q}_\Delta$, $\det(I - M\Delta) = 0 \Leftrightarrow \det(I + \mathrm{M}QQ^*\Delta) = 0$. Since $Q^*\Delta \in \Delta$ and $\bar{\sigma}(Q^*\Delta) = \bar{\sigma}(\Delta)$,we get $\mu_\Delta(MQ) = \mu_\Delta(M)$ as desired. The argument for QM is the same.

Therefore, the bounds in Figure 4-10 can be tightened to

$$\max_{Q \in \mathbf{Q}_\Delta} \rho(QM) \le \max_{\Delta \in \mathbf{B}_\Delta} \rho(\Delta M) = \mu_\Delta(M) \le \inf_{D \in \mathbf{D}_\Delta} \bar{\sigma}(DMD^{-1}) \tag{4-16}$$

where the equality comes from Lemma 2.2. Note that the last element in the D matrices in equation Figure 4-12 is normalized to 1 since for any nonzero scalar γ, $DMD^{-1} = (\gamma D)M(\gamma D)^{-1}$.

Bounds

In this section we will concentrate on the bounds

$$\max_{Q \in \mathbf{Q}_\Delta} \rho(QM) \le \mu_\Delta(M) \le \inf_{D \in \mathbf{D}_\Delta} \bar{\sigma}(DMD^{-1})$$

The lower bound is always an equality ([Doy]). Unfortunately, the quantity $\rho(QM)$ can have multiple local maxima which are not global. Thus local search cannot be guaranteed to obtain μ, but can only yield a lower bound. The μ software actually uses a slightly different formulation of the lower bound as a power algorithm which is reminiscent of power algorithms for eigenvalues and singular values ([PacD]). While there are open questions about convergence, the algorithm usually works quite well and has proven to be an effective method to compute μ.

The upper bound can be reformulated as a convex optimization problem, so the global minimum can, in principle, be found. Unfortunately, the upper bound is not always equal to μ. For block structures Δ satisfying $2S + F \leq 3$, the upper bound is always equal to $\mu_\Delta(M)$, and for block structures with $2S + F > 3$, there exist matrices for which μ is less than the infimum. This can be summarized in the following diagram, which shows for which cases the upper bound is guaranteed to be equal to μ.

				F		
		0	1	2	3	4
	0		yes	yes	yes	no
S	1	yes	yes	no	no	no
	2	no	no	no	no	no

Several of the boxes have connections with standard results:

- $S = 0$, $F = 1$: $\mu_\Delta(M) = \bar{\sigma}(M)$
- $S = 1$, $F = 0$: $\mu_\Delta(M) = \rho(M) = \inf\limits_{D \in \mathbf{D}_\Delta} \bar{\sigma}(DMD^{-1})$ This is a standard result in linear algebra. It is also equivalent to fact that Lyapunov stability and exponential stability are equivalent for linear systems.
- $S = 0$, $F = 2$: This case was studied by Redheffer ([Red1] and [Red2]).
- $S = 1$, $F = 1$: This is equivalent to a standard result on state-space computation of H_∞ norms.
- $S = 2$, $F = 0$: This is equivalent to the fact that for multidimensional systems (2-d, and higher), exponential stability is not equivalent to Lyapunov stability, [AndAJM], [PacD].
- $S = 0$, $F = 3$: For this case, the upper bound is always equal to μ. This important result is due to Doyle. [Doy]
- $S = 0$, $F \geq 4$: For this case, the upper bound is not always equal to μ. This is important, as these are the cases that arise most frequently in applications. Fortunately, the bound seems to be close to μ. The worst known example has a ratio of μ over the bound of about.85, and most systems are close to 1.

The above bounds are much more than just computational schemes, although that is their primary role in this toolbox. They are also theoretically rich, and can unify a number of apparently quite different results in linear systems theory. There are several connections with Lyapunov stability, two of which were hinted at above, but there are further connections between the upper bound scalings and solutions to Lyapunov and Riccati equations. Indeed, many major theorems in linear systems theory follow from the upper bounds and some results for *linear fractional transformations* (see "Linear Fractional Transformations"). The lower bound can be viewed as a natural generalization of the maximum modulus theorem [BoyD]. While a complete exposition of these ideas is beyond the scope of this tutorial, some of the more elementary concepts will be explored in later sections.

For the purposes of this toolbox, the most important use of the upper bound is as a computational scheme when combined with the lower bound. For reliable use of the μ theory it is essential to have upper and lower bounds. The other important feature of the upper bound is that it can be combined with H_∞ controller synthesis methods to yield an ad-hoc μ-synthesis method. Note that the upper bound, when applied to transfer functions, and maxed across frequencies, is simply a scaled H_∞ norm. This is exploited in the μ-synthesis techniques in this toolbox.

Computational Exercise with the mu Command

The calculation of bounds for the structured singular value (μ) is performed with the μ-Tools command `mu`. The input arguments to `mu` include the matrix on which μ is to be calculated, the block structure associated with the input matrix and an optional argument defining the type bound calculations to be performed. The outputs from `mu` are the lower and upper bound, scaling matrices used to achieve the upper bound, a sensitivity measure associated with the upper bound calculation and a perturbation matrix Δ which makes $\det(I - M\Delta) = 0$. A more detailed description of the `mu` can be found in the command reference section.

The following exercise uses the command `mu` to compute upper and lower bounds for the structured singular value of a given 5 × 5 complex matrix, for a variety of block structures. It is intended to show the dependence of $\mu_\Delta(M)$ on the particular structure Δ. The user can also verify the correctness of the bounds produced by the calculation.

The set of commands used are:

`mu`	upper and lower bounds for $\mu_\Delta(M)$
`unwrapd`	unwrap the D scaling matrices associated with upper bound
`unwrapp`	unwrap the perturbation Δ associated with lower bound

Syntax for mu

```
[bnds,dvec,sens,pvec] = mu(M,deltaset);
```

Description

`M` Matrix to calculate μ of, can be a CONSTANT or VARYING matrix.

`deltaset` block structure information about the set Δ; the number of perturbation blocks, their sizes and types. For example: the block structure

$$\Delta := \left\{ \begin{bmatrix} \delta_1 & 0 & 0 & 0 \\ 0 & \delta_2 & 0 & 0 \\ 0 & 0 & \delta_3 & 0 \\ 0 & 0 & 0 & \delta_4 \end{bmatrix} : \delta_i \in \mathbf{C} \right\}$$

is represented by the MATLAB array

$$\text{deltaset} = \begin{bmatrix} 1 & 1 \\ 1 & 1 \\ 1 & 1 \\ 1 & 1 \end{bmatrix} \quad .$$

The block structure

$$\Delta := \left\{ \begin{bmatrix} \Delta_1 & 0 & 0 \\ 0 & \Delta_2 & 0 \\ 0 & 0 & \delta_3 \end{bmatrix} : \Delta_1 \in \mathbf{C}^{3 \times 2}, \Delta_2 \in \mathbf{C}^{4 \times 5}, \delta_3 \in \mathbf{C} \right\},$$

is represented by the MATLAB array

$$\text{deltaset} = \begin{bmatrix} 3 & 2 \\ 4 & 5 \\ 1 & 1 \end{bmatrix} \quad .$$

Finally, the block structure

$$\Delta := \left\{ \begin{bmatrix} \delta_1 I_3 & 0 & 0 \\ 0 & \Delta_2 & 0 \\ 0 & 0 & \delta_3 I_2 \end{bmatrix} : \delta_1, \delta_3 \in \mathbf{C}, \Delta_2 \in \mathbf{C}^{2 \times 2} \right\}$$

is represented by the MATLAB array

$$\text{deltaset} = \begin{bmatrix} 3 & 0 \\ 2 & 2 \\ 2 & 0 \end{bmatrix} \quad .$$

Lower and Upper bounds: The output `bnds` is a 1×2 matrix. It is VARYING if M is VARYING, and CONSTANT if M is CONSTANT. The first entry of `bnds` (either `bnds(1,1)` or `sel(bnds,1,1)` in the VARYING case) is an upper bound for $\mu_\Delta(M)$, and the second entry is a lower bound for $\mu_\Delta(M)$.

Scaling matrices (these give upper bound): The matrix `dvec` contains the scaling matrices for the upper bound. As in the case for `bnds`, it is of the same type as M. Since the scaling matrices almost always have a number of zero entries, they are stored in dvec as a row vector. They can be unwrapped into the block diagonal form using unwrapd.

```
[Dl,Dr] = unwrapd(dvec,deltaset);
```

For the most part, these two matrices are the same, in fact, if the block structure deltaset has no nonsquare full blocks, then $D_l = D_r$. In any event, the following are always true

$$D_l = D_l^* > 0, D_r = D_r^* > 0,\ D_r\Delta = \Delta D_l \quad \forall \Delta \in \mathbf{\Delta}$$

and $\mu_{\Delta}(M) \le \bar{\sigma}(D_l M D_r^{-1})$.

Sensitivity (used in μ-synthesis): a sensitivity measure of the maximum singular value of $D_l M D_r^{-1}$ with respect to the values in D_l (and D_r). It is calculated in an ad-hoc manner, and is mainly used when fitting frequency varying D's with rational functions via the routine fitsys. We will not make use of it in this example.

Perturbation (gives lower bound): The matrix pvec contains the perturbation matrix Δ which makes $I - M\Delta$ singular. This perturbation corresponds to the lower bound in bnds. It is of the same type as M. Since the structured set usually contains many zero elements, the perturbation matrix Δ is stored efficiently in pvec as a row vector. It can be unwrapped into the block diagonal form using unwrapp.

```
delta = unwrapp(pvec,deltaset);
```

Note that $\bar{\sigma}(\Delta)$ is equal to the reciprocal of the lower bound (in bnds), and

$$\begin{array}{c} \Delta \in \mathbf{\Delta} \\ \det(I - M\Delta) = 0 \end{array}$$

Try this on some examples:

```
simplemu;
```

This creates a 5 × 5 matrix, M, and the different block structures, deltaseta, deltasetb,...,deltaseti.

Consider the block structure defined by the array deltasete. Run the mu command, and unwrap the *d*s and the perturbation.

```
[bnds,dvec,sens,pvec] = mu(M,deltasete);
[Dl,Dr] = unwrapd(dvec,deltasete);
delta = unwrapp(pvec,deltasete);
```

Verify that:

- $\Delta \in \mathbf{\Delta}$; print out the matrix delta, and check that its structure corresponds to that given by the array deltasete.
- Compare the norm of the matrix delta with the lower bound from bnds.

  ```
  bnds
  norm(delta);
  ```

- Note that the $\bar{\sigma}(\Delta)$ should equal the inverse of the lower bound (1 bnds(1,2)).
- Verify that $\det(I_5 - M\Delta) = 0$.

  ```
  det(eye(5)-M*delta)
  ```

  or

  ```
  eig(M*delta) % should have an eigenvalue at 1
  ```

- Look at the scaling matrices D_l and D_r. Note that $D_r\Delta = \Delta D_l$ for all $\Delta \in \mathbf{\Delta}$. Check that the upper bound in bnds comes from these.

  ```
  bnds(1,1)
  norm(Dl*M/Dr)
  ```

Try the other examples, and verify the consistency of all aspects of the results.

Mixed Real/Complex Structured Singular Value

Up until this point, this chapter has dealt only with complex-valued perturbation sets, as in equation Figure 4-5, and the complex structured singular value to assess stability and performance degradation under these types of perturbations. In specific instances, we have seen that it may be more natural to model some of the uncertainty with real perturbations, for instance when the real coefficients of a linear differential equation are uncertain. While it is possible to simply treat these perturbations as complex and proceed with the complex-μ analysis, the results can be expected to be conservative. Hence, researchers have developed algorithms for robustness tests with both real and complex perturbation blocks. The command `mu` also works for these extremely useful calculations.

Hence, there are three different types of uncertainty blocks allowed in the `mu` software:

- Repeated scalar real blocks
- Repeated scalar complex blocks
- Full complex blocks

The general structured singular value theory also includes full real blocks, but these are difficult to motivate from the physics of real problems, and convenient upper and lower bounds for structures with these types of blocks are not well developed. This is an ongoing area of research, and later versions of μ-Tools may support these types of blocks.

The theory for a mixed real/complex upper bound is more complicated to describe than the upper bound theory for complex μ. In addition to *D* matrices, which exploit the block diagonal structure of the perturbation, there are *G* matrices, which exploit the real structure of the perturbations. For illustrative purposes, consider a specific block structure. Suppose that

$$\Delta := \left\{ \begin{bmatrix} \delta_1 I_{2\times 2} & 0 & 0 \\ 0 & \delta_2 I_{4\times 4} & 0 \\ 0 & 0 & \Delta_3 \end{bmatrix} : \delta_1 \in \mathbf{R}, \delta_2 \in \mathbf{C}, \Delta_3 \in \mathbf{C}^{3\times 3} \right\}$$

In mu, this is denoted by

```
deltaset = [-2 0;4 0;3 3];
```

A zero (0) in the second column signifies a repeated scalar block (as before). The negative sign (–) indicates a real block.

Associated with Δ, define the sets

$$\mathbf{D}_\Delta = \left\{ \begin{bmatrix} D_1 & 0 & 0 \\ 0 & D_2 & 0 \\ 0 & 0 & d_3 I_{3\times 3} \end{bmatrix} : D_1 \in \mathbf{C}^{2\times 2}, \det(D_1) \neq 0 \right.$$
$$\left. D_2 \in \mathbf{C}^{4\times 4}, \det(D_2) \neq 0, d_3 \in \mathbf{C}, d_3 \neq 0 \right\}$$

and

$$\mathbf{G}_\Delta = \left\{ \begin{bmatrix} \mathbf{diag}[g_i]_{i=1,2} & 0 & 0 \\ 0 & 0_{4\times 4} & 0 \\ 0 & 0 & 0_{2\times 2} \end{bmatrix} : g_i \in \mathbf{R} \right\}$$

If there is a $\beta > 0$, $D \in \mathbf{D}_\Delta$ and $G \in \mathbf{G}_\Delta$ such that

$$\bar{\sigma}\left[(I + G^2)^{-\frac{1}{4}} \left(\frac{1}{\beta} DMD^{-1} - jG \right) (I + G^2)^{-\frac{1}{4}} \right] \leq 1 \tag{4-17}$$

then

$$\mu_\Delta(M) \leq \beta$$

This bound, [YouND1], is a derivative of an earlier bound in [FanTD]. The smallest $\beta < 0$ for which D and G matrices exist which satisfy this constraint is what μ-Tools calls the mixed μ upper bound for M. Using manipulations that are now standard in robust control theory, the computation of the best such β is reformulated into an Affine Matrix Inequality (AMI) and solved with special purpose convex programming techniques. For perturbation sets with multiple blocks, the general structure of the sets $\mathbf{D}_\Delta$ and $\mathbf{G}_\Delta$ remains the same, with one scaling block for each uncertainty block.

In mu, only the complex full blocks can be nonsquare. This causes the D scaling on the left of M to be slightly different from the D scaling on the right. The single d variable associated with the full block is repeated a certain number of times on the left, and a different number of times on the right, leading to nonsquare D scaling matrices (the D_l and D_r that we have already seen). Of course, the G scaling comes in different sizes too. Note that for any complex full blocks, the associated blocks of G are zeros, since G is only nonzero in the blocks associated with the real uncertainties. However, the dimension of the zero blocks of G must line up with the correct rows/columns of M. Hence, in equation Figure 4-17, there are three different Gs, all having exactly the same nonzero elements, but different sizes of zero blocks associated with any nonsquare full blocks. The different Gs are denoted G_l, G_m, and G_r. The sufficient condition for the $\mu_\Delta(M) < \beta$ is rewritten as

$$\bar{\sigma}\left[(I + G_l^2)^{-\frac{1}{4}}\left(\frac{1}{\beta}D_l M D_r^{-1} - jG_m\right)(I + G_r^2)^{-\frac{1}{4}}\right] \le 1 .$$

(4-18)

The lower bound generated in mu comes from a power iteration which finds matrices $\Delta \in \mathbf{\Delta}$ that make $I - M\Delta$ singular. The power iteration for mu is a generalization of the power iteration used in earliest versions of μ-Tools. The generalization is described in detail in [YouD].

The combination of upper and lower bounds makes the mu software unique. The upper bounds give a guarantee on the worst-case performance degradation that can occur under perturbation. The lower bounds actually exhibit a perturbation that causes significant performance degradation. This perturbation can then be used in time-domain simulations to better understand its effect.

Specifics About Using the mu Command with Mixed Perturbations

The syntax of mu and muunwrap are

```
[bnds,dvec,sens,pvec,gvec] = mu(M,deltaset,options);
[dl,dr,gl,gm,gr] = muunwrap(dvec,gvec,deltaset);
[delta] = muunwrap(pvec,deltaset);
```

Their purposes are:

mu	general mixed μ computation
muunwrap	extract block-diagonal scalings *D* and *G* from row vectors, and extract from row vector the smallest perturbation Δ which causes singularity.

Other functions associated with mu (such as unwrapd and unwrapp, which have been illustrated already) are detailed in Chapter 8, "Reference" under mu.

Computational Exercise with the mu Command — Mixed Perturbations

The input argument deltaset is as before, but real repeated scalar blocks are specified with a negative integer rather than a positive integer. For example, deltaset = [-3 0; 2 3] is similar to deltaset = [3 0; 2 3] except that the 3 × 3 repeated block is treated as real rather than complex. Also note that redefining deltaset = abs(deltaset) always gives the complex version of a mixed μ problem.

For instance, the block structure

$$\Delta := \left\{ \begin{bmatrix} \delta_1 & 0 & 0 & 0 \\ 0 & \delta_2 & 0 & 0 \\ 0 & 0 & \delta_3 & 0 \\ 0 & 0 & 0 & \delta_4 \end{bmatrix} : \delta_i \in \mathbf{C}, i = 1, 2, \delta_j \in \mathbf{R}, j = 3, 4. \right\}$$

is represented by the array

```
deltaset = [1 1;1 1;-1 0;-1 0];
```

The block structure

$$\Delta = \left\{ \mathrm{diag}\,[\Delta_1 \;\; \Delta_2 \;\; \delta_3 I_{3\times3}] : \Delta_1 \in \mathbf{C}^{3\times2}, \Delta_2 \in \mathbf{C}^{4\times5}, \delta_3 \in \mathbf{R} \right\},$$

is represented by the array

```
deltaset = [3 2;4 5;-3 0];
```

Finally, the block structure

$$\Delta := \left\{ \begin{bmatrix} \delta_1 I_{3\times3} & 0 & 0 \\ 0 & \Delta_2 & 0 \\ 0 & 0 & \delta_3 I_{2\times2} \end{bmatrix} : \delta_1 \in \mathbf{C}, \delta_3 \in \mathbf{R}, \Delta_2 \in \mathbf{C}^{2\times2} \right\}$$

is represented by the array

```
deltaset = [3 0;2 2;-2 0];
```

The correctness of the upper bound can easily be checked with the inequality in equation Figure 4-18. The correctness of the lower bound can be verified by calculating the perturbation, Δ, that `mu` returns, verifying its block structure and norm, checking that the matrix $M\Delta$ has an eigenvalue exactly at 1 (which is equivalent to $I - M\Delta$ being singular).

Try some examples on a constant 5×5 matrix.

```
simprmu
```

This loads a 5×5 complex matrix, M, and different block structures `deltaseta, deltasetb,...,deltaseti`. Consider the block structure Δ defined by `deltasete`, Run the `mu` command, and unwrap the D, G, and perturbation matrices.

```
[bnds,dvec,sens,pvec,gvec] = mu(M,deltasete);
[dl,dr,gl,gm,gr] = muunwrap(dvec,gvec,deltasete);
[delta] = muunwrap(pvec,deltasete);
```

Verify that:

- $\Delta \in \mathbf{\Delta}$; print out delta, and check that its block structure corresponds to that given by deltasete.

  ```
  deltasete
  delta
  ```

- Compare the lower bound from μ with the norm of Δ.

  ```
  [bnds(1,2) 1/norm(delta)]
  ```

- Verify that $\det(I_5 - M\Delta) = 0$

  ```
  eig(M*delta)
  ```

- Verify the upper bound by checking the structure of dl, dr, gl, gm, and gr, and the inequality in equation Figure 4-18.

  ```
  deltasete
  dl
  dr
  gl
  gm
  gr
  oobdmdimjg = 1/bnds(1,1)*dl*mat/dr - sqrt(-1)*gm;
  gscl_l = inv(sqrtm(sqrtm(eye(5) + gl*gl))));
  gscl_r = inv(sqrtm(sqrtm((eye(5) + gr*gr))));
  norm(gscl_l*oobdmdimjg*gscl_r)
  ```

Try the other block structures, and verify the consistency of all aspects of the bounds, scaling matrices, and perturbations.

Linear Fractional Transformations

Using only the definition of μ, some simple theorems about a class of general matrix transformations called linear fractional transformations (LFTs) can be proven. To introduce these, consider a complex matrix M partitioned as

$$M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \tag{4-19}$$

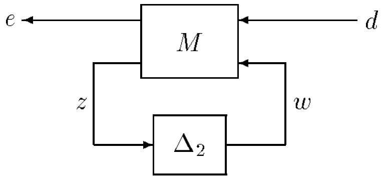
and suppose there is a defined block structure Δ_2 which is compatible in size with M_{22} (for any $\Delta_2 \in \Delta_2$, $M_{22}\Delta_2$ is square). For $\Delta_2 \in \Delta_2$, consider the following loop equations,

$$\begin{aligned} e &= M_{11}d + M_{12}w \\ z &= M_{21}d + M_{22}w \\ w &= \Delta_2 z \end{aligned} \tag{4-20}$$

This set of equations Figure 4-20 is called *well posed* if for any vector d, there exist unique vectors w, z, and e satisfying the loop equations. It is easy to see that the set of equations is well posed if and only if the inverse of $I - M_{22}\Delta_2$ exists. If not, then depending on d and M, there is either no solution to the loop equations, or there are an infinite number of solutions. When the inverse does indeed exist, the vectors e and d must satisfy $e = F_L(M,\Delta_2)d$, where

$$F_L(M, \Delta_2) = M_{11} + M_{12}\Delta_2(I - M_{22}\Delta_2)^{-1}M_{21} \tag{4-21}$$

$F_L(M,\Delta_2)$ is a linear fractional transformation on M by Δ_2, and in a feedback diagram appears as



The subscript L on F_L pertains to the *lower* loop of M and is closed by Δ_2. An analogous formula describes $F_U(M,\Delta_1)$, which is the resulting matrix obtained by closing the *upper* loop of M with a matrix $\Delta_1 \in \boldsymbol{\Delta}_1$.

In this formulation, the matrix M_{11} is assumed to be something nominal, and $\Delta_2 \in \mathbf{B}_2 := \{\Delta_2 \in \boldsymbol{\Delta}_2 : \bar{\sigma}(\Delta_2) \leq 1\}$ is viewed as a norm-bounded perturbation from an allowable perturbation class, $\boldsymbol{\Delta}_2$. The matrices M_{12}, M_{21}, and M_{22} and the formula F_L reflect prior knowledge on how the unknown perturbation affects the nominal map, M_{11}. This type of uncertainty, called *linear fractional*, is natural for many control problems, and encompasses many other special cases considered by researchers.

The constant matrix problem to solve is:

- Determine whether the LFT is well posed for all $\Delta_2 \in \mathbf{B}_2$, and,
- If so, then determine how *large* $F_L(M,\Delta_2)$ can get for this norm-bounded set of perturbations.

The next section has three simple theorems which answer this problem.

Well Posedness and Performance for Constant LFTs

Let M be a complex matrix partitioned as

$$M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix} \tag{4-22}$$

and suppose there are two defined block structures, $\boldsymbol{\Delta}_1$ and $\boldsymbol{\Delta}_2$, which are compatible in size with M_{11} and M_{22} respectively. Define a third structure $\boldsymbol{\Delta}$ as

$$\boldsymbol{\Delta} = \left\{ \begin{bmatrix} \Delta_1 & 0 \\ 0 & \Delta_2 \end{bmatrix} : \Delta_1 \in \boldsymbol{\Delta}_1, \Delta_2 \in \boldsymbol{\Delta}_2 \right\}. \tag{4-23}$$

Now there are three structures with which we may compute μ. The notation we use to keep track of this is as follows: $\mu_1(\cdot)$ is with respect to $\boldsymbol{\Delta}_1$, $\mu_2(\cdot)$ is with respect to $\boldsymbol{\Delta}_2$, : $\mu_{\boldsymbol{\Delta}}(\cdot)$ is with respect to $\boldsymbol{\Delta}$. In view of this, $\mu_1(M_{11})$, $\mu_2(M_{22})$, and $\mu_{\boldsymbol{\Delta}}(M)$ all make sense, though for instance, $\mu_1(M)$ does not. Again, define the norm-bounded perturbation sets as

$$\mathbf{B}_i := \{\Delta_i \in \boldsymbol{\Delta}_i : \bar{\sigma}(\Delta_i) \leq 1\}.$$

Let $\Delta_2 \in \boldsymbol{\Delta}_2$. The linear fractional transformation, $F_L(M,\Delta_2)$ is well posed if I - $M_{22}\Delta_2$ is invertible, and in that case is defined as

$$F_L(M, \Delta_2) = M_{11} + M_{12}\Delta_2(I - M_{22}\Delta_2)^{-1}M_{21} \tag{4-24}$$

The first theorem is nothing more than restating the definition of μ.

Theorem 4.4: The linear fractional transformation $F_L(M,\Delta_2)$ is well posed for all $\Delta_2 \in \mathbf{B}_2$ if and only if $\mu_2(M_{22}) < 1$.

As the *perturbation* Δ_2 deviates from zero, the matrix $F_L(M,\Delta_2)$ deviates from M_{11}. The range of values that $\mu_1(F_L(M,\Delta_2))$takes on is intimately related to $\mu_\Delta(M)$, as follows:

Theorem 4.5 [Main Loop Theorem]: The following are equivalent:

1 $\mu_\Delta(M) <1$

2 $\mu_2(M_{22}) < 1$, and

$$\max_{\Delta_2 \in \mathbf{B}_2} \mu_1(F_L(M, \Delta_2)) < 1$$

3 $\mu_1(M_{11}) < 1$, and

$$\max_{\Delta_1 \in \mathbf{B}_1} \mu_2(F_U(M, \Delta_1)) < 1$$

Proof: The proof is based on Schur formulae for determinants of block partitioned matrices, and can be found in [PacD]. The basic idea is simple, and we present the proof showing 3 → 1.

Let $\Delta_i \in \Delta_i$ be arbitrary, with $\bar{\sigma}(\Delta_i) \le 1$. Define

$$\Delta := \begin{bmatrix} \Delta_1 & 0 \\ 0 & \Delta_2 \end{bmatrix}$$

Obviously, $\Delta \in \Delta$, and $\bar{\sigma}(\Delta) \le 1$.

Now

$$\det(I - M\Delta) = \det\begin{bmatrix} I - M_{11}\Delta_1 & -M_{12}\Delta_2 \\ M_{21}\Delta_1 & I - M_{22}\Delta_2 \end{bmatrix}$$

Since $\mu_1(M_{11}) < 1$, and $\bar{\sigma}(\Delta_1) \le 1$, it follows that $I - M_{11}\Delta_1$ is invertible, giving (using the Schur formula)

$$\det(I - M\Delta) =$$
$$\det(I - M_{11}\Delta_1)\det(I - M_{22}\Delta_2 + -M_{21}\Delta_1(I - M_{11}\Delta_1)^{-1}M_{12}\Delta_2)$$

Collecting the Δ_2 terms leaves

$$\det(I - M\Delta) = \det(I - M_{11}\Delta_1)\det(I - F_U(M,\Delta_1)\Delta_2)$$

Since each $\Delta_i \in \mathbf{B}_i$, the assumption implies that $\mu_2(F_U(M,\Delta_1)) < 1$. By definition, then,

$$I - F_U(M,\Delta_1)\Delta_2$$

is nonsingular, and so then is $I - M\Delta$. Now, this argument holds for any $\Delta_i \in \Delta_i$, hence the definition of μ gives

$$\mu_\Delta(M) < 1$$

Remark: This theorem forms the basis for all uses of μ in linear system robustness analysis, whether from a state-space, frequency domain, or Lyapunov approach.

Frequency Domain μ Review

Since the frequency domain μ tests play a key role in robustness analysis, we summarize in this section the computational procedure and its subsequent interpretations. These involve not only constant, complex structured perturbations, but also linear, time-invariant, dynamical perturbations (unmodeled dynamics) as well.

Robust Stability

The most well-known use of μ as a robustness analysis tool is in the frequency domain. Suppose $M(s)$ is a stable, multi-input, multi-output transfer function of a linear system, M. For clarity, assume M has n_z inputs and n_w outputs. Let Δ be a block structure, as in equation Figure 4-5, and assume that the dimensions are such that $\Delta \subset \mathbf{C}^{n_z \times n_w}$. We want to consider feedback perturbations to M which are themselves dynamical systems, with the block-diagonal structure of the set Δ.

In this section, we outline the proofs for the situation where the perturbations are assumed to be stable. This is not a restriction with parametric real uncertainty, as constant parameters are clearly stable. However, when using a multiplicative or additive unmodeled dynamics perturbation to model uncertainty in an unstable component (see the example in the “Unmodeled Dynamics” section), it is useful to allow unstable perturbations, with the restriction that the number of right-half-plane poles of the component remains constant. An alternate approach is to allow a block of the perturbation matrix to be an unstable transfer function. However, we restrict it to only take on values that preserve the number of right-half-plane poles of the perturbed component with which it is associated. In this case, the theorems we state are still correct, though the proofs must be modified. In fact, even more sophisticated assumptions about the perturbed systems can be made, including structured coprime factor uncertainty and gap metric uncertainty, but these are beyond the scope of this tutorial.

Let $\mathbf{S}$ denote the set of real-rational, proper (no poles at $s = \infty$), stable, transfer matrices. Associated with any block structure Δ, let $\mathbf{S}_\Delta$ denote the set of all block diagonal, stable rational transfer functions, with block structure like Δ.

$$\mathbf{S}_\Delta := \{\Delta \in \mathbf{S} : \Delta(s_o) \in \Delta \text{ for all } s_o \in \overline{\mathbf{C}}_+\}$$

Theorem 4.6: Let $\beta > 0$. The loop shown in Figure 4-23 is well-posed and internally stable for all $\Delta \in \mathbf{S}_\Delta$ with $\|\Delta\|_\infty < \frac{1}{\beta}$ if and only if

$$\|M\|_\Delta := \sup_{\omega \in \mathbf{R}} \mu_{\$\Delta}(M(j\omega)) \leq \beta$$

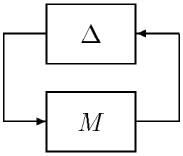


Figure 4-23: Robust Stability

Explanation of proof: This result is proven using the definition of μ, and the multivariable Nyquist criterion, [CheD]. Since $M(s)$ and $\Delta(s)$ are assumed stable, the loop is internally stable if and only if the Nyquist plot of $\det\,(I - M(j\omega)\Delta(j\omega))$ does not pass through, or encircle the origin. If $\|M\|_\Delta \leq \beta$, this guarantees that for any $\Delta \in \mathbf{S}_\Delta$, with $\|\Delta\|_\infty < \frac{1}{\beta}$, any $\omega \in \mathbf{R}$, and any $\alpha \in [0, 1]$, the determinant

$$\det[I - M(j\omega)(\alpha\ \Delta(j\omega))] \neq 0$$

This guarantees that the Nyquist plot of $\det[I]$, (the above expression with $\alpha = 0$) and the Nyquist plot of $\det[I - M(j\omega)\Delta(j\omega)]$, (above expression with $\alpha = 1$) encircle the origin the same number of times. It is obvious that the Nyquist plot of $\det[I]$ is just a single point at 1, and hence does not encircle the origin at all, therefore the Nyquist plot of $\det[I - M(j\omega)\Delta(j\omega)]$ does not pass through, or encircle the origin, and hence the loop is indeed stable. Conversely, if $\|M\|_\Delta > \beta$, then at some particular frequency $\bar{\omega}$, $\mu_\Delta(M(j\bar{\omega})) > \beta$. By definition of $\mu_\Delta(\cdot)$, this means there is a constant matrix $\Delta \in \mathbf{\Delta}$ such that $\bar{\sigma}(\Delta) < \frac{1}{\beta}$, and $\det(I - M(j\bar{\omega})\Delta) = 0$. Now, it is possible to find a real-rational, stable block diagonal transfer matrix $\Delta \in \mathbf{S}_\Delta$ such that $\|\Delta\|_\infty < \frac{1}{\beta}$, and $\Delta(j\bar{\omega}) = \Delta$. Hence, using this real-rational Δ, the loop will have a closed-loop pole at $s = j\bar{\omega}$. A more careful proof of these ideas is found in [CheD]. An alternate proof using the maximum-modulus property of μ can be found in [PacP].

In summary, the peak value on the μ plot of the frequency response that the perturbation *sees* determines the size of perturbations that the loop is robustly stable against.

Robust Performance

Stability is not the only property of a closed-loop system that must be robust to perturbations. Typically there are exogenous disturbances acting on the system (wind gusts, sensor noise) which result in tracking and regulation errors. Under perturbation, the effect that these disturbances have on error signals can greatly increase. In most cases, long before the onset of instability, the closed-loop performance will degrade to the point of unacceptability. Hence the need for a *robust performance* test. Such a test will indicate the worst-case level of performance degradation associated with a given level of perturbations.

Assume M is a stable, real-rational, proper transfer function, with $n_z + n_d$ inputs, and $n_w + n_e$ outputs. Partition M in the obvious manner, so that M_{11} has n_z inputs and n_w outputs, and so on. Let $\boldsymbol{\Delta} \subset \mathbf{C}^{n_w \times n_z}$ be a block structure, as in equation Figure 4-5. Define an augmented block structure

$$\boldsymbol{\Delta}_P := \left\{ \begin{bmatrix} \Delta & 0 \\ 0 & \Delta_F \end{bmatrix} : \Delta \in \boldsymbol{\Delta}, \Delta_F \in \mathbf{C}^{n_d \times n_e} \right\}$$

The setup is to address theoretically the robust performance questions about the loop shown in Figure 4-24.

The perturbed transfer function from d to e is denoted by $F_U(M,\Delta)$.

Theorem 4.7: Let $\beta > 0$. For all $\Delta(s) \in \mathbf{S}_\Delta$ with $\|\Delta\|_\infty < \frac{1}{\beta}$, the loop shown above is well-posed, internally stable, and $\|F_U(M,\Delta)\|_\infty \le \beta$ if and only if

$$\|M\|_{\Delta_P} := \sup_{\omega \in \mathbf{R}} \mu_{\Delta_P}(M(j\omega)) \le \beta$$

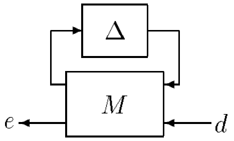


Figure 4-24: Robust Performance

The proof of this is exactly along the lines of the earlier proof, but also applying Theorem 4.5. See [DoyWS] and [PacD] for details.

Real vs. Complex Parameters

There are many approaches to model real, uncertain parameters. Suppose that a coefficient c, in a particular system, is assumed to be constant, but unknown, and the value of c is modeled to lie in an interval, say,

$$c \in [0.8\ 1.6]$$

This can be modeled effectively with a real perturbation,

$$c \in \{1.2 + (0.4)\delta : \delta \in \mathrm{R},\ |\delta| \le 1\}$$

Clearly, this set captures the uncertainty in the coefficient c. What is the correct interpretation of the uncertain set model if δ is taken as complex,

$$c \in \{1.2 + (0.4)\delta : \delta \in \mathrm{C},\ |\delta| \le 1\}\ ?$$

In a linear, time-invariant system, robustness to this constant, complex uncertain parameter c is mathematically *equivalent* to robustness to all stable, linear, time-invariant transfer functions, $\hat{c}(s)$, whose Nyquist plots lie in the disk shown in Figure 4-25.

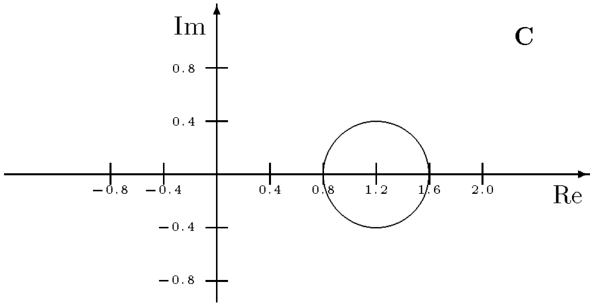


Figure 4-25: Complex Disc Covering Real Interval: Restriction of the Nyquist Plot of $\hat{c}$

Using complex parameters, the uncertain model for c represents a stable linear system whose characteristics are similar to an uncertain real gain, but deviate in a manner quantified by the disc-shaped constraint on its frequency response. In general, using disks instead of intervals leads to more conservative robustness properties. With `mu`, it is easy and fast to explore the differences in the robustness properties as the uncertainty model changes.

Block Structures with All Real Blocks

The function μ is not necessarily a continuous function when all of the perturbation blocks are real. This mathematical fact is pointed out in [BarKST], and an example is given where the robustness margin to real parameter uncertainty changes abruptly for infinitesimal changes in the problem data. Also, in the [BarKST] example, the structured singular value of the frequency response exhibits discontinuities across frequency.

What is the significance of these issues on `mu`? The discontinuities can cause problems in the convergence of the lower bound power algorithm. For problems with purely real uncertainty, the lower bound algorithm may converge to a value which is significantly lower than μ itself, or may not even converge at all. This could be a serious problem, but usually it is not, because almost all problems have a full complex block associated with a robust performance specification. It turns out that if a μ problem has a complex block that *counts*, then the function μ will be continuous at the problem data.

Sometimes, though, a robust stability calculation for an uncertain system with only real uncertainties is needed. Consider the robust stability problem represented in Figure 4-26.

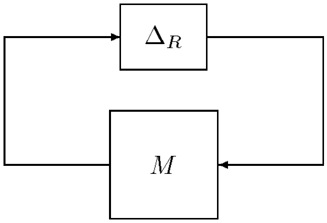


Figure 4-26: Robust Stability with Real Uncertainty

While the upper bound from `mu` will be effective, the lower bound potentially will have convergence problems, yielding little information in terms of *bad perturbations*. A fix, which has both engineering and mathematical justification, is available.

In μ-Tools vernacular, let M be the VARYING matrix (usually a frequency response) of which μ is to be computed. Let blkrs_R be the real block structure, that is compatible in dimension with M. Recall that abs(blkrs_R) is exactly the same block structure as blkrs_R, but consists of complex blocks. Define:

```
pdim = ynum(M); %only real blocks --> square
alpha = 0.1;
fixl = [eye(pdim) ; alpha*eye(pdim)];
fixr = fixl';
% duplicate complex blocks below reals
blkrs_RC = [blkrs_R ; abs(blkrs_R)];
Mmix = mmult(fixl,M,fixr);
[bnd_RC,rowd_RC,sens,rowp_RC] = mu(Mmix,blkrs_RC);
```

What does all of this mean? The block structure has been expanded (to twice the original size) by including complex blocks which are exactly the same dimension as the original real blocks. The matrix has been expanded (to twice the original size) by the multiplication on the left and right. The scale factors of α = 0.1 imply that the input/output channels, which the complex uncertainty affects, are each scaled down by a factor of 10, giving an overall scaling of the complex blocks of 0.01. The μ calculation determines upper and lower bounds for robust stability in the uncertain system shown in Figure 4-27.

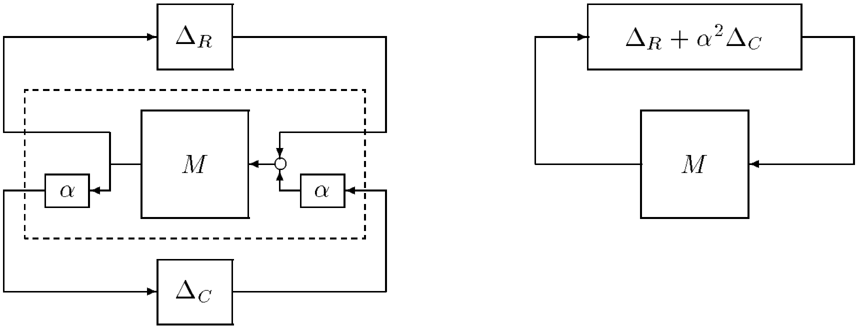


Figure 4-27: Replacing Real Uncertainty with Real+Complex Uncertainty

In this process, each real parameter δ_R has been replaced by a real parameter plus a smaller complex parameter ($\alpha^2\delta_C$). Rather than computing robustness margins to purely real parameters, the modified problem determines the robust stability characteristics of the system with respect to predominantly real uncertainties, though each uncertainty is allowed to have a very small complex part. This slight variation in the uncertainty model is easy to accept in engineering problems, since models of uncertainty are rarely fixed, and small amounts of phase in coefficients of physical models usually can be explained by some underlying dynamics that have been ignored. It can also be proven that as the parameter α converges to 0, the calculated robustness margin, which is a function of α, converges to the robustness margin associated with just the original real parameters.

In summary, the modified problem contains both real and complex uncertainty; consequently:

- μ with respect to this structure has guaranteed continuity properties.
- The lower bound from mu generally has better convergence behavior.

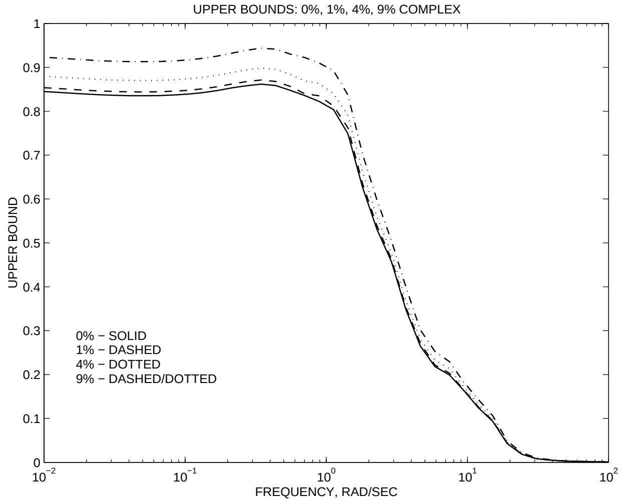
Moreover, although the modified problem is not equivalent to the original problem, it has relevant engineering interpretation and provable convergence properties as the complex weighting α goes to 0.

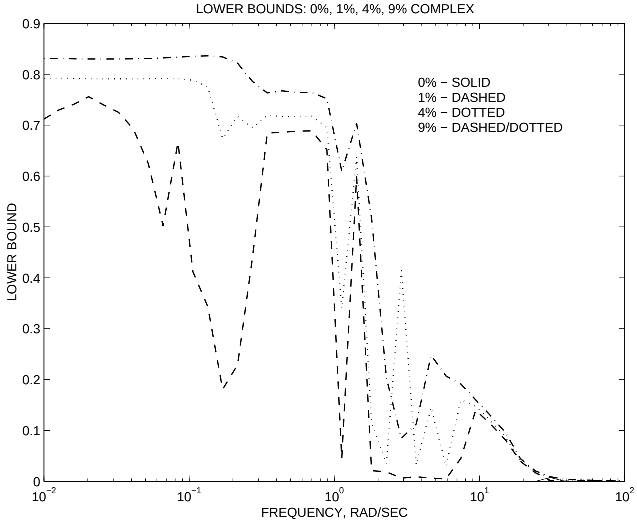
As an example of this, consider the 9 × 9 transfer function $M(s)$, with a robust stability uncertainty block structure of

$$\Delta_R := \{\mathbf{diag}[\delta_i]_{i=1,...,9} : \delta_i \in \mathbf{R}\},$$

from the “Space Shuttle Robustness Analysis” section in Chapter 7. Using mu, calculate the robust stability of the system in Figure 4-26, and for $\alpha = 0.1, 0.2, 0.3$ in Figure 4-27.

```
load shut_rs
minfo(clp_muRS)
blkrs_R = [-1 0;-1 0;-1 0;-1 0;-1 0;-1 0;-1 0;-1 0;-1 0];
clp_muRSg = frsp(clp_muRS,logspace(-2,2,40));
fix1 = [eye(9) ; 0.1*eye(9)];
fix2 = [eye(9) ; 0.2*eye(9)];
fix3 = [eye(9) ; 0.3*eye(9)];
blk_RC = [blkrs_R ; abs(blkrs_R)];
m1 = mmult(fix1,clp_muRSg,fix1');
m2 = mmult(fix2,clp_muRSg,fix2');
m3 = mmult(fix3,clp_muRSg,fix3');
[rbnd,rp] = mu(clp_muRSg,blkrs_R);
[bnd1,rd1,s1,rp1] = mu(m1,blk_RC);
[bnd2,rd2,s2,rp2] = mu(m2,blk_RC);
[bnd3,rd3,s3,rp3] = mu(m3,blk_RC);
allbnds = abv(rbnd,bnd1,bnd2,bnd3);
vplot('liv,d',sel(allbnds,':',1))
title('UPPER BOUNDS: 0%, 1%, 4%, 9% COMPLEX')
xlabel('FREQUENCY, RAD/SEC')
ylabel('UPPER BOUND')
vplot('liv,d',sel(allbnds,':',2))
title('LOWER BOUNDS: 0%, 1%, 4%, 9% COMPLEX')
xlabel('FREQUENCY, RAD/SEC')
ylabel('LOWER BOUND')
```





For each case, the upper bound shows a slight increase as the percentage of allowable complex perturbation is increased. This is expected. The lower bound behaves similarly, though the introduction of very small complex terms has a more dramatic effect. For 0%, the lower bound from mu is zero — the program is simply unable to find purely real perturbations which cause singularity. However, upon introducing a 1% complex term in each perturbation, the lower bound jumps up to 0.76, indicating mu has found a diagonal perturbation causing singularity, with each term of the perturbation having the form δ_R + $0.01\delta_C$, $\delta_R \in \mathbf{R}$, $\delta_C \in \mathbf{C}$, and

$$|\delta_R| \leq \frac{1}{0.76}, \qquad |\delta_C| \leq \frac{1}{0.76}$$

Using unwrapp, this perturbation can be constructed. For larger values of complex contribution, the lower bound is even better behaved, and more closely matches up with the corresponding upper bound. Of course, the relevance of these nonreal perturbations on the actual robust stability properties must be assessed through careful engineering judgement and considerations.

[PacP] contains a more detailed discussion of the continuity properties of mixed μ, and the general relationship between the robustness bound in Figure 4-26 and Figure 4-27.

Finally, in defense of mu — usually, robustness problems involve a performance objective, beyond robust stability. This performance objective leads to a full complex block in the uncertainty block structure (when calculating robust performance tests, the performance block must be a complex block). In those cases, the lower bound performance of mu is quite well-behaved, and the small complex augmentation to the real blocks is unnecessary. This real perturbation problem is explored in detail in the “Space Shuttle Robustness Analysis” section in Chapter 7. Also, the robust performance characteristics are successfully computed using mu, without adding small amounts of complex uncertainty to each real parameter.

Generalized μ

Generalized μ allows us to put additional constraints on the directions that I - $M\Delta$ becomes singular. Given a matrix $M \in \mathbf{C}^{n \times n}$, and $C \in \mathbf{C}^{m \times n}$, find the smallest $\Delta \in \boldsymbol{\Delta}$ such that

$$\begin{bmatrix} I - \Delta M \\ C \end{bmatrix}$$

is not full column rank. In terms of a mathematical expression, we write

$$\mu_{\Delta}(M, C) := \frac{1}{\min\left\{\bar{\sigma}(\Delta) : \Delta \in \boldsymbol{\Delta}, \mathrm{rank}\begin{bmatrix} I - \Delta M \\ C \end{bmatrix} < n\right\}}$$

This quantity can be bounded above easily, using standard μ ideas. Suppose

$$\mu_{\Delta}(M,\mathrm{C}) \geq \beta$$

Then, there is a $\Delta \in \boldsymbol{\Delta}$, $\bar{\sigma}(\Delta) \leq \frac{1}{\beta}$ and a nonzero vector x such that

$$(I - \Delta M)x = \theta_n, \qquad Cx = \theta_m$$

Hence, for every matrix $Q \in \mathbf{C}^{n \times m}$, it follows that

$$(I - \Delta(M + QC))x = \theta_n$$

so that for every matrix $Q \in \mathbf{C}^{n \times m}$, $\mu_{\Delta}(M + QC) \geq \beta$.

By contrapositive, if there exists a matrix Q such that

$$\mu_{\Delta}(M + QC) < \beta$$

then $\mu_{\Delta}(M,\mathrm{C}) < \beta$.

Hence, we have

$$\mu_{\Delta}(M, C) \leq \min_{Q \in \mathbf{C}^{n \times m}} \mu_{\Delta}(M + QC)$$

Now, it is possible to compute the optimal matrix Q which minimizes the standard upper bound for $\mu_{\Delta}(M + QC)$. The optimization problem can be reformulated into an affine matrix inequality, [PacZPB], and solved with a combination of heuristics and general purpose AMI solvers. This is how `genmu` computes the upper bound $\mu_{\Delta}(M,\mathrm{C})$.

Using the mu Software

One of the most important uses of the mu software is to study the sensitivity of the calculated robustness properties to the uncertainty models themselves. That is, if a particular perturbation's model is changed from real to complex, how significantly does this affect the computed robustness properties? Or, if the weighting coefficient on a particular perturbation is increased by 40%, how significantly does this impact the computed robustness properties? If the change is significant, then the robustness properties are extremely sensitive to the uncertainty model (real vs. complex, effective size of the perturbation). The appropriate and relevant representation of the uncertainty becomes an issue of importance.

From an engineering perspective, one must have confidence in the appropriateness of the uncertainty model on which decisions are ultimately made. While μ-Tools cannot resolve these questions, careful use of the mu software can help you assess the overall robustness of the closed-loop system, and give a better understanding of the destabilizing mechanisms that are present.

References

Additional information on the structured singular value, linear fractional transformations and their use in robustness analysis on uncertain linear systems, as well as a historical account of the development of the theory can be found in the following references.

[AndAJM:] B. Anderson, P. Agathoklis, E. Jury, and M. Mansour, "Stability and the matrix Lyapunov equation for discrete 2-dimensional systems," *IEEE Transactions on Circuits and Systems*, Vol. 33, no. 3, pp. 261-267, March 1986.

[BarKST:] B. Barmish, P. Khargonekar, Z. Shi, and R. Tempo, "Robustness margin need not be a continuous function of the problem data," *Systems and Control Letters*, Vol. 15, pp. 91-98, 1989.

[BoyD:] S. Boyd and C. Desoer, "Subharmonic functions and performance bounds on linear time-invariant feedback systems," *IMA Journal of Mathematical Control and Information*, Vol. 2, pp. 153-170, 1985.

[BoyE:] Boyd, S., and El Ghaoui, L., "Method of centers for minimizing generalized eigenvalues," *Linear Algebra and Its Applications*, special issue on Numerical Linear Algebra Methods in Control, Signals and Systems, vol. 188, pp. 63-111, July 1993.

[CheD:] M.J. Chen and C.A. Desoer, "Necessary and sufficient condition for robust stability of linear distributed feedback systems," *International Journal of Control*, Vol. 35, no. 2, pp. 255-267, 1982.

[Doy:] J.C. Doyle, "Analysis of feedback systems with structured uncertainties," *IEEE Proceedings*, Vol. 129, Part D, no. 6, pp. 242-250, Nov. 1982.

[DoyPZ:] J. Doyle, A. Packard, and K. Zhou, "Review of LFTs, LMIs and μ," *Proceedings of the 30th IEEE Conference on Decision and Control*, pp. 1227-1232, 1991.

[DoyS:] 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, Feb. 1981.

[DoyWS:] J.C. Doyle, J. Wall and G. Stein, "Performance and robustness analysis for structured uncertainty," *Proceedings of the 21st IEEE Conference on Decision and Control*, pp. 629-636, Dec. 1982.

[FanT:] M.K.H. Fan and A.L. Tits, “Characterization and Efficient Computation of the Structured Singular Value,” *IEEE Trans. Auto. Control*, Vol. AC-31, no. 8, pp. 734-743, Aug. 1986.

[FanTD:] M. Fan, A. Tits, and J. Doyle, “Robustness in the presence of joint parametric uncertainty and unmodeled dynamics,” *IEEE Trans. Auto. Control*, vol. 36, pp. 25-38, Jan. 1991.

[Kai:] T. Kailath, *Linear Systems*, Prentice-Hall, 1980.

[MorM:] B. Morton and R. McAfoos, “A Mu-test for real parameter variations,” *Proceedings of the American Control Conference*, pp. 135-138, 1985.

[Osb:] E.E. Osborne, “On Preconditioning of Matrices,” *J. Assoc. Comp. Math*., 7, pp. 338-345, 1960.

[Ove:] M. Overton, “Large-Scale Optimization of Eigenvalues,” *SIAM Journal of Optimization*, Vol. 2, no. 1, pp. 88-120, February 1992.

[PacD:] A. Packard and J. Doyle, “The complex structured singular value,” *Automatica*, Vol. 29, pp. 71-109, 1993.

[PacP:] A. Packard and P. Pandey, “Continuity of the real/complex structured singular value,” *IEEE Transactions on Automatic Control*, Vol. 38, no. 3, pp. 415-428, March 1993.

[PacZPB:] A. Packard, K. Zhou, P. Pandey, and G. Becker, “A collection of robust control problems leading to LMI’s,” *30th IEEE Conference on Decision and Contro*l, pp. 1245-1250, Brighton, UK, 1991.

[Red1:] R. Redheffer, “Inequalities for a matrix Riccati equation,” *Journal of Mathematics and Mechanics*, Vol. 8, no. 3, 1959.

[Red2:] Redheffer, “On a certain linear fractional transformation,” *Journal of Mathematical Physics* Vol. 39. pp. 269-286, 1960.

[Saf1:] M.G. Safonov, “Tight bounds on the response of multivariable systems with component uncertainty,” *Allerton Conference on Communication, Control and Computing*, pp. 451-460, 1978.

[Saf2:] M.G. Safonov, “Stability and robustness of multivariable feedback systems,” Ph.D. dissertation, MIT Press, 1980.

[Saf3:] M.G. Safonov, “Stability margins of diagonally perturbed multivariable feedback systems,” *Proceedings of the IEE*, Vol. 129, Part D, no. 6, pp. 251-256, Nov. 1982.

[YouD:] P. M. Young and J. C. Doyle, “Computation of μ with real and complex uncertainties,” *Proceedings of the 29th IEEE Conference on Decision and Control*, pp. 1230-1235, 1990.

[YouND1:] P. M. Young, M. P. Newlin, and J. C. Doyle, “μ analysis with real parametric uncertainty,” Proceedings of the 30th IEEE Conference on Decision and Control, pp. 1251-1256, 1991.

[YouND2:] P. M. Young, M. P. Newlin, and J. C. Doyle, “Practical computation of the mixed μ problem,” *Proceedings of the American Control Conference*, pp. 2190-2194, 1992.

5

Control Design via μ Synthesis

The structured singular value, μ, is the appropriate tool for analyzing the robustness (both stability and performance) of a system subjected to structured, LFT perturbations. This is evident from the discussions and examples in Chapter 4. In this section, we cover the mechanics of a controller design methodology based on structured singular value objectives. We rely heavily on the upper bound for μ.

Problem Setting

In order to apply the general structured singular value theory to control system design, the control problem has been recast into the linear fractional transformation (LFT) setting as in Figure 5-1.

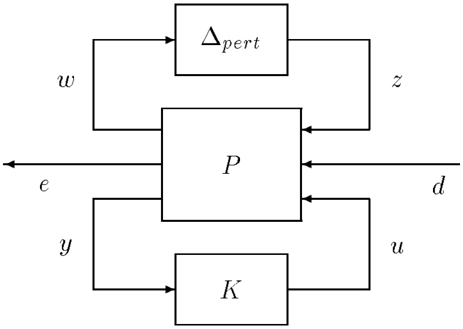


Figure 5-1 LFT Description of Control Problem

The system labeled P is the open-loop interconnection and contains all of the known elements including the nominal plant model and performance and uncertainty weighting functions. The Δ_{pert} block is the uncertain element from the set Δ_{pert}, which parametrizes all of the assumed model uncertainty in the problem. The controller is K. Three sets of inputs enter P: perturbation inputs !w, disturbances d, and controls u. Three sets of outputs are generated: perturbation outputs z, errors e, and measurements y.

The set of systems to be controlled is described by the LFT

$$\{F_U(P, \Delta_{pert}) : \Delta_{pert} \in \Delta_{pert}, \max_{\omega} \bar{\sigma}[\Delta_{pert}(j\omega)] \leq 1\},$$

The design objective is to find a stabilizing controller K, such that for all such perturbations Δ_{pert}, the closed-loop system is stable and satisfies

$$\|F_L[\underbrace{F_U(P,\Delta_{pert})}_{\text{perturbed plant}},K]\|_{\infty} \text{ ð } 1.$$

Observing Figure 5-2, it is clear that

$$F_L[F_U(P,\Delta_{pert})K] = F_U[F_L(P,K),\Delta_{pert}].$$

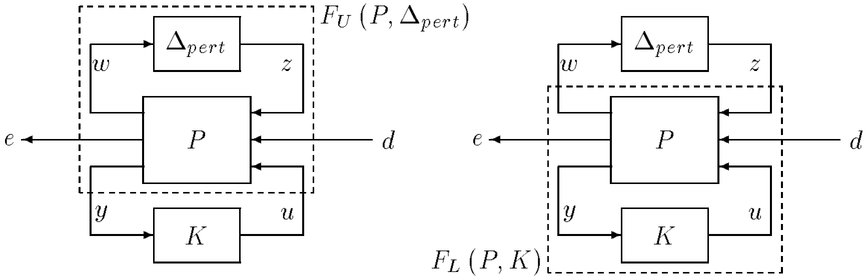


Figure 5-2: Two Different Views of the Closed Loop

Therefore, the design objective is to find a nominally stabilizing controller K, such that for all $\Delta_{pert} \in \mathbf{\Delta}_{pert}$, $\max_{\omega}\bar{\sigma}[\Delta_{pert}(j\omega)] \leq 1$, the closed-loop system is stable and satisfies

$$\|F_U[F_L(P,K),\Delta_{pert}]\|_\infty ð 1.$$

Given any K, this performance objective can be checked utilizing a robust performance test on the linear fractional transformation $F_L(P,K)$. The robust performance test should be computed with respect to an augmented uncertainty structure,

$$\Delta := \left\{ \begin{bmatrix} \Delta_{pert} & 0 \\ 0 & \Delta_F \end{bmatrix} : \Delta_{pert} \in \mathbf{\Delta}_{pert}, \Delta_F \in \mathbf{C}^{n_d \times n_e} \right\}.$$

The structured singular value provides the correct test for robust performance. We know from the discussion in Chapter 4 that K achieves robust performance if and only if

$$\max_{\omega} \mu_\Delta(F_L(P, K)(j\omega)) < 1 \ .$$

The goal of μ *synthesis* is to minimize over all stabilizing controllers K, the peak value of $\mu_\Delta(\cdot)$ of the closed-loop transfer function $F_L(P,K)$. More formally,

$$\min_{\substack{K \\ \text{stabilizing}}} \max_{\omega} \mu_\Delta(F_L(P,K)(j,\omega)) \tag{5-1}$$

This is shown in Figure 5-3.

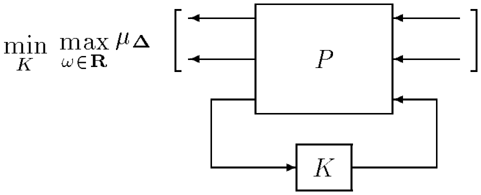

$\min_{K} \max_{\omega \in \mathbf{R}} \mu_{\mathbf{\Delta}}$
P
K

Figure 5-3: μ Synthesis

Replacing μ With Its Upper Bound

For tractability of the μ synthesis problem it is necessary to replace $\mu_\Delta(\cdot)$ with the upper bound. In Chapter 4, we saw that for a constant matrix *M* and an uncertainty structure Δ, an upper bound for $\mu_\Delta(M)$ is an optimally scaled maximum singular value,

$$\mu_\Delta(M) \le \inf_{D \in \mathbf{D}_\Delta} \bar{\sigma}(DMD^{-1})$$

Recall that $\mathbf{D}_{x\Delta}$ is the set of matrices with the property that $D\Delta = \Delta D$ for every $D \in \mathbf{D}_\Delta$, $\Delta \in \Delta$.

Using this upper bound, the optimization in equation Figure 5-1 is reformulated as

$$\min_{\substack{K \\ \text{stabilizing}}} \max_{\omega} \min_{D_\omega \in \mathbf{D}_\Delta} \bar{\sigma}[D_\omega F_L(P,K)(j\omega)D_\omega^{-1}] \tag{5-2}$$

Remember, the *D* minimization is simply an approximation to $\mu[F_L(P,K)(j\omega)]$. D_ω is chosen from the set of scalings, $\mathbf{D}_\Delta$, independently at every ω. Hence, we have

$$\min_{\substack{K \\ \text{stabilizing}}} \min_{D.,D_\omega \in \mathbf{D}_\Delta} \max_{\omega} \bar{\sigma}[D_\omega F_L(P,K)(j\omega)D_\omega^{-1}] \tag{5-3}$$

By $D.,D_\omega \in \mathbf{D}_\Delta$, we mean a frequency-dependent function *D* that satisfies $D_\omega \in \mathbf{D}_\Delta$ for each ω. The general expression $\max_\omega \bar{\sigma}[f(\omega)]$ is notated as $\|f\|_\infty$, giving

$$\min_{\substack{K \\ \text{stabilizing}}} \min_{D.,D_\omega \in \mathbf{D}_\Delta} \left\| DF_L(P,K)D^{-1} \right\|_\infty \tag{5-4}$$

Consider a single matrix $D \in \mathbf{D}_\Delta$, and a complex matrix *M*. Suppose that *U* is a complex matrix with the same structure as *D*, but satisfying $U^*U = UU^* = I$. Each block of *U* is a unitary (orthogonal) matrix. Recall that matrix multiplication by an orthogonal matrix does not affect the maximum singular value, hence

$$\begin{aligned}\bar{\sigma}[(UD)M(UD)^{-1}] &= \bar{\sigma}[UDMD^{-1}U^*] \\ &= \bar{\sigma}(DMD^{-1})\end{aligned}$$

So, replacing D by UD does not affect the upper bound. Using this freedom in the phase of each block of D, we can restrict the frequency-dependent scaling matrix $D\omega$ of equation Figure 5-4 to be a real-rational, stable, minimum-phase transfer function, $\hat{D}(s)$, and not affect the value of the minimum.

Hence the new optimization is

$$\min_{\substack{K \\ \text{stabilizing}}} \quad \min_{\substack{\hat{D}(s) \in \mathbf{D}_{\Delta} \\ \text{stable, min-phase}}} \left\| \hat{D}F_L(P, K)\hat{D}^{-1} \right\|_{\infty} \tag{5-5}$$

This optimization is currently solved by an iterative approach, referred to as *$D - K$ iteration*. A block diagram depicting the optimization is shown in Figure 5-4.

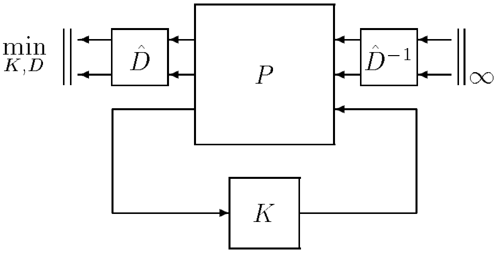


Figure 5-4: Replacing μ with Upper Bound

A specific example clarifies some of the ideas. Assume for simplicity that the uncertainty block Δ_{pert} only has full, unmodeled dynamics (*ie.*, *complex*) blocks, say, N of them. Then the set Δ_{pert} is of the form

$$\Delta_{pert} = \left\{ \text{diag}[\Delta_1, \Delta_2, \ldots, \Delta_N] : \Delta_i \in \mathbf{C}^{r_i \times c_i} \right\} \tag{5-6}$$

The set Δ has the additional fictitious block (for the robust performance characterization)

$$\Delta = \left\{ \operatorname{diag}[\Delta_1, \Delta_2, \ldots, \Delta_N, \Delta_F] : \Delta_i \in \mathbf{C}^{r_i \times c_i}, \Delta_F \in \mathbf{C}^{n_d \times n_e} \right\} \tag{5-7}$$

Therefore, the scaling set $\mathbf{D}_\Delta$ is easily seen to be

$$D_\Delta = \{ \operatorname{diag}[d_1 I, d_2 I, \ldots, d_N I, I] : d_i > 0 \} \tag{5-8}$$

The elements of $\mathbf{D}_\Delta$, which are defined in Figure 5-8 to be real and positive, can be allowed to take on any nonzero complex values and not change the value of the upper bound, $\inf_{D \in \mathbf{D}_\Delta} \bar{\sigma}(DMD^{-1})$. Using this freedom in the phase of each entry of D, we can restrict the frequency-dependent scaling matrix D_ω of equation Figure 5-4 to be a real-rational, stable, minimum-phase transfer function, $\hat{d}(s)$. The optimization is now

$$\min_{K, \hat{d}} \left\| \begin{bmatrix} \hat{d}_1 I & \ldots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \ldots & \hat{d}_N I & 0 \\ 0 & \ldots & 0 & I \end{bmatrix} F_L(P, K) \begin{bmatrix} \hat{d}_1 I & \ldots & 0 & 0 \\ \vdots & \ddots & \vdots & \vdots \\ 0 & \ldots & \hat{d}_N I & 0 \\ 0 & \ldots & 0 & I \end{bmatrix}^{-1} \right\|_\infty$$

D-K Iteration: Holding D Fixed

To solve equation Figure 5-5, first consider holding $D(s)$ fixed at a given, stable, minimum phase, real-rational $\hat{D}(s)$. Then, solve the optimization

$$\min_{\substack{K \\ \text{stabilizing}}} \left\| \hat{D} F_L(P, K) \hat{D}^{-1} \right\|_\infty \tag{5-9}$$

Define P_D to be the system shown in Figure 5-5.

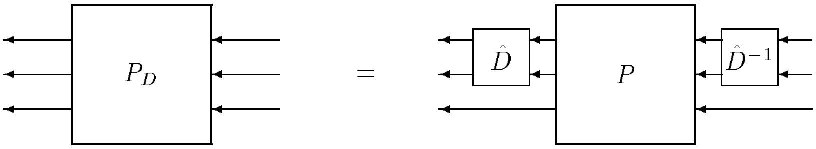


Figure 5-5: Absorbing Rational D Scaling

It is clear that the optimization in equation Figure 5-9 is equivalent to

$$\min_{\substack{K \\ \text{stabilizing}}} \left\| F_L(P_D, K) \right\|_\infty$$

Since P_D is known at this step, this optimization is precisely an H_∞ optimization control problem, as covered in Chapter 3. The solution to the H_∞ problem is well known and involves solving algebraic Riccati equations in terms of a state-space model for P_D.

D-K Iteration: Holding *K* Fixed

With *K* held fixed, the optimization over *D* is carried out in a two-step procedure:

1 Finding the optimal frequency-dependent scaling matrix *D* at a large, but finite set of frequencies (this is the upper bound calculation for μ).

2 Fitting this optimal frequency-dependent scaling with a stable, minimum-phase, real-rational transfer function $\hat{D}$.

The two-step procedure is a viable and reliable approach. The primary reason for its success is the efficiency with which both of the individual steps are carried out. The μ upper bound is based on a convex optimization problem. For this problem, we have developed many heuristics, which when combined with standard convex minimization tools leads to a fast, and accurate computation of the upper bound.The fitting algorithm, using `genphase` and `fitsys`, is based on FFT, least squares and again heuristics. It is extremely fast and works well in most situations.

Two-Step Procedure for Scalar Entries *d* of *D*

We explain the two-step procedure for the case of scalar entries *d* of the *D* scaling matrix, as described earlier. The two-step procedure for full-block *D* scalings is covered in the next section.

A stabilizing controller, *K*(*s*), is given, and the closed-loop $F_L(P,K)$ is formed. At each frequency, we solve the minimization corresponding to the upper bound for μ.

$$\min_{D_\omega \in \mathbf{D}} \bar{\sigma}[D_\omega F_L(P,K)(j\omega)D_\omega^{-1}]$$

This minimization is done over the real, positive *D*ω from the set $\mathbf{D}_\Delta$ defined in equation Figure 5-8. This is carried out with μ, in the upper bound optimization. Recall that the addition of phase to each d_i (ω) does not affect the value of $\bar{\sigma}[D_\omega F_l(P,K)(j\omega)D_\omega^{-1}]$. In other words, the important aspect of the scaling d_i is *its magnitude*, $|d_i(j\omega)|$.

Hence, each positive function, d_i, which is defined on a finite set of frequencies, is fit (in magnitude) by a proper, stable, minimum-phase transfer function, $\hat{d}_i(s)$. This is accomplished as follows: Use the Bode integral formulae to determine the phase $\theta_i(\omega)$ of the stable, minimum-phase function L_i that satisfies

$$|L_i(j\omega)| = d_i(\omega)$$

for all ω. Then, use the transfer function fitting routine `fitsys` to construct a real-rational transfer function $\hat{d}_i(s)$ such that

$$\hat{d}_i(j\omega) \approx \underbrace{e^{j\theta_i(\omega)}}_{\text{phase}} \underbrace{d_i(\omega)}_{\text{magnitude}} = L_i(j\omega)$$

These rational functions are collected together in a diagonal transfer function matrix $\hat{D}(s)$,

$$\hat{D}(s) = \mathrm{diag}[\hat{d}_1(s)I, \hat{d}_2(s)I, \ldots \hat{d}_{F-1}(s)I, I]$$

and absorbed into the original open-loop generalized plant P (to yield P_D, as described earlier).

Two-Step Procedure for Full *D* (Optional Reading)

If there are repeated scalar uncertainty blocks, the corresponding blocks of the D matrices are square, full, positive definite Hermitian matrices. Furthermore, at frequencies where $M(j\omega)$ is a real matrix, the optimal scaling matrix D is also real. Fitting these requires more care than fitting the scalar d scalings described earlier. We describe the approach used in μ-Tools. Suppose that $D : j\mathbf{R} \rightarrow \mathbf{C}^{n \times n}$ is a continuous function, with:

1 $\lim_{\omega \rightarrow \infty} D(j\omega) =: \bar{D} \in \mathbf{R}^{n \times n}$

2 $D(0) \in \mathbf{R}^{n \times n}$

3 $\forall \omega \in \mathbf{R}, D(j\omega) = D^*(j\omega) > 0$

4 $\bar{D} = \bar{D}^T > 0$

Then, under some smoothness assumptions, we do the following. For each 1 ð i ð n:

1 Find a stable minimum phase rational function $\hat{g}_i$ such that for all $\omega \in \mathbf{R}$,

$$\left|\hat{g}_i(j\omega)\right| \approx D_{ii}(j\omega)$$

2 Define a real-valued function $\phi_i : \mathbf{R} \to \mathbf{R}$ such that

$$e^{j\theta_i(\omega)} := \frac{\left|\hat{g}_i(j\omega)\right|}{\hat{g}_i(j\omega)}$$

3 For each $k = 1,2,...,i-1,i+1,...,n$, find rational $\hat{g}_{ik}$ (not necessarily stable or minimum phase) such that for all $\omega \in \mathbf{R}$,

$$\hat{g}_{ik}(jw) \approx e^{j\varnothing_{i(w)}} D_{ik(jw)}$$

Now, put all of these rational functions into a matrix

$$\hat{G}(s) := \begin{bmatrix} \hat{g}_1 & \hat{g}_{12} & \cdots & \hat{g}_{1n} \\ \hat{g}_{21} & \hat{g}_2 & \cdots & \hat{g}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \hat{g}_{n1} & \hat{g}_{n2} & \cdots & \hat{g}_n \end{bmatrix}$$

and define $\Phi := \mathrm{diag}[\phi_i]$. If the rational fitting was done accurately, then, for all ω

$$\hat{G}(j\omega) = e^{j\Phi(\omega)} D(j\omega)$$

Also, $\hat{G}$ has no poles on the imaginary axis, and has nonzero determinant everywhere on the imaginary axis. Hence, by spectral factorization techniques (and the command `extsmp`), we can find a stable, minimum-phase $\hat{D}$ and a unitary matrix function $\hat{U}$ such that

$$\hat{G}(j\omega) = \hat{U}(j\omega)\hat{D}(j\omega)$$

for all ω. The stable, minimum-phase rational matrix $\hat{D}$ is an appropriate scaling for the iteration.

Commands for D – K Iteration

Within μ-Tools there are four ways to perform $D - K$ iteration. They are:

1. Use the graphical user interface `dkitgui` for automated, adjustable, and visual iterations that allow for easy monitoring of progress.

2. Use the script file `dkit` (improved syntax, algorithms and ease-of-use from version 2.0) for automated but adjustable iterations.

3. Use the `dkit` command in the *auto* mode to run a specified number of iterations in an automatic mode (requires no user intervention).

4. Write your own iteration loop, using commands such as `hinfsyn`, `frsp`, `mu`, and `msf`. This approach is not recommended.

Discussion

There are two shortcomings with the $D - K$ iteration control design procedure:

- We have approximated $\mu_\Delta(\cdot)$ by its upper bound. This is not a serious problem since the value of μ and its upper bound are often close.
- The $D - K$ iteration is not guaranteed to converge to a global, or even local minimum [SteD]. This is a very serious problem, and represents the biggest limitation of the design procedure.

In spite of these drawbacks, the $D - K$ iteration control design technique appears to work well on many engineering problems. It has been applied with success to vibration suppression for flexible structures, flight control, chemical process control problems, and acoustic reverberation suppression in enclosures.

At this point, we recommend that you proceed to Chapter 7 for examples of the iteration.

Reference

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

Auto-Fit for Scalings (Optional Reading)

In dkit and dkitgui, there are routines that automatically choose the order of the fit. This is done by comparing the effectiveness of the rational fit. For simplicity of explanation, consider the situation where all of the uncertainty blocks are full, so that the *D*-scaling matrix is made up of several scalar functions, d_i.

Suppose that the *k*th scaling $d_k(\omega)$is fit with an *r*th order, stable, minimum phase function $\hat{d}_k(s) \approx e^{j\theta_k(\omega)} d_k(\omega)$. The suitability of this rational fit is assessed by first defining a scaling matrix

$$D_{k,r}(\omega) := \begin{bmatrix} d_1(\omega) & \dots & 0 & 0 & 0 & \dots & 0 \\ \vdots & \ddots & 0 & 0 & 0 & \dots & 0 \\ 0 & \dots & d_{k-1}(\omega) & 0 & 0 & \dots & 0 \\ 0 & \dots & 0 & \hat{d}_k(j\omega) & 0 & \dots & 0 \\ 0 & \dots & 0 & 0 & d_{k+1}(\omega) & \dots & 0 \\ \vdots & \dots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & \dots & 0 & 0 & 0 & \dots & I \end{bmatrix}$$

Note that in this matrix we have simply replaced the optimal frequency-by-frequency scaling d_k (obtained in the μ upper bound computation) with the frequency response of the rational fit $\hat{d}_k$. Next, we compare the values of

$$\bar{\sigma}[D(\omega)F_L(P,K)(j\omega)D^{-1}(\omega)] \quad \text{and} \quad \bar{\sigma}[D_{[k,r]}(\omega)F_L(P,K)(j\omega)D^{-1}_{[k,r]}(\omega)]$$

If these are close, then *r* is deemed a suitable order for the *k*th scaling function d_k. The measure of closeness can be chosen. Define

$$\beta := \max_{\omega \in \mathbf{R}} \bar{\sigma}[D(\omega)F_L(P,K)(j\omega)D^{-1}(\omega)]$$

Our rules are as follows. For frequencies where

$$\bar{\sigma}[D(\omega)F_L(P, K)(j\omega)D^{-1}(\omega)] \geq 0.5\beta$$

we require that

$$\bar{\sigma}[D_{k,r}(\omega)F_L(P, K)(j\omega)D_{k,r}^{-1}(\omega)] \leq 1.03\bar{\sigma}[D(\omega)F_L(P, K)(j\omega)D^{-1}(\omega)]$$

The quantity 1.03 is the `AutoTol` parameter in `dkit`, which can be modified easily. In `dkitgui`, the `Auto-Fit Tolerance`, which is adjustable in the **Parameter** window, varies this parameter from 1.01 (tight) to 1.06 (loose).

For other frequencies, we require that

$$\bar{\sigma}[D_{k,r}(\omega)F_L(P, K)(j\omega)D_{k,r}^{-1}(\omega)] \leq \bar{\sigma}[D(\omega)F_L(P, K)(j\omega)D^{-1}(\omega)] + 0.1\beta$$

Note that the order of $\hat{d}_k$ is chosen based on its performance as a scaling while all of the other scalings are set to their optimal ($d_k(\omega)$). You can easily modify the constants 0.5 and 0.1 to demand tighter tolerance on the auto-fitting algorithm.

6

Graphical User Interface Tools

This chapter describes three μ-Analysis and Synthesis Toolbox (μ-Tools) graphical user interface tools:

- Workspace tool: `wsgui`
- D – K Iteration tool: `dkitgui`
- Simulation tool: `simgui`

The functionality and application of these tools is illustrated with a multivariable control problem.

Workspace User Interface Tool: wsgui

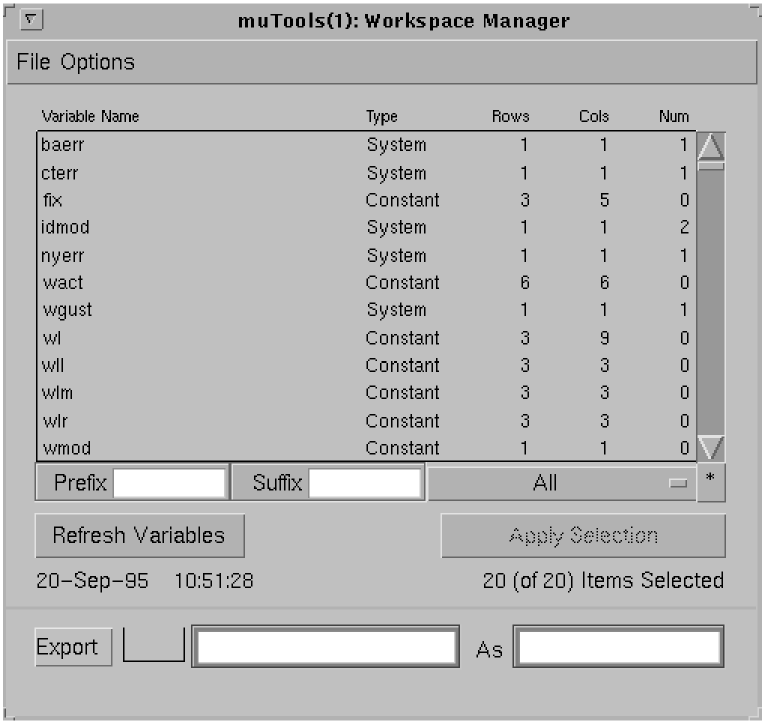
The μ-Tools Workspace Manager, `wsgui`, is used to view variables in the workspace, drag them to other μ-Tools graphical user interface (GUI) drop boxes, and export variables from the μ-Tools user interfaces to the workspace. In this section we only describe how the tool is used to view the workspace variables. In other sections we show how the tool integrates with other μ-Tools GUI tools.

Clear the workspace, create the weighting functions used in the Shuttle example, and start the Workspace Manager.

```
clear; mk_wts; wsgui
```

The **wsgui Workspace Manager** window appears as in Figure 6-1.

Each time **Refresh Variables** is pressed, the MATLAB command `who` is executed, and `minfo` is run to determine the variable type and dimension. This information is displayed in the main scrollable table. The date and time of the last `Refresh` are displayed below the button.



Variable Name	Type	Rows	Cols	Num
baerr	System	1	1	1
cterr	System	1	1	1
fix	Constant	3	5	0
idmod	System	1	1	2
nyerr	System	1	1	1
wact	Constant	6	6	0
wgust	System	1	1	1
wl	Constant	3	9	0
wll	Constant	3	3	0
wlm	Constant	3	3	0
wlr	Constant	3	3	0
wmod	Constant	1	1	0

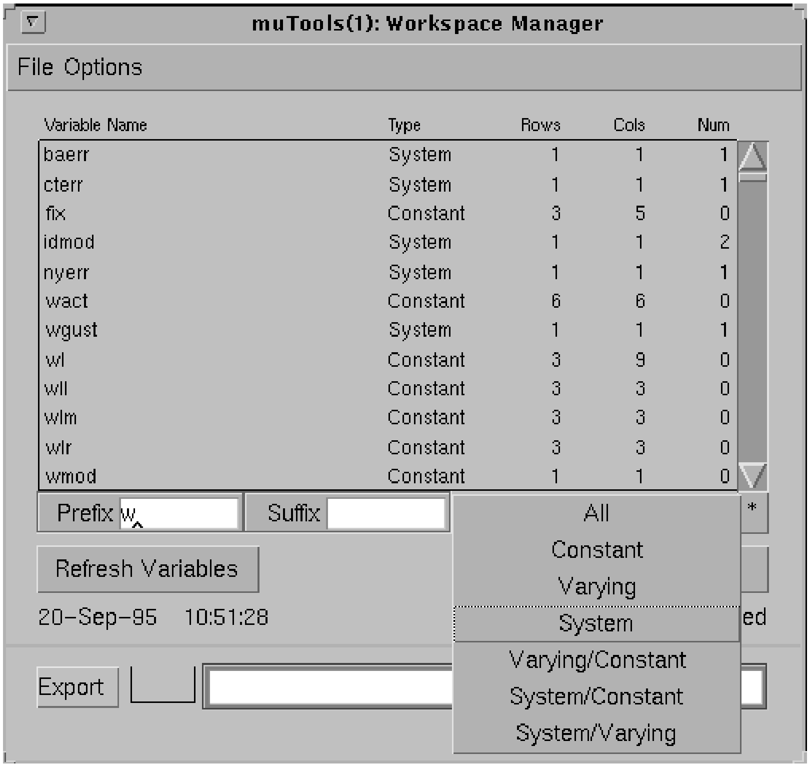
20–Sep–95 10:51:28

20 (of 20) Items Selected

Figure 6-1 Initial wsgui Main Window

The scrollable table can be moved up/down one page by pressing above/below the slider. Pressing the arrows at the end of the slider moves the table one line.

A filter is used to make viewing of a reduced number of selections easy. The `Prefix`, `Suffix`, and matrix type filters are on the bottom of the scrollable table. The matrix type filter is a pop-up menu to the right of `Suffix`. For instance, let's look at SYSTEM matrices whose names begin with `w`. Type a `w` in the `Prefix` box, as shown in Figure 6-2. Note that the **Apply Selection** button becomes enabled, yet all 20 matrices in the workspace are still displayed (even those that don't start with a `w`). The selection filter is only applied when you press the **Apply Selection** button. Move to the right to the pop-up menu, which currently displays `All`, and select `System`, as shown in Figure 6-2. The **Apply Selection** remains enabled, and again, matrices that are not `SYSTEM` matrices are still displayed. Press **Apply Selection** to apply the filter (first letter = w, matrix type = SYSTEM). The scrollable table refreshes, leaving 7 (of the original 20) matrices displayed, as seen in Figure 6-3.

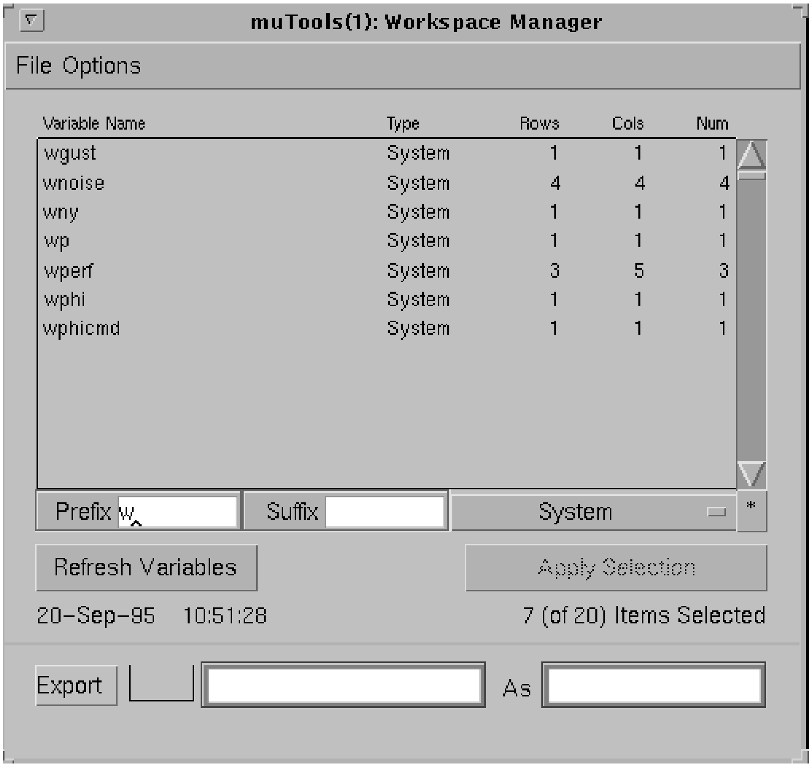


Variable Name	Type	Rows	Cols	Num
baerr	System	1	1	1
cterr	System	1	1	1
fix	Constant	3	5	0
idmod	System	1	1	2
nyerr	System	1	1	1
wact	Constant	6	6	0
wgust	System	1	1	1
wl	Constant	3	9	0
wll	Constant	3	3	0
wlm	Constant	3	3	0
wlr	Constant	3	3	0
wmod	Constant	1	1	0

Prefix w Suffix

20-Sep-95 10:51:28

Figure 6-2: wsgui Main Window with Selected Options



Variable Name	Type	Rows	Cols	Num
wgust	System	1	1	1
wnoise	System	4	4	4
wny	System	1	1	1
wp	System	1	1	1
wperf	System	3	5	3
wphi	System	1	1	1
wphicmd	System	1	1	1

Prefix w

Suffix

System

20-Sep-95 10:51:28

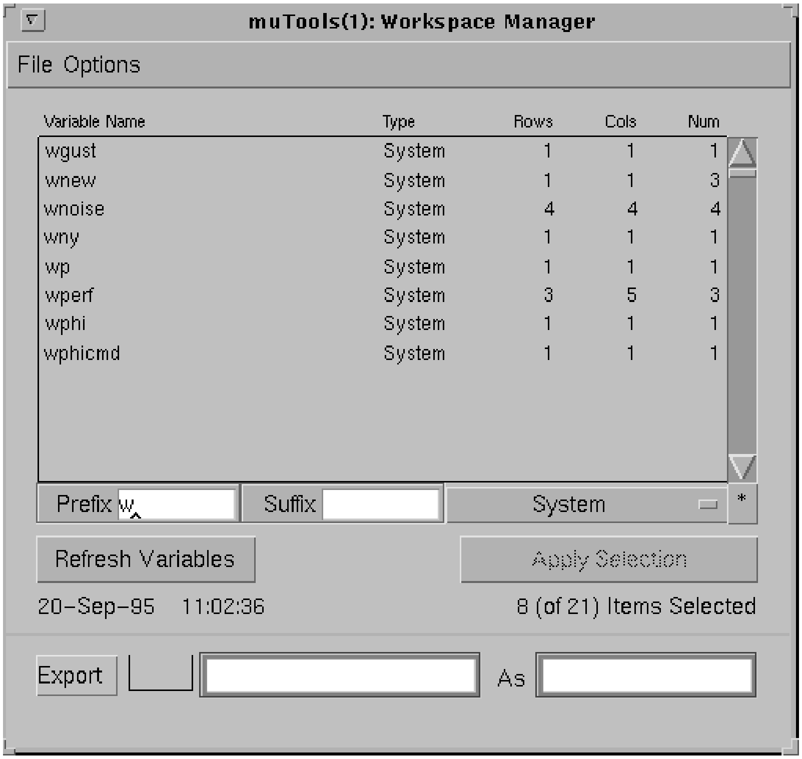
7 (of 20) Items Selected

Figure 6-3: wsgui Main Window After Apply Selection

Now, go into the MATLAB command window, and create a new SYSTEM matrix, with first letter w

```
wnew = nd2sys([1 2],[1 2 3 4]);
```

Note that this does not immediately appear in the scrollable table, even though it satisfies the selection criteria. This new variable will not appear until the **Refresh Variables** button is pushed. Press the **Refresh Variables** button and note that when the table is refilled, the system wnew appears as expected, Figure 6-4.



Variable Name	Type	Rows	Cols	Num
wgust	System	1	1	1
wnew	System	1	1	3
wnoise	System	4	4	4
wny	System	1	1	1
wp	System	1	1	1
wperf	System	3	5	3
wphi	System	1	1	1
wphicmd	System	1	1	1

Prefix w   Suffix   System   *

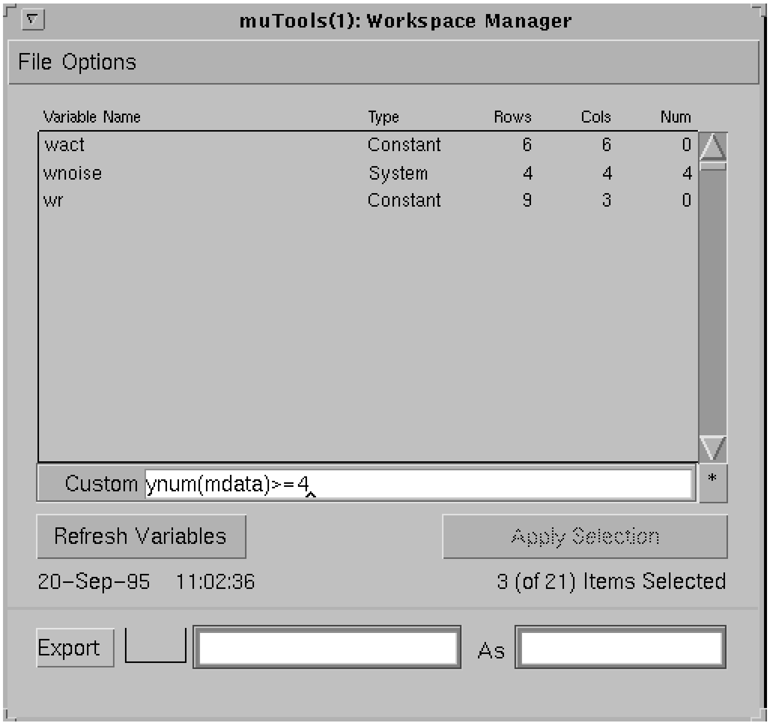
20-Sep-95   11:02:36   8 (of 21) Items Selected

Figure 6-4: wsgui Main Window

Often, you want to create a more complicated selection criteria. The `Custom` filter can be used to do this. Press the push button marked with a * (to the right of the matrix type pop-up menu) to switch to the `Custom` filter. Lets find all matrices with four or more outputs. In the `Custom` box, type

```
ynum(mdata)>=4
```

and press **Apply Selection**. The results are shown in Figure 6-5. This will evaluate the expression in the `Custom` box, and select those workspace variables for which the expression is true. In the `Custom` box, use `mdata` to indicate the matrix's value, and `mname` to substitute the matrix's name. Hence, selections can be based on the name and value of any workspace variable.



Variable Name	Type	Rows	Cols	Num
wact	Constant	6	6	0
wnoise	System	4	4	4
wr	Constant	9	3	0

Custom ynum(mdata)>=4

20-Sep-95 11:02:36 3 (of 21) Items Selected

Figure 6-5: wsgui Main Window with Custom Selection Open

File Menu

The **File** menu at the top of the **Workspace Manager** window has three menu items as seen in Figure 6-6.

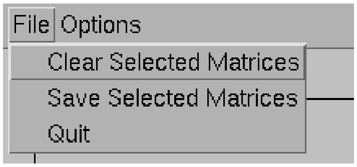
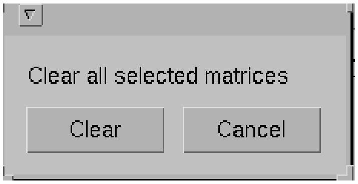


Figure 6-6: File Menu

The `Clear Selected Matrices` item allows you to clear the variable names currently appearing in the **Workspace Manager** window from the MATLAB workspace. Upon selecting the `Clear Selected Matrices` item you are prompted with the box shown in Figure 6-7.

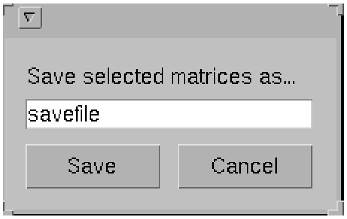


Clear all selected matrices

Figure 6-7: Clear Selected Matrices Box

Pressing the **Clear** push button clears the selected variables from the MATLAB workspace. Pressing **Cancel** cancels this command.

Similarly, the `Save Selected Matrices` item allows you to save the variable names currently appearing in the **Workspace Manager** window to a MATLAB MAT-file. Upon selecting `Save Selected Matrices`, you are prompted with the box shown in Figure 6-8.



Save selected matrices as...

savefile

Figure 6-8: Save Selected Matrices Box

You must enter the name of the file in editable text frame in which to store these variables. The default filename for the variables to be saved into is `savefile`. Pressing the **Save** push button saves the selected variables from the MATLAB workspace to the filename as defined by the editable text string. Pressing **Cancel** cancels this command. The `Quit` item quits the workspace tool and deletes the **Workspace Manager** window.

Options Menu

The **Options** menu at the top of the **Workspace Manager** window has three menu items, as seen in Figure 6-9.

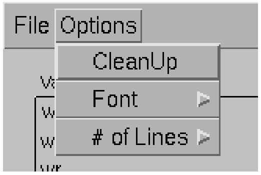


Figure 6-9: Options Menu

The CleanUp item redisplays the variable names and data appearing in the **Workspace Manager** window. The other two menu items are Font and # of Lines. These items correspond to the font type and number of items shown in the **workspace** window. The Font menu item allows you to select a font size of 7 to 12 to display the data (see Figure 6-10). The # of Lines menu item allows you to select the number of lines of data displayed in the workspace window. You can select between [12 20 28 36 44] lines of data. This is especially useful since the **Workspace Manager** window is resizable.

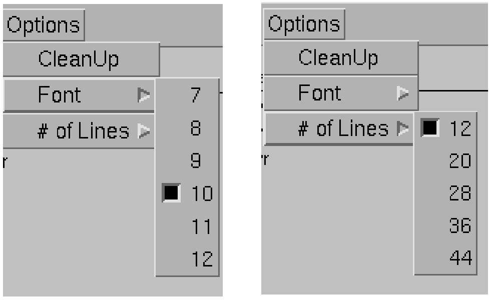


Figure 6-10: Font and # of Lines Menu Items

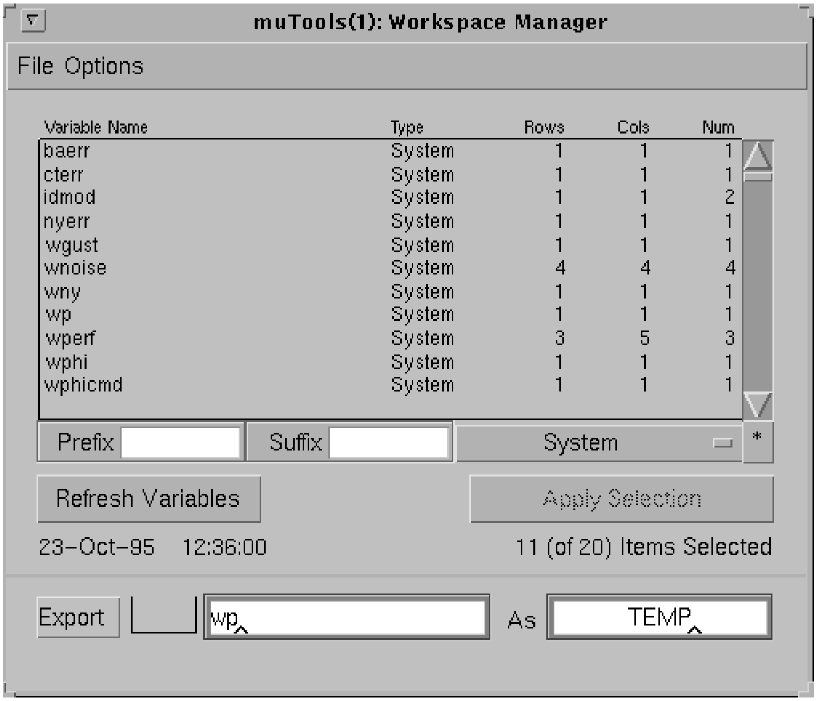
Note You must select the **CleanUp** item from the **Options** menu after resizing.

Export Button

The **Export** button and editable text boxes at the bottom of the **Workspace Manager** window allow you to export data from other μ-Tools user interfaces to the MATLAB workspace. The **Export** button also allows you to copy workspace variables, although it is just as easy to do this at the MATLAB command line.

Consider the following example of how to copy variables. Select all the SYSTEM matrices currently in your MATLAB workspace. To copy the `wp` SYSTEM matrix to the variable `TEMP`, type `wp` in the editable text box to the right of the **Export** button and `TEMP` in the editable text box to the right of `As`. Your **Workspace Manager** window should correspond to Figure 6-11.

Pressing the **Export** button copies `wp` to the variable `TEMP` in your MATLAB workspace. The text display in the message bar shows the MATLAB command and the time and date it was executed. Your **Workspace Manager** window should look like Figure 6-12 after the **Refresh Variables** button is pressed. The drop box to the right of the **Export** button provides another manner to deposit information to be copied into the workspace. For more information about how to use μ-Tools drop boxes see the “Dragging and Dropping Icons” section of this chapter.



Variable Name	Type	Rows	Cols	Num
baerr	System	1	1	1
cterr	System	1	1	1
idmod	System	1	1	2
nyerr	System	1	1	1
wgust	System	1	1	1
wnoise	System	4	4	4
wny	System	1	1	1
wp	System	1	1	1
wperf	System	3	5	3
wphi	System	1	1	1
wphicmd	System	1	1	1

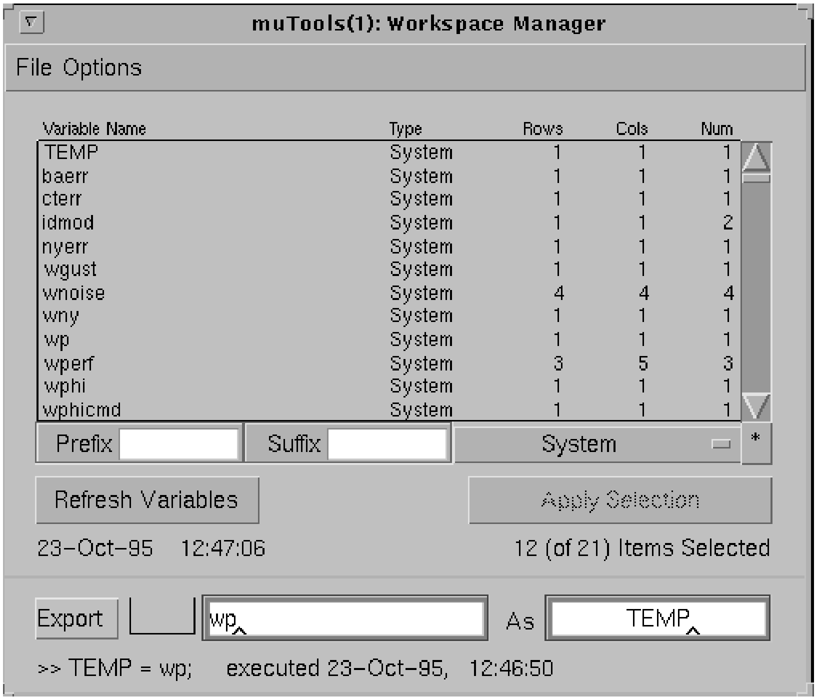
Prefix
Suffix
System

23-Oct-95 12:36:00

11 (of 20) Items Selected

Export wp As TEMP

Figure 6-11: wsgui Main Window with SYSTEMs Selected and Export Data



Variable Name	Type	Rows	Cols	Num
TEMP	System	1	1	1
baerr	System	1	1	1
cterr	System	1	1	1
idmod	System	1	1	2
nyerr	System	1	1	1
wgust	System	1	1	1
wnoise	System	4	4	4
wny	System	1	1	1
wp	System	1	1	1
wperf	System	3	5	3
wphi	System	1	1	1
wphicmd	System	1	1	1

Prefix Suffix System

23-Oct-95 12:47:06 12 (of 21) Items Selected

Export wp As TEMP

```
>> TEMP = wp;     executed 23-Oct-95,   12:46:50
```

Figure 6-12: wsgui Main Window After Refreshing Exported Data

Spinning Satellite Example: Problem Statement

In this section we describe an example application that will be used in the following two sections to demonstrate μ-analysis and synthesis methods using μ-Tools graphical user interfaces.

Consider a satellite spinning about its *z*-axis with a control objective to minimize the x and y-axes rotations due to *x* and *y*-axes disturbances. The control inputs are torque actuators in the *x* and *y*-axis and the feedback measurements are angular sensors. In this example the sensor measurements are poorly aligned with the axis of rotation being measured. The poor alignment of the sensor measurements is introduced on purpose to illustrate the importance of direction in a multivariable control problem.

The following are the linear, state equations of the satellite.

$$\frac{d}{dt}\begin{bmatrix}\Theta_x \\ \Theta_y\end{bmatrix} = \begin{bmatrix}0 & 10 \\ -10 & 0\end{bmatrix}\begin{bmatrix}\Theta_x \\ \Theta_y\end{bmatrix} + \begin{bmatrix}0 & 1 \\ 1 & 0\end{bmatrix}\begin{bmatrix}u_1 \\ u_2\end{bmatrix}$$

$$\begin{bmatrix}y_1 \\ y_2\end{bmatrix} = \begin{bmatrix}1 & 10 \\ -10 & 1\end{bmatrix}\begin{bmatrix}\Theta_x \\ \Theta_y\end{bmatrix}$$

Therefore, the nominal state-space model for the spinning satellite is defined as

$$G = \left[\begin{array}{c|c}A_G & B_G \\ \hline C_G & D_G\end{array}\right] = \left[\begin{array}{cc|cc}0 & 10 & 1 & 0 \\ -10 & 0 & 0 & 1 \\ \hline 1 & 10 & 0 & 0 \\ -10 & 1 & 0 & 0\end{array}\right]$$

The spinning satellite has the following characteristics that need to be included in the control problem formulation:

- The model of the channel 1 actuator has uncertainty or error of 20% at low frequency, below 1 rad/sec. This modeling error reaches 100% uncertainty at 20 rad/sec with very large potential errors in this actuator model above 200 rad/sec. We choose to model this in the μ framework as a multiplicative input uncertainty. A frequency domain weight is constructed to describe the percentage modeling error as a function of frequency. The multiplicative uncertainty weight associated with the first actuator is

$$W_{\mathrm{del1}} := \frac{10(s+4)}{s+200}$$

A frequency response plot of this weight is shown in Figure 6-13.

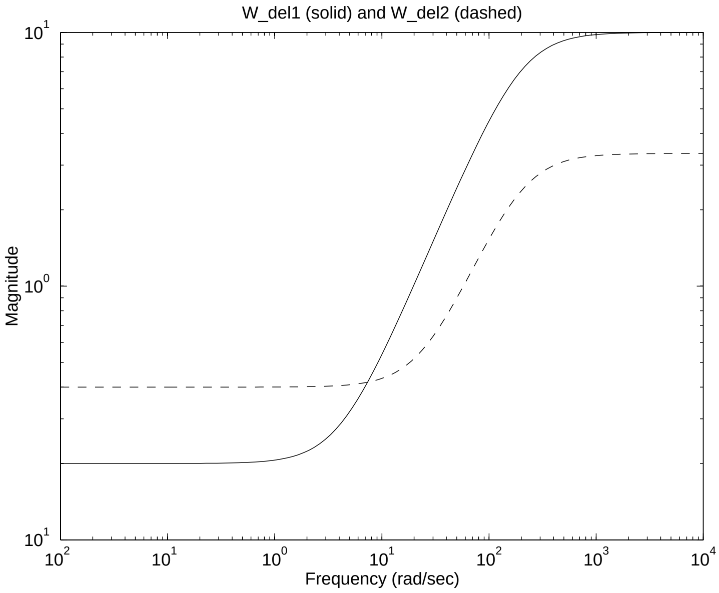


Figure 6-13: Uncertainty Weights: W_{del1} (solid), W_{del2} (dashed)

- The channel 2 actuator has 40% uncertainty at low frequency and the uncertainty in this model reaches 100% at 58 rad/sec. The multiplicative uncertainty weight associated with this actuator is

  $$W_{\mathrm{del2}} := \frac{10(s+24)}{3(s+200)}$$

  A frequency response plot of this weight is shown in Figure 6-13.

- The two sensor measurements are assumed to be noisy. A sensor noise weight, W_n, of the form $W_n = w_n I_{2\times2}$ is used to model the sensor noise. The diagonal structure of the noise weight indicates that there is an equal amount of noise in each measurement. The scalar transfer w_n function is

  $$w_n := \frac{12(s+25)}{5(s+6000)}$$

  Based on this weight at low frequency, below 20 rad/sec, the noise signal has a magnitude of $\pm\, 0.01$radians, at high frequency the noise level reaches a magnitude of $\pm\, 2.4$ radians.

- The desired closed-loop performance is to achieve 100:1 disturbance rejection at DC. This can also be interpreted as desiring 1% tracking error at DC. The performance objective is the same in each channel. Hence it is represented by a diagonal performance weighting function W_p, $W_p = w_p I_{2\times2}$. The scalar transfer function w_p is defined as

  $$w_p = \frac{s+4}{2(s+0.02)}$$

  This weight also defines a desired closed-loop bandwidth of 2 rad/sec and a limit on the peak of the output sensitivity function to a value of 2. A frequency response of the performance weighting functions is shown in Figure 6-14.

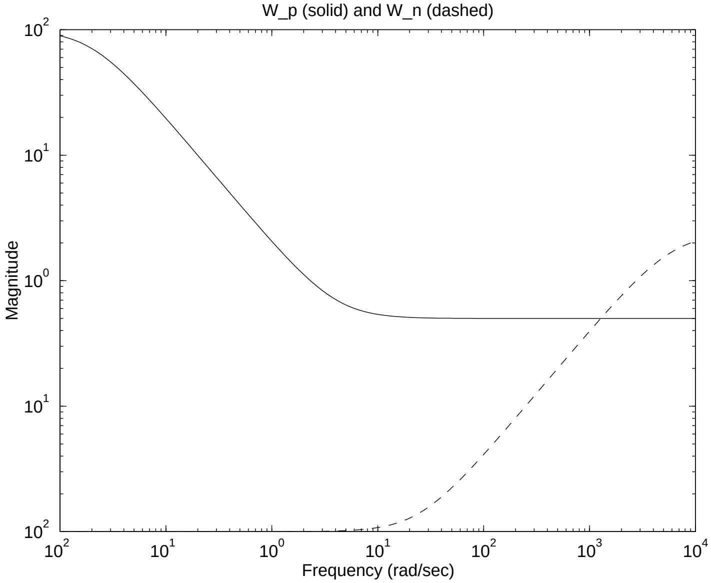


Figure 6-14: Performance Weights: W_p (solid), W_n (dashed)

The control design block diagram for the spinning satellite is shown in Figure 6-15. The uncertainty weights, W_{del1} and W_{del2}, and the performance weights, W_p and W_n, are design parameters that you, the control engineer, can manipulate. These weights are used to incorporate information about the physical system into the control design.

Let P denote this open-loop interconnection. Suppose we order the inputs and outputs in P, as shown in Figure 6-16 (each signal represents a vectored valued signal with two components).

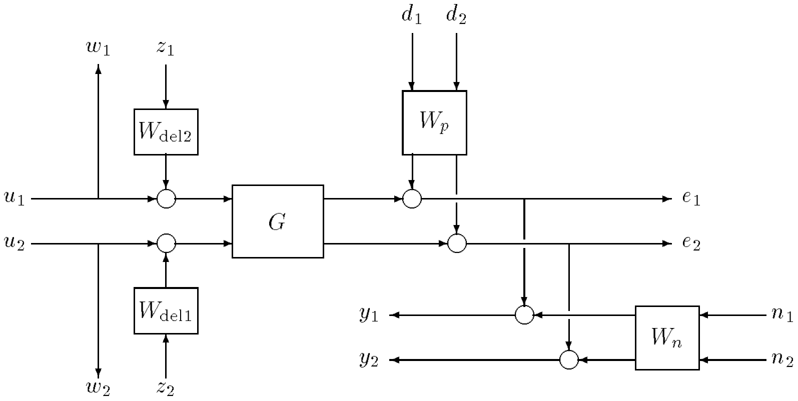


Figure 6-15: Spinning Satellite Interconnection Structure

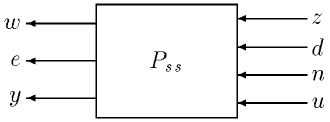


Figure 6-16: Spinning Satellite Control Design System

This is a robust, multivariable feedback control problem. The open-loop interconnection in Figure 6-15 and Figure 6-16 have several features:

- Uncertainty in each Input Channel (z/w)
- Disturbance and Error at Output of `Plant` (d/e)
- Sensor Noise on Measurements (n)

The uncertainty block structure for the problem is

$$\Delta := \{\text{diag}[\delta_1,\delta_2] : \delta_1,\delta_2 \in \mathbf{C}\}$$

To create and load the `P_ss` interconnection structure shown in Figure 6-16, type

```
ssic
```

at the command line. `dkitgui`, a graphical user interface for $D - K$ iteration, is used to design a robust controller that achieves all of these specifications. `simgui` is used to analyze the time response of the resulting controllers.

D-K Iteration User Interface Tool: dkitgui

In this section we describe the graphical user interface for μ-synthesis via $D - K$ iteration, an approach to robust control design. The GUI-tool for $D - K$ iteration is created by the command `dkitgui`. There are five windows associated with this tool:

- **Main Iteration** window, which is the main interface for the user during the iteration.
- **Setup** window, where initial data is entered.
- **Parameter** window, which is occasionally used to modify properties of the $D - K$ iteration, such as H_∞ parameters, and to select the variables that are automatically exported to the workspace each iteration.
- **Frequency Response** window, where the plots of μ and $\bar{\sigma}$ of the closed-loop transfer function matrix are displayed.
- **Scaling** window, where the rational fits of the frequency-dependent *D*-scale data are shown, and can be modified.

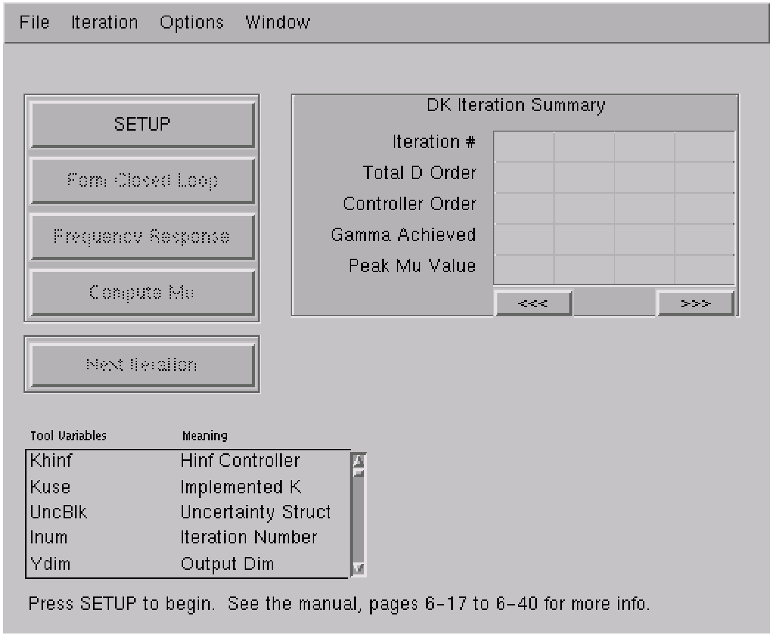
The spinning satellite control design example from the previous section is used to illustrate the basic features of `dkitgui`. Start the tool by typing

```
dkitgui
```

Depending on the computer, `dkitgui` takes up to a minute to start. The `main interactive` window finally displays the message

```
Press SETUP to begin
```

in its lower left corner, as shown in Figure 6-17. This is the location of the message bar where information about the $D - K$ iteration is displayed.



Tool Variables	Meaning
Khinf	Hinf Controller
Kuse	Implemented K
UncBlk	Uncertainty Struct
Inum	Iteration Number
Ydim	Output Dim

Press SETUP to begin. See the manual, pages 6-17 to 6-40 for more info.

Figure 6-17: Initial dkitgui Main Window

Press the **SETUP** button as instructed. This brings the **Setup** window to the foreground. The **Setup** window is shown in Figure 6-18.

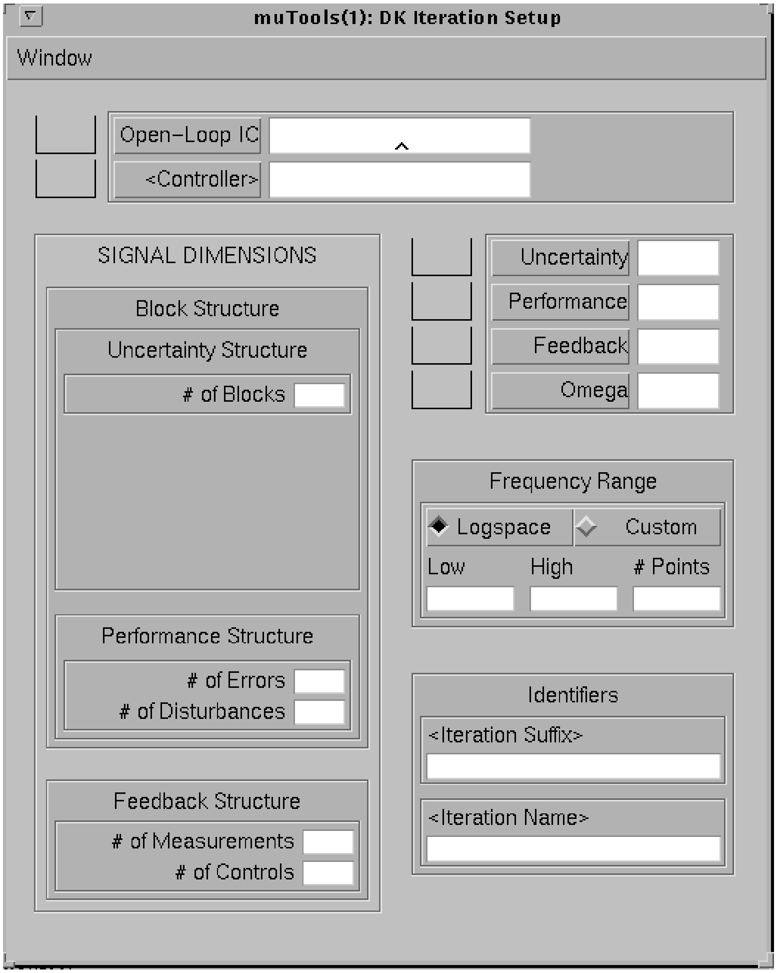


Figure 6-18: Initial dkitgui Setup Window

To start an iteration, you must enter the following items:

- Open-loop interconnection structure
- Number of uncertainty blocks
- Sizes of the uncertainty blocks
- Sizes of the performance transfer functions (errors and disturbances)

- Sizes of the feedback signals (measurements and controls)
- Frequency response range

You can enter this information in any order, except that the dimensions of the uncertainty blocks can only be specified after entering the number of uncertainty blocks. All data entry points are `uicontrol` editable text objects, and operate in a machine dependent manner with which you should be familiar. As before, when we instruct you to enter data in an editable text object, this implicitly means enter the text, and complete the action (by pressing Return, by pressing the mouse on another object, by moving the mouse pointer out of the text object, etc.). See the *MATLAB Function Reference* online for more details on completing a text entry.

Notice that three items in the **Setup** window — `Controller`, `Iteration Suffix`, and `Iteration Name` — are enclosed in < >. This notation denotes a variable that is optional, and no action is necessary.

Enter `P_ss` (the open-loop interconnection) in the open-loop editable text frame (see Figure 6-19). Press the **Open-Loop IC** checkbox to load the data into the `Open-Loop IC` variable. The pointer turns into an hourglass while MATLAB loads the data. Upon loading the data, the pointer turns back into an arrow, and the matrix type (S for System, C for Constant, and V for Varying) of the `Open-Loop IC` variable and its dimensions are displayed to the right of the editable text. In this example the variable `P_ss` is a SYSTEM matrix with six outputs, eight inputs, and eight states. The `<Controller>` data is optional. This allows you to load an initial controller to start the $D - K$ iteration or during the $D - K$ iteration you may want to load a reduced order controller.

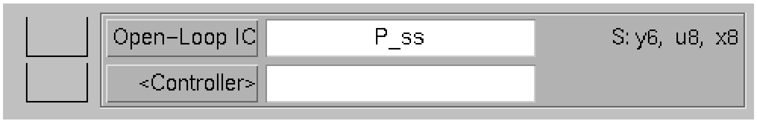
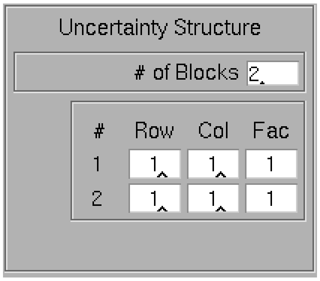


Figure 6-19: Open-Loop/Controller Input Frame

Since there are two uncertain actuators in the spinning satellite problem, the uncertainty structure has 2, 1-by-1 uncertainty blocks. Enter a 2 in the `Uncertainty Structure # of Blocks` text field. This opens a 2-by- 3 editable matrix, as seen in Figure 6-20, where the dimensions of the uncertainty blocks are entered. Enter a 1 for each row and column dimension. The third column, labeled `Fac`, is used for scaling the size of the uncertainty during the iteration.

The value of Fac may be varied from.1 to 10, effectively reducing or increasing the size of the uncertainty by a factor of 10. Leave these factors as 1 for the time being.



Uncertainty Structure

# of Blocks 2

#	Row	Col	Fac
1	1	1	1
2	1	1	1

Figure 6-20: Uncertainty Structure Frame

In the spinning satellite problem (Figure 6-16), there are four exogenous disturbances (two disturbance torques and two measurement noises) and two penalized errors (two tracking errors). Enter a 2 in the **# of Errors** editable text and a 4 into the # of Disturbances editable text in the **Performance Structure** frame. The spinning satellite has two measured variables for feedback and two control actuators. Enter these in the **# of Measurements** and **# of Controls** editable text, respectively, in the **Feedback Structure** frame.

Drop Box Data

In the right column of the **Setup** window are push buttons with the variables names **Uncertainty**, **Performance**, **Feedback**, and **Omega**, as seen in Figure 6-21.

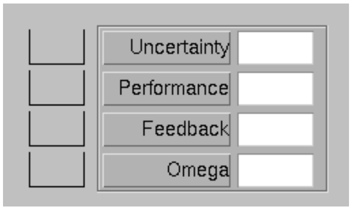


Figure 6-21: Uncertainty, Performance, Feedback, and Omega Frame

Entering data with these push buttons is optional. These push buttons are used to enter the same data as **SIGNAL DIMENSIONS** and **Frequency Range** data. To the left of each push button is a drop box and to the right of each push button is an editable text box. Data that is dropped into the drop boxes or entered in the editable text o*verwrites* the data in the left column of the **Setup** window. (See the "Dragging and Dropping Icons" section at the end of this chapter for more details.) For example, instead of entering a `2` into the **# of Blocks** text and `1`, `1`, `1`, and `1` into each block structure, you could type `ones(2,2)` in the **Uncertainty** editable text and press the **Uncertainty** push button. This action overloads this data into the Uncertainty Structure by redrawing the Uncertainty Structure frame and filling out the corresponding editable text locations. Similarly the **Performance** data overwrites the Performance Structure data, **Feedback** overwrites the Feedback Structure data, and **Omega** overwrites the Frequency Range data.

Completing Problem Setup

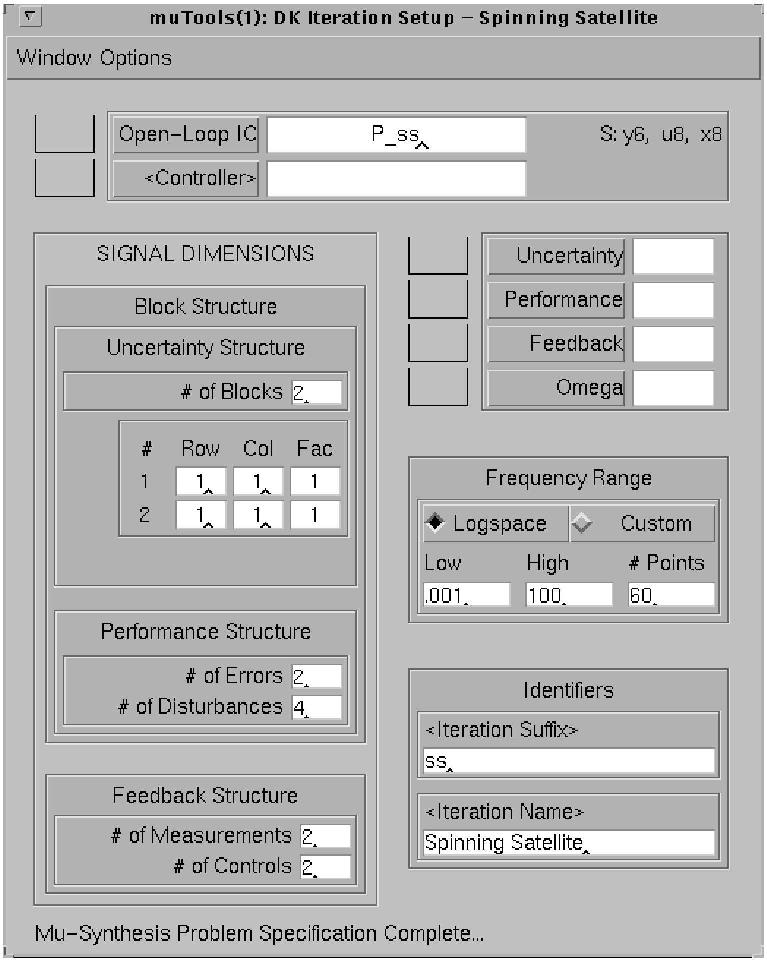
To complete the problem setup, enter the frequency response information appropriate for the problem in the **Frequency Range** frame. Keep the **Logspace** option, and enter 0.001 for the **Low** frequency, 100 for the **High** frequency, and 60 in the **# Points** editable text. Selecting the **Custom** option allows you to enter any valid MATLAB expression in the editable text frame.

This completes the input of the data required by `dkitgui` to start a $D - K$ iteration. Upon correctly entering the required data, the message

```
Mu-Synthesis Problem Specification Complete...
```

appears in the message bar of the Main window.

Enter text in the optional fields **Iteration Suffix** and **Iteration Name**. **Iteration Suffix** is used for data that is automatically exported to the workspace. This string is appended to default names for the variables as they are exported to the workspace. Enter `ss` in the editable text suffix box. We will see the precise effect of this later in the example. The **Iteration Name** modifies the title of each figure. Enter `Spinning Satellite` in the editable text frame, and notice its effect. The **Setup** window should appear as in Figure 6-22.



muTools(1): DK Iteration Setup – Spinning Satellite

Open–Loop IC: P_ss

<Controller>

S: y6, u8, x8

SIGNAL DIMENSIONS

Block Structure

Uncertainty Structure

# of Blocks: 2

#	Row	Col	Fac
1	1	1	1
2	1	1	1

Performance Structure

# of Errors: 2

# of Disturbances: 4

Feedback Structure

# of Measurements: 2

# of Controls: 2

Frequency Range

◆ Logspace ◇ Custom

Low	High	# Points
.001	100	60

Identifiers

<Iteration Suffix>: ss

<Iteration Name>: Spinning Satellite

Mu–Synthesis Problem Specification Complete...

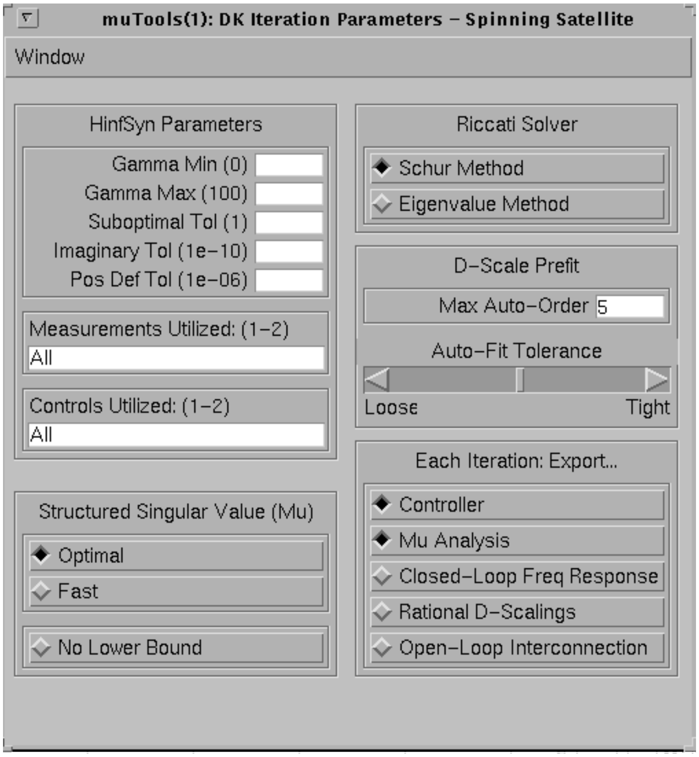
Figure 6-22: dkitgui Setup Window After Entering Data

To continue with the $D-K$ iteration, first hide the **Setup** window. This is done by pulling down the **Window** menu and selecting **Hide Setup** from the menu and returning to the main window. Now select the **Parameters** window from the **Window** menu.

D-K Iteration Parameters Window

The fifth window titled **Parameters**, shown in Figure 6-23, contains additional data you may want to enter for the $D - K$ iteration. This window is shown if **Parameters** is selected from the **Windows** menu. All of the parameters in this window are set to default values. No data entry is required and you can return to the main **Iteration** window to initiate the first iteration. The **Parameters** window contains five settings you can change. They are:

- `HinfSyn Parameters`
- `Structured Singular Value (Mu)`
- `Riccati Solver`
- `D-Scale Prefit`
- `Each Iteration Export`



muTools(1): DK Iteration Parameters – Spinning Satellite

HinfSyn Parameters

Gamma Min (0)

Gamma Max (100)

Suboptimal Tol (1)

Imaginary Tol (1e–10)

Pos Def Tol (1e–06)

Measurements Utilized: (1–2) All

Controls Utilized: (1–2) All

Structured Singular Value (Mu)

◆ Optimal

◇ Fast

◇ No Lower Bound

Riccati Solver

◆ Schur Method

◇ Eigenvalue Method

D–Scale Prefit

Max Auto–Order 5

Auto–Fit Tolerance

Each Iteration: Export...

◆ Controller

◆ Mu Analysis

◇ Closed–Loop Freq Response

◇ Rational D–Scalings

◇ Open–Loop Interconnection

Figure 6-23: D – K Iteration Parameter Window

The **HinfSyn Parameters** and **Riccati Solver** settings, shown in the table below, correspond to the inputs of the H_∞ control design program `hinfsyn`. **Gamma Min** and **Gamma Max** are the minimum and maximum γ values.

The **Suboptimal Tol** is how close to the optimal γ value is desired. The **Imaginary Tol** and **Positive Def Tol** are `epr` and `epp` in the `hinfsyn` program. They correspond to the measure of when the real part of an eigenvalue of the Hamiltonian matrices is zero and determination of the positive definiteness of the X_∞ and Y_∞ solutions. The current default value for each parameter is shown in parentheses, ( ), to the right of each label. The **Riccati Solver** has a mutually exclusive set of buttons for selecting either the Schur or Eigenvalue method to be used to solve the H_∞ Riccati equations.

Hinfsyn Parameters	Riccati Solver
Gamma Min	Schur Method
Gamma Max	Eigenvalue Method
Suboptimal Tol	
Imaginary Tol	
Pos Def Tol	

The **HinfSyn Parameters** frame also allows you to deselect the measurements and controls used during the control design process. For the spinning satellite example there are two controls and two measurements. The **Measurements Utilized** frame indicates that all measurements are currently being used. You can input to the **Measurements Utilized** editable text a standard MATLAB vector to denote the measurements that are to be used. Similarly, you can select in the **Controls Utilized** frame which control inputs are to be used. The resulting control design will have zeros in the state space B or C matrix of the controller corresponding to the measurements inputs or control outputs that have been deselected.

The **Structured Singular Value (Mu)** settings frame has two sets of options. These options correspond to calculation of the structured singular value (μ) using the `mu` program during $D - K$ iteration. The first set of buttons is mutually exclusive. You can either select to use greatest accuracy or a less accurate but faster technique for calculating μ. (The `Optimal` method calls the `mu` program with the option 'c,' Fast method calls `mu` with the 'f' option). You can also select to calculate only an upper bound for μ which calls `mu` with the 'U' option. Selecting this option speeds up the μ calculation, but you will be unable to see how different the upper and lower μ bounds are when they are calculated and plotted in the `Mu/SVD Plot` window.

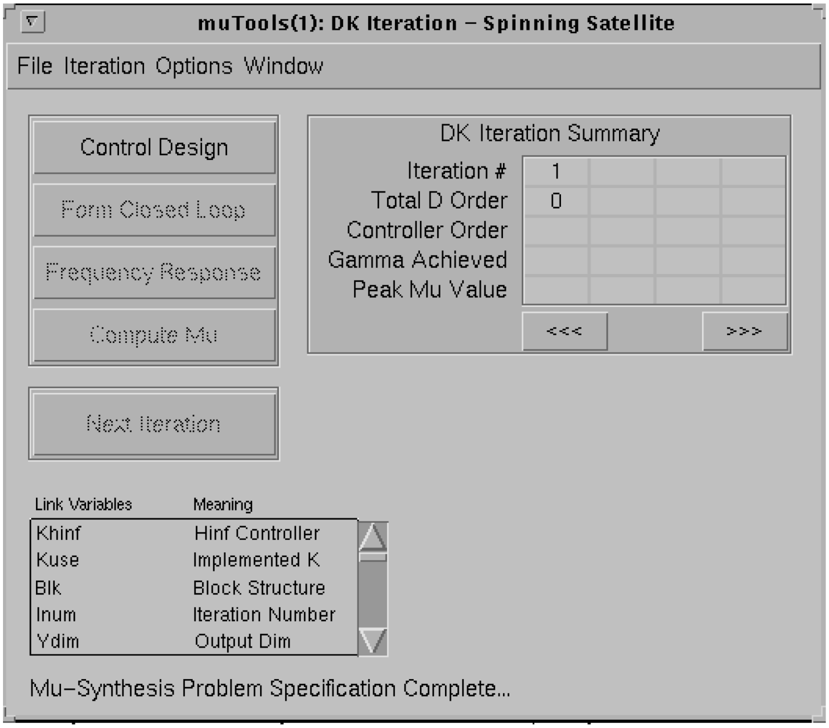
The **D-Scale Prefit** frame contains settings for calculation of the rational *D*-scales. The **Max Auto-Order** defines the maximum *D*-scale state order to be used to fit an individual *D*-scaling during the prefitting part of the *D*-scales fitting routine. The **Max Auto-Order** default is five states. The **Auto-Fit Tolerance** scroll bar allows you to define how *close* the rational scaled μ upper bound is to approximate the actual μ upper bound in a norm sense. Selecting **Loose** will result in lower order *D*-scales being used in the *D*-scale prefitting, whereas selecting **Tight** will likely result in higher order *D*-scales during the *D*-scale prefit computation. This setting can play an important role in determining which minimum of the $D - K$ iteration is achieved. Currently this is done by trial and error.

The Each Iteration: Export... frame shows data in the form of radio buttons that is available to be exported to the MATLAB workspace. In the following list, a subscript i denotes that the integer iteration number is added to the variable's name. These variables are:

- **Controller**, which is exported as K_i.
- **Mu Analysis** exports $\texttt{mubnd}_i$, $\texttt{ddata}_i$, $\texttt{dsens}_i$, and $\texttt{pert}_i$. The $\texttt{mubnd}_i$ variable is selected based on the **Structured Singular Value (Mu)** frame. This is normally the upper and lower bounds for μ. The $\texttt{ddata}_i$ is the *D*-scale data, $\texttt{dsens}_i$ is the sensitivity, and $\texttt{pert}_i$ is the worst-case perturbation as a function of frequency. These are the standard outputs of the `mu` function.
- **Closed-Loop Freq Response** exports the closed-loop system, $\texttt{clp}_i$, and its frequency response $\texttt{clp}_i\texttt{g}$.
- **Rational D-scalings** exports the left, $\texttt{dl}_i$, and right, $\texttt{dr}_i$, rational *D*-scalings.
- **Open-Loop Interconnection** is exported as $\texttt{olic}_i$.

The Iteration Suffix string, which was input in the **Setup** window, is appended to the end of all the output variables selected in the **Each Iteration: Export...** frame. In this example, the **Controller** and the **Mu Analysis** data is selected for output. Therefore after the **Control Design** button executes for the first time, the variable `K1ss` will be in the workspace.

Hide the **Parameter** window and return to the main $D - K$ iteration window to continue this example. This can be done by simply pulling down the **Window** menu and first selecting the **Iteration** option. This moves the main **Iteration** window into the foreground. The main window is now shown in Figure 6-24. Go back to the **Parameter** window, pull-down the **Window** menu, and select **Hide Parameter** to hide the **Parameter** window.



DK Iteration Summary

Iteration #	1			
Total D Order	0			
Controller Order				
Gamma Achieved				
Peak Mu Value				

Link Variables	Meaning
Khinf	Hinf Controller
Kuse	Implemented K
Blk	Block Structure
Inum	Iteration Number
Ydim	Output Dim

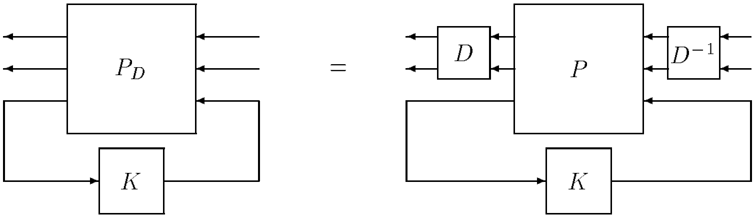
Mu–Synthesis Problem Specification Complete...

Figure 6-24: Main D – K Iteration Window After All the Data Is Specified

D-K Iteration

The main window has (at this point) five significant items:

- Five push buttons, whose actions constitute the $D - K$ iteration. Recall, the $D - K$ iteration pertains to the picture shown below, and is



  - Hold D fixed, design K to minimize H_∞ norm.
  - Then, hold K fixed, and find the new optimal D scalings.
  - Go to Step 1.
- A read-only `DK Iteration` Summary table.
- A menu bar with **File**, **Iteration**, **Options** and **Window** menus.
- A scrollable list of linkable variables that can be dragged from this tool into other tools' drop boxes.
- A message bar.

The scrollable list of linkable variables includes the $D - K$ iteration variables.

Linkable Variables	Meaning
`Khinf`	H_∞ controller
`Kuse`	`Controller K` used in the current $D - K$ iteration.
`Blk`	Block structure for μ calculation
`Ydim`	Output dimension of the controller
`Udim`	Input dimension of the controller
`IC`	Open-loop interconnection structure

Linkable Variables	**Meaning**
`Clpg`	Closed-loop frequency response
`Clpg`	Closed-loop system

These variables can be dragged and dropped at anytime into other μ-Tools user interface commands.

The **Control Design** push button is enabled, and is the first step of the iteration. At this point, the *D* matrices are set to identity (of appropriate dimension). Design the first H_∞ controller by pressing the **Control Design** button. The standard gamma iteration data from `hinfsyn` is displayed in the MATLAB command window. The **DK Iteration Summary** table is updated at the end of the control design, and the **Form Closed-Loop** button is enabled.

The next two steps are simple — form the closed-loop system, and calculate closed-loop frequency response. Press the **Form Closed-Loop** and **Frequency Response** buttons as they are enabled. As the frequency response of the closed-loop system is calculated, a running tab of the number of frequency points calculated is shown in the message frame of the main window. The norm of the closed-loop transfer function as a function of frequency is plotted in the **Frequency Response** window as seen in Figure 6-25. (Note that Figure 6-25 includes the μ plot, which is not accurate at this point in the $D - K$ iteration.) In general, this transfer function has the *D*-scalings from the previous iteration (for the first iteration it's just identity) and the controller which was just designed. Upon completing the frequency response the **Compute Mu** button is enabled.

Now, compute the structured singular value, μ, of the closed-loop frequency response by pressing the **Compute Mu** button. The block structure was defined in the **Setup** window, in the `Block Structure` field. Recall (see Chapter 4) that the upper bound for μ is computed by determining the optimal *D*-scalings as a function of frequency. Like the frequency response, a running tab of the μ calculation is shown in the message frame of the main window. The results are shown in Figure 6-25.

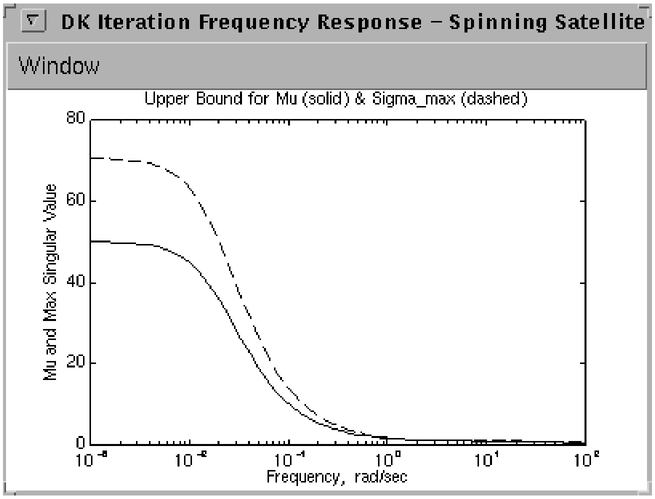
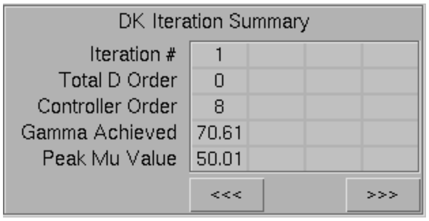


Figure 6-25: μ Upper Bound and Maximum Singular Value for First D – K Iteration

You have now completed one $D - K$ iteration. The `DK Iteration Summary` table is completely updated for the first iteration, as shown in Figure 6-26. The **Next Iteration** button is highlighted. Once the **Next Iteration** button is pushed you cannot effectively return to the previous iteration.

Suppose you desire a more refined frequency response for the μ calculation. You can do this before you move to the second iteration by changing the data in the `Frequency Range` frame in the **Setup** window. Changing the number of frequency response points from 60 to 80 results in the `<Frequency Response>` button being enabled. Note that the sideways carrots < > around the button name denote that you may select the frequency response or go to the next iteration. Continuing to the next iteration without pressing the **Frequency** button will not change any of the data calculated during the first iteration.

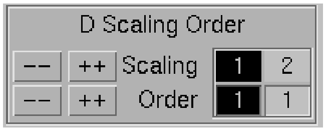


DK Iteration Summary

Iteration #	1			
Total D Order	0			
Controller Order	8			
Gamma Achieved	70.61			
Peak Mu Value	50.01			

Figure 6-26: D – K Iteration Summary After First Iteration

Pressing the **Next Iteration** button results in the *D*-scale data output from the μ calculation being fit with rational *D*-scales. In the spinning satellite example, there are two uncertainty blocks and one performance block, therefore, there are two *D*-scales to be fit. A table entitled **D Scaling Order** appears in the main *D* – *K* iteration window, as shown in Figure 6-27. The table contains the scaling number and the order of each scaling. Each *D*-scale data is prefit with up to a maximum state order transfer function in an attempt to minimize the difference between the scaled μ upper bound and the μ upper bound with the rational *D*-scales. The maximum order of the prefit *D*-scales is specified in the `D-Scale Prefit` field, with the `Max Auto-Order` data in the **Parameters** window.



D Scaling Order

Scaling	1	2
Order	1	1

Figure 6-27: D-Scalings Order Table

The *D*-scaling information for the first *D*-scaling is shown in graphical form in the **Scaling** window, Figure 6-28. Note that the first *D*-scaling was fit with a first order transfer function (see Figure 6-27). There are three plots shown in this window. The top plot shows the μ upper bound, which contains the *D*-scaling data, and a plot of the scaled upper bound. The scaled upper bound is calculated with the rational *D*-scales wrapped in to the original closed-loop frequency response. The middle plot shows the *D*-scale magnitude data and the rational fit for the first *D*-scale. The *D*-scale magnitude data is the variable being fit. The bottom plot shows the sensitivity of the μ upper bound to changes in the *D*-scale. The larger the sensitivity, the more important it is to fit the *D*-scale well in that frequency range.

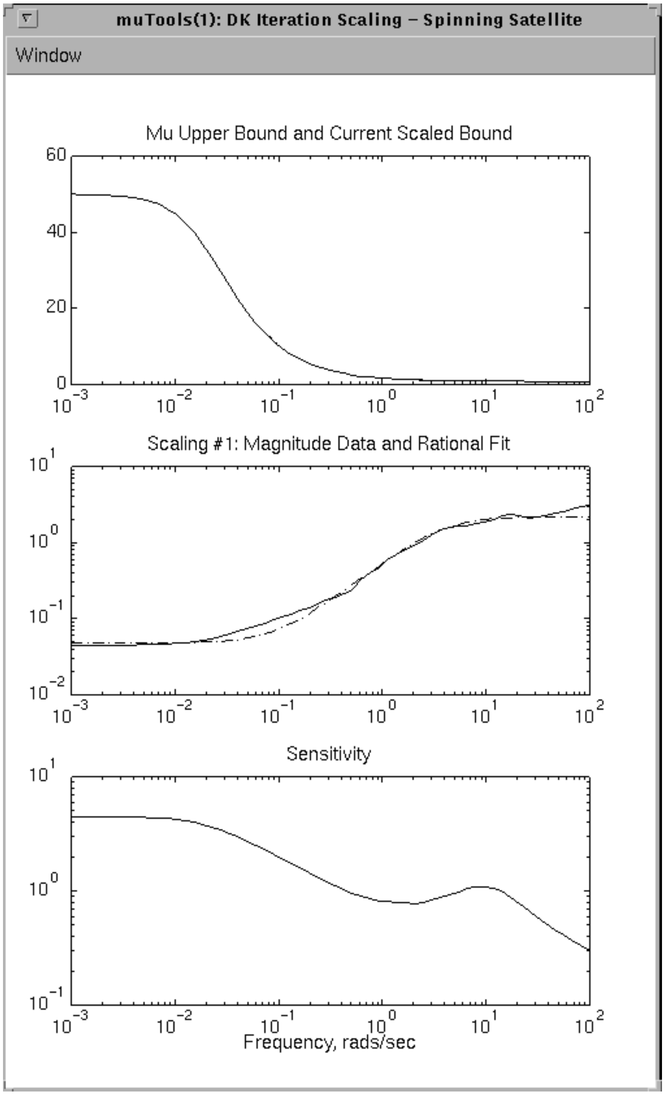

muTools(1): DK Iteration Scaling – Spinning Satellite
Window
Mu Upper Bound and Current Scaled Bound
60
40
20
0
10^-3
10^-2
10^-1
10^0
10^1
10^2
Scaling #1: Magnitude Data and Rational Fit
10^1
10^0
10^-1
10^-2
Sensitivity
Frequency, rads/sec

Figure 6-28: D-Scalings for Second Iteration

You can change which *D*-scale data is shown in the **Scaling** window by pressing the '- -' or '++' buttons to the left of the **Scaling** title. This will cycle through each of the *D*-scalings. The '- -' or '++' buttons to the left of the **Order** title decrement or increment the order of the *D*-scale fit by 1. You can also change the order of the *D*-scale fit by editing the *D*-scale order directly. Changing the *D* fit order will affect the middle plot, which shows the magnitude data (solid line) and the rational fit (dashed line) and will also affect the current scaled upper bound (dashed line) shown in the top figure of the **Scaling** window. Note that the goal of $D - K$ iteration is to reduce the μ upper bound. It is usually important that the current scaled upper bound, which incorporates the rational *D*-scalings, closely matches the calculated μ upper bound. This is especially true in the frequency range where μ is large.

The *D*-scale data for this example are both fit with first order transfer functions. These fits appear to be sufficient, therefore we will go on to the second control design. Pressing the **Control Design** button wraps the rational *D*-scalings into the original interconnection structure, `P_ss`, and designs a new H_∞ controller. For this second $D - K$ iteration, a γ value of 7.89 is achieved.

We now desire to run the next three buttons in sequence. This can be done by pulling down the **Iteration** menubar in the main **Iteration** window. Selecting **Auto Steps**, dragging the mouse to the right of **Auto Steps** allows you to choose the menu item **Next 3 Steps**, as shown in Figure 6-29. This automatically runs the next three steps of the $D - K$ iteration. Selecting either the **Auto Steps** or **Auto Iterate** will result in the appearance of a **Stop** button in the main **Iteration** window below the `DK Iteration` **Summary** table. Pressing the **Stop** button terminates the automated $D - K$ iteration after the current button running has completed.

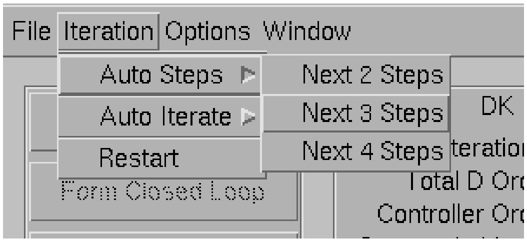
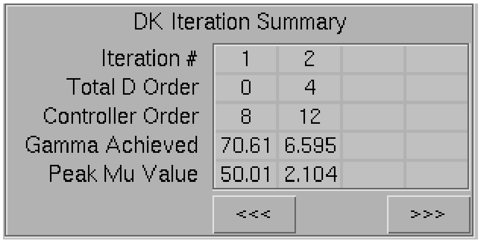


Figure 6-29: Main Window Iteration Pull-Down Menu

Under the **Iteration** menu, the **Restart** option allows you to restart a $D - K$ iteration at the very beginning while leaving the **Setup** window data intact. This is often useful if the incorrect weights were selected and you would like to reload the system interconnection structure and start over.

Note that after two complete *D* – *K* iterations we have achieved a μ value of 2.11 (Figure 6-30). The objective is to achieve a μ value less than 1. Therefore, several more *D* – *K* iterations may be required. User interaction can be eliminated from the *D* – *K* iteration by selecting the **Auto Iterate** menu item from the DK **Iteration** window, Iteration menubar, as shown in Figure 6-31. Dragging the mouse to the right of the `Auto Iterate` menu item allows you to select up from one to eight automated *D* – *K* iterations. During each iteration, the rational *D*-scale order is selected automatically by the `dkitgui` program. Again, the **Stop** button allows you to terminate the **Auto Iterate** option at any time. After three complete *D* – *K* iterations, a μ value of 1.03 is achieved. Selecting one more automated *D* – *K* iteration results in a μ value of 0.91 after four iterations (Figure 6-32).



DK Iteration Summary

Iteration #	1	2		
Total D Order	0	4		
Controller Order	8	12		
Gamma Achieved	70.61	6.595		
Peak Mu Value	50.01	2.104		

Figure 6-30: D – K Iteration Summary After Two Iterations

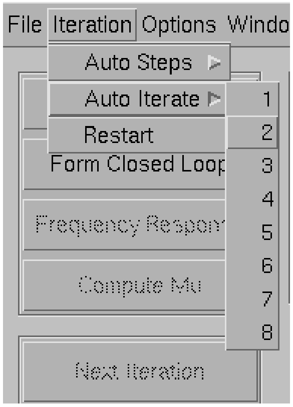
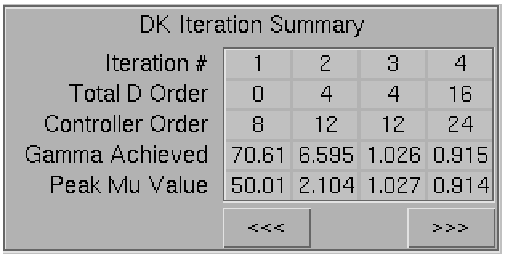


Figure 6-31: Main Window Iteration Pull-Down Menu

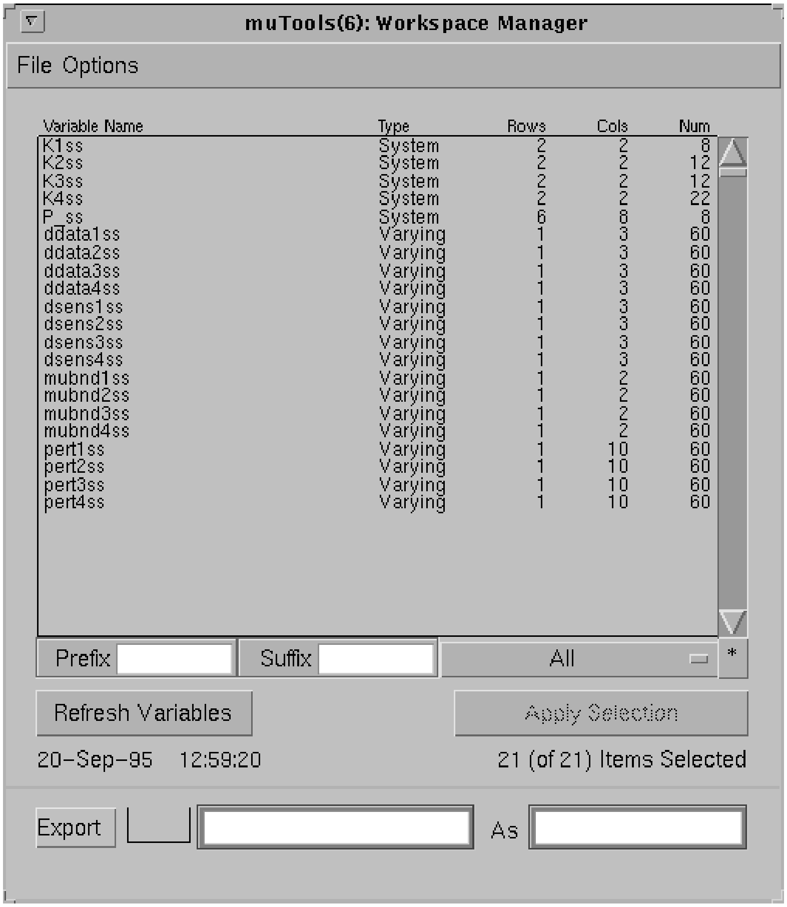


DK Iteration Summary

Iteration #	1	2	3	4
Total D Order	0	4	4	16
Controller Order	8	12	12	24
Gamma Achieved	70.61	6.595	1.026	0.915
Peak Mu Value	50.01	2.104	1.027	0.914

Figure 6-32: D – K Iteration Summary After Four Iterations

Based on the setting of the **Each Iteration Export** radio buttons in the **Parameter** window, the controller designed each iteration has been exported to the MATLAB workspace. Therefore, after four *D* – *K* iterations controllers `K1ss`, `K2ss`, `K3ss`, and `K4ss` are present in your workspace. We also have data from the μ analysis, `ddata`$_i$`ss`, `desns`$_i$`ss`, `mubnd`$_i$`ss`, and `pert`$_i$`ss` in the workspace. If you have your Workspace Manager open, press the **Refresh Variables** button. The results are shown in Figure 6-33.



Variable Name	Type	Rows	Cols	Num
K1ss	System	2	2	8
K2ss	System	2	2	12
K3ss	System	2	2	12
K4ss	System	2	2	22
P_ss	System	6	8	8
ddata1ss	Varying	1	3	60
ddata2ss	Varying	1	3	60
ddata3ss	Varying	1	3	60
ddata4ss	Varying	1	3	60
dsens1ss	Varying	1	3	60
dsens2ss	Varying	1	3	60
dsens3ss	Varying	1	3	60
dsens4ss	Varying	1	3	60
mubnd1ss	Varying	1	2	60
mubnd2ss	Varying	1	2	60
mubnd3ss	Varying	1	2	60
mubnd4ss	Varying	1	2	60
pert1ss	Varying	1	10	60
pert2ss	Varying	1	10	60
pert3ss	Varying	1	10	60
pert4ss	Varying	1	10	60

20–Sep–95 12:59:20

21 (of 21) Items Selected

Figure 6-33: Data in the MATLAB Workspace After Four D – K Iterations

We have successfully designed a controller to achieve our objectives since μ is less than 1. To quit the *D* – *K* iteration graphical user interface, pull down the **File** menu from the main **Iteration** window and highlight the `Quit` option.

Options Menu

The **Options** menu in the main **dkitgui** window allows you to perform two specialized operations. Selecting the **Auto_Refresh K** item from the **Options** menu will refresh the **<Controller>** editable text in the **Setup** window after successful completion of the **Control Design** button. Any valid MATLAB expression can be typed in the **<Controller>** editable text space. This allows you to perhaps have an automated controller reduction scheme that would get executed after the design of the H_∞ controller.

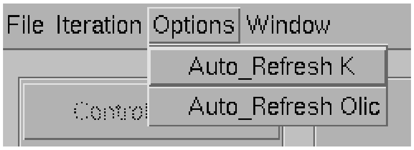


Figure 6-34: D – K Options Menu

Selecting the **Auto_Refresh Olic** item from the **Options** menu will refresh the **Open-Loop IC** editable text in the **Setup** window after successful completion of the fitting the *D*-scale data. Any valid expression can be typed in the `Open-Loop IC` editable text space. This allows you to have an automated program that modifies the open-loop interconnection weightings based on the value of μ from the previous iteration.

LFT Time Simulation User Interface Tool: simgui

The GUI tool for time simulations is created by the command `simgui`. `simgui` has two interface windows and up to six plot windows:

- Main **Simulation Tool** window, which is the main interface for the simulation.
- **Parameter** window, which is used to modify properties of the time simulation, such as the final time, integration step size, initial conditions, and variables automatically exported to the workspace.
- **Plot** windows, where the plots of time responses are displayed. You can open up to six of these windows.

Spinning Satellite Example: Time Simulation

Several controllers for the spinning satellite example have been designed in the previous `dkitgui` section. We will modify the spinning satellite open-loop interconnection shown in Figure 6-15, to include additional outputs for simulation purposes. Let P_{tss} denote this new open-loop interconnection, with the inputs/outputs in P_{tss}, as shown in Figure 6-35 (each signal shown below represents a vectored valued signal with two components).

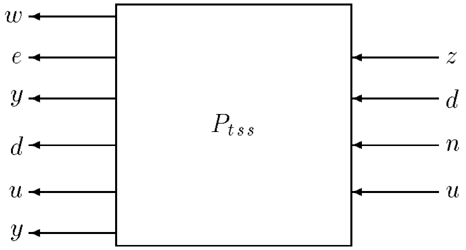


Figure 6-35: Time Simulation Interconnection System: P_{tss}

Two controllers to be analyzed are K1ss and K4ss. K1ss is the controller design from the first $D-K$ iteration and K4ss was designed in the fourth $D-K$ iteration. They are either present in your current workspace or they can be loaded by typing

```
load ss_cont
```

at the command line. The function simgui will be used to analyze the time response of these controllers. Typing the command tssic creates and loads the P_{tss} interconnection structure.

```
tssic
```

Setting Up simgui

Start the linear fractional transformation (LFT) time simulation tool by typing

```
simgui
```

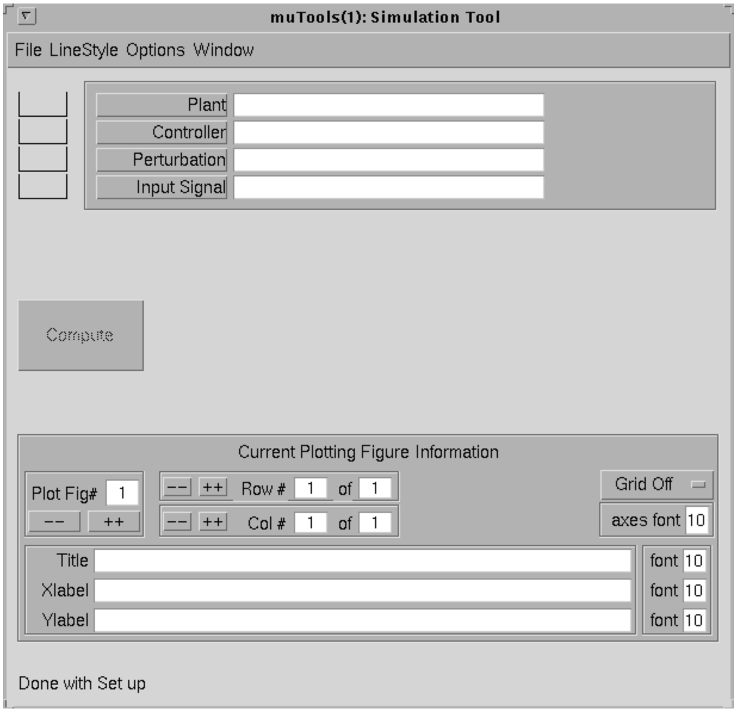
Depending on the computer, this takes up to a minute to start. The message

```
Setting up simulation GUI....
```

appears in the lower left corner of the main window. This is the location of the message bar where information about the time simulation is displayed. Upon completion of the simgui setup, the main window displays the message

```
Done with setup
```

in the message bar, as shown in Figure 6-36.


muTools(1): Simulation Tool
File LineStyle Options Window
Plant
Controller
Perturbation
Input Signal
Compute
Current Plotting Figure Information
--
++
Title
Xlabel
Ylabel

Plot Fig# 1

Row # 1 of 1

Col # 1 of 1

Grid Off

axes font 10

font 10

font 10

font 10

Done with Set up

Figure 6-36: Initial simgui Main Window

The `simgui` tool simulates linear fractional models and plots their responses. `simgui` is based on the standard linear fractional model shown in Figure 6-37.

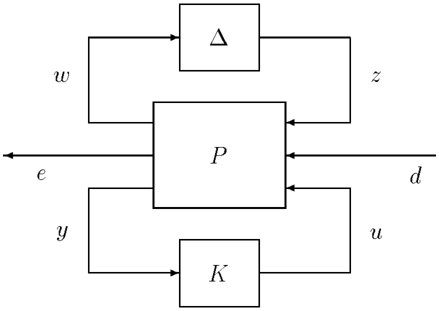


Figure 6-37: Standard Linear Fractional Model

The *P* block corresponds to the open-loop plant interconnection model, referred to as the Plant in simgui. The *K* block corresponds to a state-space controller, Controller in simgui. The Δ block corresponds to the perturbation to the model, Perturbation in simgui. The Input Signal to the time simulation is denoted by *d*. The individual systems are formed using the starp command. The time simulation outputs available for plotting correspond to the variable *e*. Three types of simulation are possible: continuous-time (default) using trsp, discrete-time using dtrsp and sample-data using sdtrsp. In this example, we discuss the continuous-time simulation. This requires the open-loop interconnection, Plant, Controller, and Perturbation to be continuous-time systems, or constants.

To start the time simulation, you must *at least* enter the following items:

- Open-loop plant interconnection model, Plant
- Input excitation time signal, Input Signal

If this is the only data entered, the input dimension of the Plant matrix must match the row dimension of the Input Signal VARYING or CONSTANT matrix. In addition you can enter a:

- Controller
- Perturbation

Note The dimensions of the respective systems and inputs corresponding to those in Figure 6-37 must match one another. If they do not match one another, simgui will display an error message in the message bar.

The Plant, Controller, Perturbation, and Input Signal variables can be entered in any order. For this example enter P_tss (the open-loop interconnection) in the **Plant** text entry, (see Figure 6-36), and press the **Plant** push button to load the plant data. The MATLAB pointer turns into an hourglass while MATLAB loads the data. After the data is loaded, the MATLAB pointer turns back into an arrow, and the Plant data matrix type (S for System, C for Constant, and V for Varying) and its dimensions are displayed to the right of the **Plant** text entry box. Similarly, enter K1ss in the **Controller** text entry and press the **Controller** push button. K1ss was one of the controllers loaded with the load ss_cont command. Enter [0.1 0;0

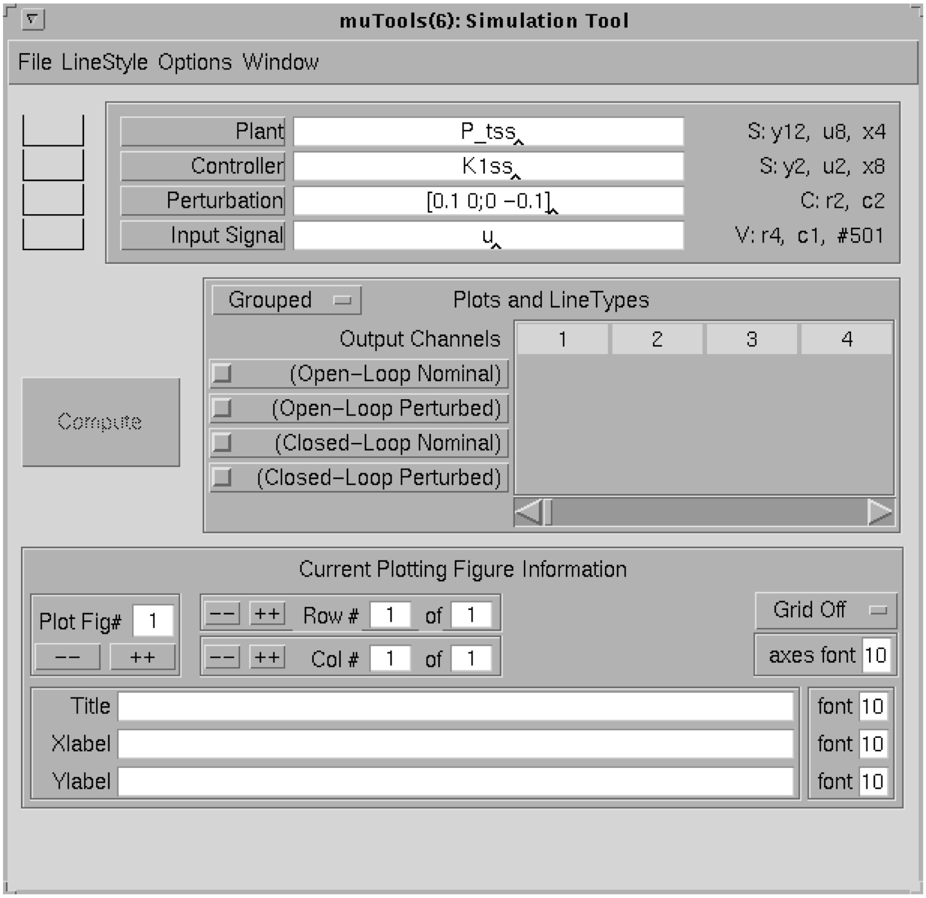
-0.1]) in the **Perturbation** text entry and press the **Perturbation** push button.

Plant, Perturbation, and Controller correspond directly to P, Δ, and K in Figure 6-37. Based on the dimensions of these matrices, the **Input Signal** must have four rows of signals for the system to have the correct dimensions. An error message will be displayed if the dimensions of the interconnected system are incorrect. The resulting closed-loop, perturbed system has a total of eight outputs. They are e, y, d, and u, as shown in Figure 6-35.

Let's create an input signal for the spinning satellite example. The first two inputs are disturbances, and the third and fourth inputs are sensor noise. For this example, input a unit step command into channel 1, zero input to channel 2, and a normally distributed random noise signal into channels 3 and 4 which is scaled by 0.05. Typing the following commands at the MATLAB prompt will generate this input signal.

```
u1 = step_tr(0,1,.01,5);
u2 = mscl(u1,0);
t = getiv(u1);
u34 = siggen('0.05*randn(2,1)',t);
u = abv(u1,u2,u34);
```

Enter u in the **Input Signal** text entry and press its push button. The successful entry of the input signal results in the appearance of the **Plots and Line Types** scroll table in the main simulation window, as shown in Figure 6-38.


muTools(6): Simulation Tool
File LineStyle Options Window
Compute
Grouped
Plots and LineTypes
Output Channels
1
2
3
4
Current Plotting Figure Information
-- ++
Grid Off

Plant P_tss S: y12, u8, x4

Controller K1ss S: y2, u2, x8

Perturbation [0.1 0;0 −0.1] C: r2, c2

Input Signal u V: r4, c1, #501

□ (Open–Loop Nominal)

□ (Open–Loop Perturbed)

□ (Closed–Loop Nominal)

□ (Closed–Loop Perturbed)

Plot Fig# 1

Row # 1 of 1

Col # 1 of 1

axes font 10

Title font 10

Xlabel font 10

Ylabel font 10

Figure 6-38: Main Simulation Window with Data

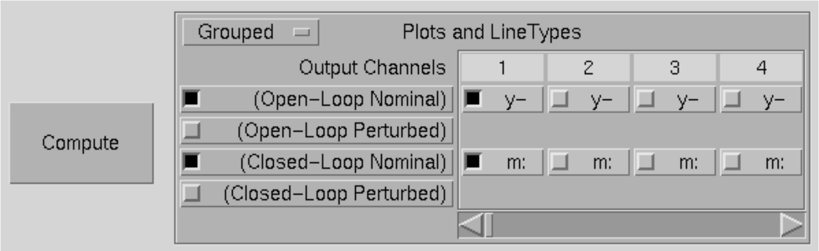
Plots and Line Types Scroll Table

The **Plots and Line Types** scroll table allows you to select the time responses and outputs desired. The four checkboxes on the left side of the scroll table correspond to the four time simulations that can be performed:

- **Open-Loop Nominal**: response of the open-loop plant, `Plant`, without a controller or perturbation included.
- **Open-Loop Perturbed**: response of the open-loop plant, `Plant`, with the perturbation, `Perturbation`, included but no controller.
- **Closed-Loop Nominal**: response of the closed-loop plant, `starp(Plant,Controller)`, without the perturbation included.
- **Closed-Loop Perturbed**: response of the closed-loop plant, `starp(Plant,Controller)`,with the perturbation, `Perturbation`, included.

Currently each checkbox name is enclosed by ( ). This indicates that the time response data for these four is either not current or does not exist. Also each button is not highlighted and no line type data appears in the scroll table. This indicates that currently no time response data is desired. Pressing the **Open-Loop Nominal** and **Closed-Loop Nominal** checkboxes enables the **Compute** button and `Ready to Simulate` appears in the message bar. This action also displays the checkboxes associated with the **Open-Loop** and **Closed-Loop Nominal** responses.

The scroll table contains information corresponding to the eight outputs. The scroll bar at the bottom of the table (see Figure 6-39) allows you to scroll the table to select or deselect output channels. The scroll table contains the four response types and output channels and line type information corresponding to these outputs. Each response and output channel has a corresponding checkbox. The **Plots and Line Types** scroll table also has a pull-down button to denote the **Grouped** `or` **Free Form** operation of the checkboxes (see Figure 6-40). The default is **Grouped**, which means that only one output channel, and all four types of responses, are selected when an output checkbox is pressed. Initially the first output checkboxes of the selected responses are enabled. Therefore, currently in this example, pressing the **Compute** button would result in the first output of the **Open-Loop** and **Closed-Loop Nominal** responses being plotted in the `Plot Page #1`.



Output Channels	1	2	3	4
■ (Open–Loop Nominal)	■ y–	□ y–	□ y–	□ y–
□ (Open–Loop Perturbed)				
■ (Closed–Loop Nominal)	■ m:	□ m:	□ m:	□ m:
□ (Closed–Loop Perturbed)				

Figure 6-39: simgui Plots and Line Types Scroll Table

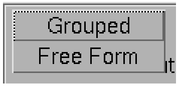


Figure 6-40: Grouped/Free Form Button

Leaving the **Grouped** pull-down menu selected, as seen in Figure 6-40, and pressing the third output of the **Closed-Loop Nominal** response results in the third output of all four responses being selected. Therefore, currently in this example, this would result in the third output of the **Open-Loop** and **Closed-Loop Nominal** responses being plotted in the `Plot Page #1`. The **Free Form** option decouples the responses. Pressing on any output channel checkbox only selects or deselects that checkbox.

The text in the checkbox corresponds to a MATLAB color type, next to a MATLAB line type. For example, the default line types and colors for a color monitor are: **Open-Loop Nominal** outputs are yellow (`y`) and solid lines (–). **Closed-Loop Nominal** outputs are magenta (`m`) and dotted lines (`:`).

The **LineStyle** menu provides a way of modifying the line color and type (Figure 6-41). Selecting **edit** from the **LineStyle** menu changes the checkboxes into `uicontrol` editable text objects, and operates in a machine dependent manner with which you should be familiar. As before, when we instruct you to enter data in an editable text object, this implicitly means enter the text, and complete the action (by pressing **Return**, by pressing the mouse on another object, by moving the mouse out of the text object, etc.). See the online *MATLAB Function Reference* for more details on completing a text entry.

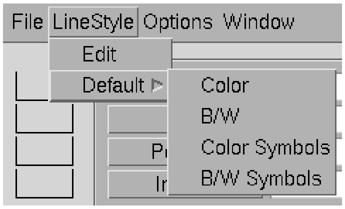
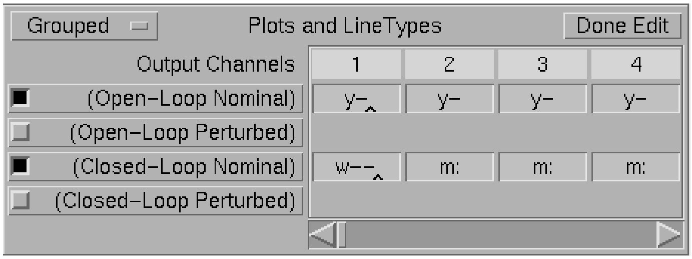


Figure 6-41: LineStyle Pull-Down Menu

You can modify the first output of the **Closed-Loop Nominal** channels to be white (`w`) and dashed (`--`), as shown in Figure 6-42.



Output Channels	1	2	3	4
■ (Open–Loop Nominal)	y–	y–	y–	y–
□ (Open–Loop Perturbed)				
■ (Closed–Loop Nominal)	w––	m:	m:	m:
□ (Closed–Loop Perturbed)				

Figure 6-42: Modified Plots and Line Types Scroll Table

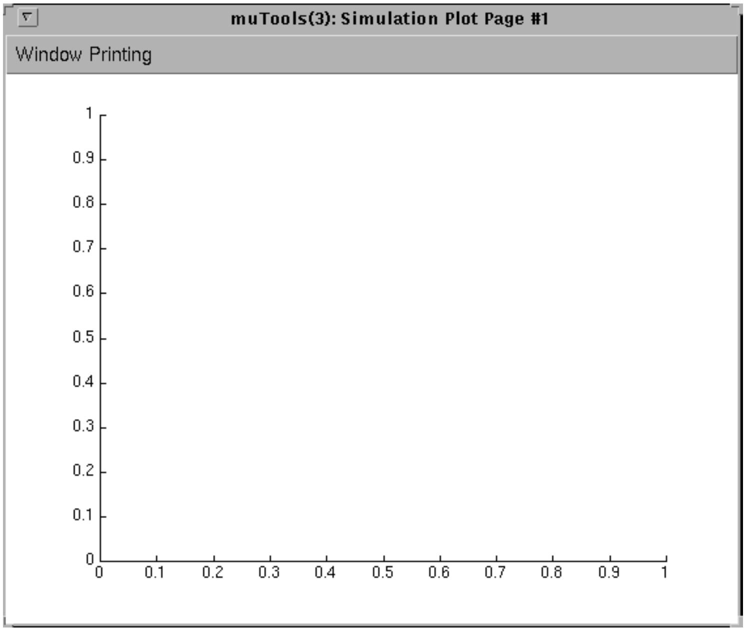
Press the **Done Edit** button in the top right corner of the **Plots and Line Types** table to return the table to checkboxes for output selection. The **Default** selection of the **LineStyle** menu provides four different line type defaults:

- Color
  1. **Open-Loop Nominal** outputs: yellow, solid (`y-`)
  2. **Open-Loop Perturbed** outputs: red, dashed (`r--`)
  3. **Closed-Loop Nominal** outputs: magenta, dotted (`m:`)
  4. **Closed-Loop Perturbed** outputs: white, dashed-dotted (`w-.`)
- B/W (Black and White)
  1. **Open-Loop Nominal** outputs: white, solid (`w-`)
  2. **Open-Loop Perturbed** outputs: white, dashed (`w--`)
  3. **Closed-Loop Nominal** outputs: white, dotted (`w:`)
  4. **Closed-Loop Perturbed** outputs: white, dashed-dotted (`w-.`)

- Color Symbols
  1. **Open-Loop Nominal** outputs: yellow, x (`yx`)
  2. **Open-Loop Perturbed** outputs: red, star (`r*`)
  3. **Closed-Loop Nominal** outputs: magenta, plus (`m+`)
  4. **Closed-Loop Perturbed** outputs: white, circle (`wo`)
- B/W (Black and White) Symbols
  1. **Open-Loop Nominal** outputs: white, x (`wx`)
  2. **Open-Loop Perturbed** outputs: white, star (`w*`)
  3. **Closed-Loop Nominal** outputs: white, plus (`w+`)
  4. **Closed-Loop Perturbed** outputs: white, circle (`wo`)

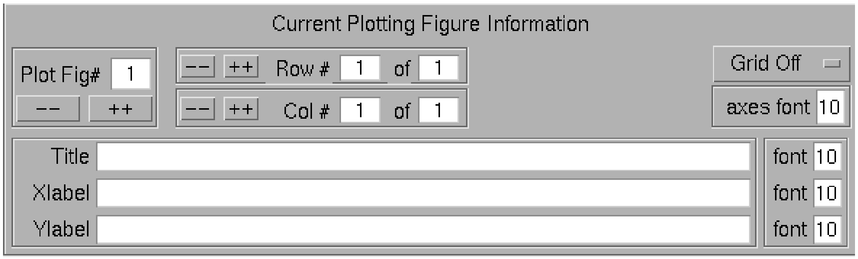
Plotting Window Setup and Titles

The output data selected in the scroll table is displayed in a single plot determined by the information in the **Current Plotting Figure Information** frame, as shown in Figure 6-44. You can format up to six pages (MATLAB plot figures) to display output data. The `simgui` tool initializes to have one plot page (figure) and one set of plot axes on that page, as shown in Figure 6-43. Setup of the plotting pages and titles are controlled via the **Current Plotting Figure Information** frames in Figure 6-44.



muTools(3): Simulation Plot Page #1

Figure 6-43: Simulation Plot Page #1



Current Plotting Figure Information

Plot Fig# 1

Row # 1 of 1

Col # 1 of 1

Grid Off

axes font 10

Title font 10

Xlabel font 10

Ylabel font 10

Figure 6-44: Plotting Data Frames and Titles

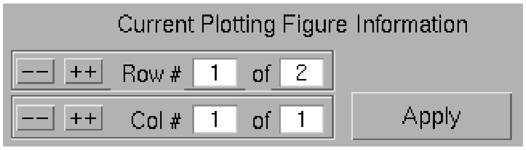
The **Plot Figure** frame, **Plot Fig#**, corresponds to the plotting window number displayed. There can be a maximum of six plotting windows. Pressing the increment **++** and decrement **--** buttons changes the page number and brings the new plot window to the foreground. If the page number is incremented to a page that previously didn't exist, a new page is created with one plot axes. You can also change the **Plot Fig#** editable text to reflect the plot page desired. The input to this editable text must be a positive integer between 1 and 6.

The row number, **Row#** frame, has two editable texts and a decrement -- and increment ++ button as seen in Figure 6-44. The far right editable text, after `of`, corresponds to the number of subplot axes rows desired for the current page (**Plot Fig#**). The first editable text, after **Row#**, indicates the row number of the corresponding subplot on the given **Plot Fig#**. The minimum **Row#** editable text value is 1, and its maximum values correspond to the value shown in the second editable text of **Row#**. The increment and decrement buttons increase or decrease the row number by 1. This is shown in the first **Row#** editable text. The minimum row number is 1 and the maximum is the value of the second **Row#** editable text.

Similarly, the column number, **Col#** frame, has two editable texts and a decrement -- and increment ++ button (see Figure 6-46). The second editable text, after `of`, corresponds to the number of subplot axes columns desired for the current page (**Plot Fig#**). The first editable text, after **Col#**, indicates the column number of the corresponding subplot on the given **Plot Fig#**. The increment and decrement buttons increase or decrease by 1 the column number shown in the first **Col#** editable text. The minimum column number is 1 and the maximum is the value of the second **Col#** editable text.

For example, selecting the second editable text of **Row#** to be 2 and **Col#** to be 3 would result in six subplots, two rows of three columns. Changing either the second editable text of **Row#** or **Col#** will display the result in a push button label **Apply**, appearing in the right of these two frames, as shown in Figure 6-44. The new desired subplot description is applied after this button is pushed. As before, the inputs to the editable text frames must be positive, nonzero integers.

The **Plot Fig#** and the second editable text frames of the **Row#/Col#** frames indicate the layout of the current simulation plot page. The **Plot Fig#**, and first editable text of the **Row#**, and **Col#** frames indicate which plot data is currently in the **Plots and LineTypes** scroll table and their corresponding plot labels and fonts. The first editable text in the **Row#** and **Col#** frames indicate the row and column number of the corresponding subplot in the **Plot Fig#**. The minimum first editable text values of **Row#,** and **Col#** are 1, and their maximum values correspond to the values shown in the second editable text frames in **Row#** and **Col#** respectively. In this example, let's include two plots on the first page, **Plot Fig# 1**, one above the other. Enter a 2 in the # of rows editable text frame and press the **Apply** button, as seen in Figure 6-45.



Current Plotting Figure Information

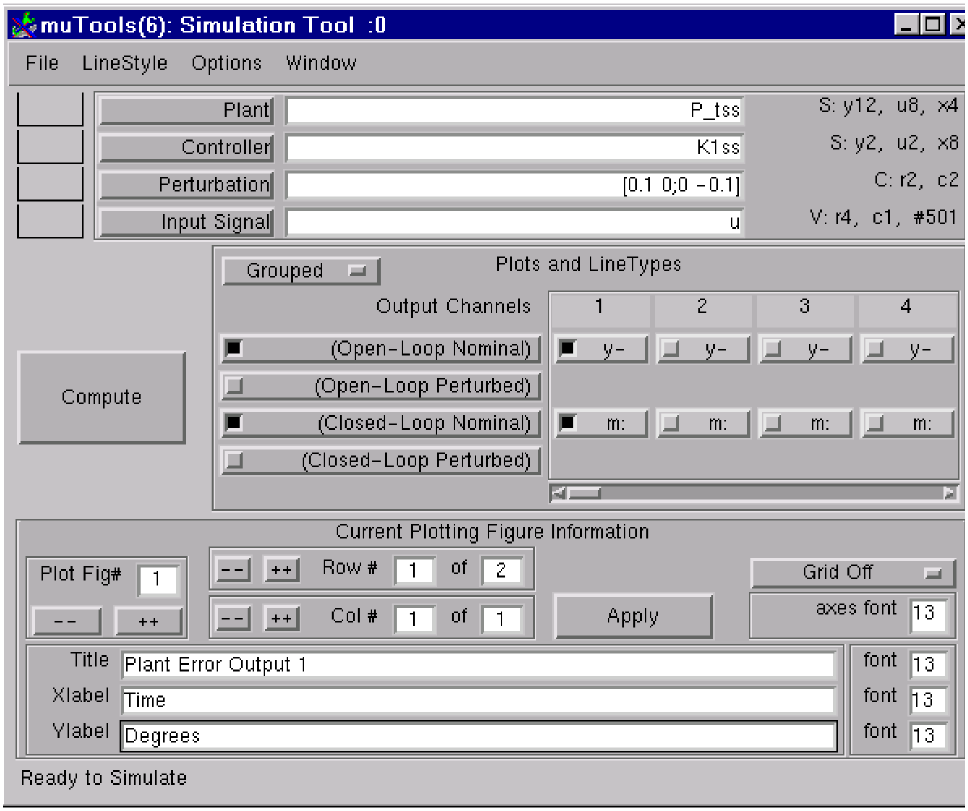
Figure 6-45: Current Plotting Figure Information with Apply Button Enabled

Increment **Row#** to 2 to indicate the bottom plot.

The remaining seven editable text frames and pull down button correspond to labeling and marking the subplot denoted by the data in the **Plot Fig#**, and the first editable text in the **Row#**, and **Col#** frames. The editable text string associated with the **Title** frame is the same as a string entered using the MATLAB `title` command for the given subplot. The `font` size of the title can be entered to the right of the **Title** frame editable text. The default font size for all the labels is 10. Input to the font editable text must be a positive integer. Similarly the **Xlabel** and **Ylabel** editable text strings correspond to the MATLAB `xlabel` and `ylabel` commands. The **axes font** editable text allows you to change the font of the subplot axes fonts. This data is restricted to be a positive, nonzero integer. The button labeled **Grid Off** toggles between **Grid Off** and **Grid On** for the current subplot.

For the top subplot in the current example, enter `Plant Error Output 1` in the **Title** editable text, an *x*-axis label of `Time` and a *y*-axis label of `Degrees`. Select the font size to be 13 for all the text and the axes, grid off. Select output 1 of the **Open-Loop Nominal** and **Closed-Loop Nominal** responses in the **Plots and LineType** scroll table to be plotted.

Selecting an output from the **Plots and LineType** scroll table results in the enabling of the **Compute** push button in the main `simgui` simulation window and the appearance of the string `Ready to Simulate` in the main window message bar. After entering this data, the main simulation window should look like Figure 6-46.



Plant: P_tss

Controller: K1ss

Perturbation: [0.1 0;0 -0.1]

Input Signal: u

S: y12, u8, x4

S: y2, u2, x8

C: r2, c2

V: r4, c1, #501

Grouped

Output Channels	1	2	3	4
■ (Open-Loop Nominal)	■ y-	□ y-	□ y-	□ y-
□ (Open-Loop Perturbed)				
■ (Closed-Loop Nominal)	■ m:	□ m:	□ m:	□ m:
□ (Closed-Loop Perturbed)				

Plot Fig# 1

Row # 1 of 2

Col # 1 of 1

Grid Off

axes font 13

Title: Plant Error Output 1 font 13

Xlabel: Time font 13

Ylabel: Degrees font 13

Ready to Simulate

Figure 6-46: Main Simulation Window with Subplot(1,1,1) Data Entered

Increment the **Row#** to 2, to indicate the bottom plot (subplot 212 of `Plot Page #1`). Enter the title of `Plant Error Output 2`, an *x*-axis label of `Time` and a *y*-axis label of `Degrees`. Select the font size to be 11 for the text and the axes, grid off, and select output 2 **Open-Loop Nominal** and **Closed-Loop Nominal** responses and the second output of these two responses to be plotted in the **Plots and LineType** scroll table. Your first simulation plot page should look the same as Figure 6-47.

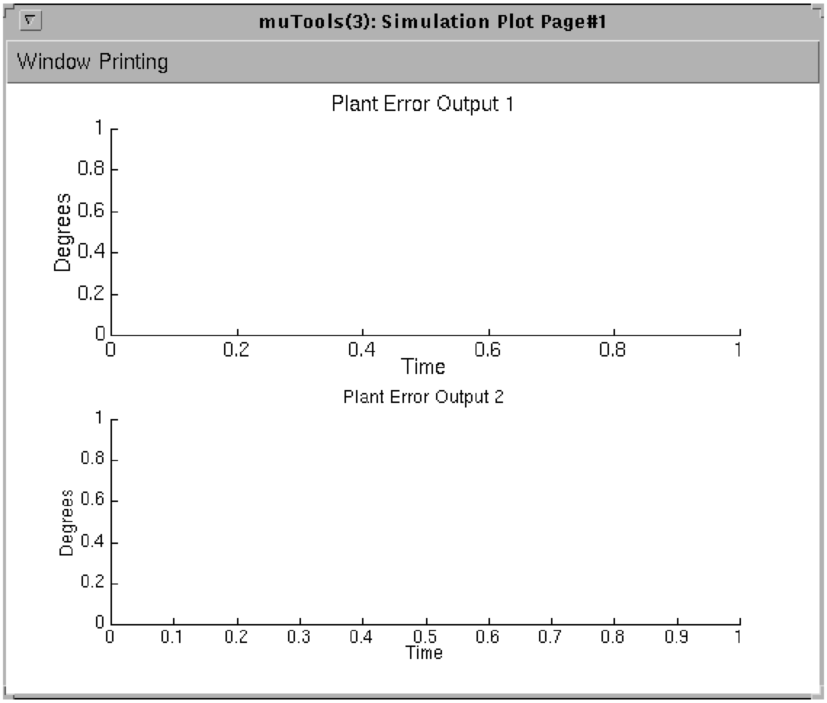
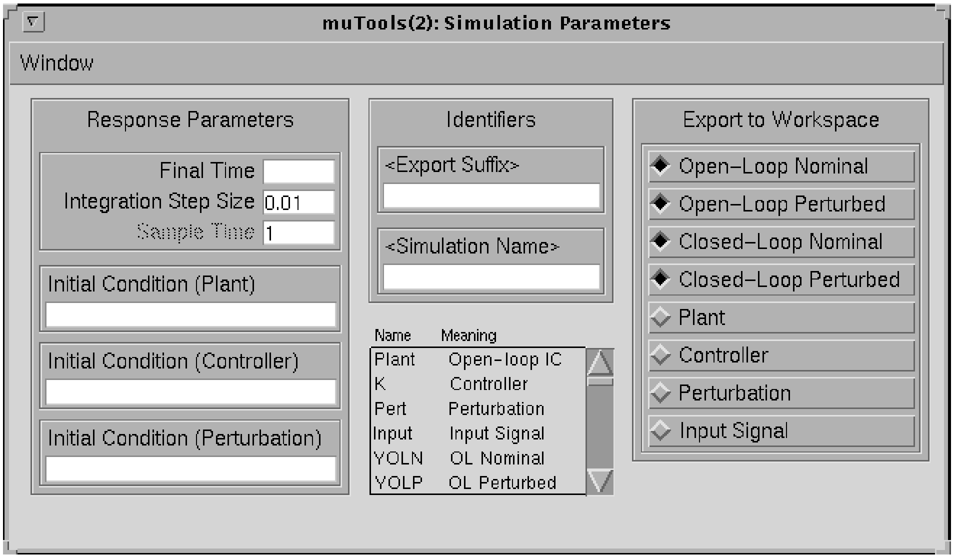


Figure 6-47: First Simulation Plot Page

Increment the **Plot Fig#** to 2. This creates a new page with one plot axes and brings it to the foreground. The **Window** pull-down menu in any **simgui** window shows all the current **simgui** windows and allows you to hide the current window or bring any of the other windows to the foreground. For **Plot Fig#** 2, enter the title of `Closed-loop plant outputs and controls`, an *x*-axis label of `Time` and a *y*-axis label of `Degrees - Newtons`. Select the font size to be 9 for the title, and 10 for all the text and the axes, grid off, and select the **Closed-Loop Nominal** responses. Change the **Grouped** button in the **Plots and LineType** scroll table to `Free Form` and select the *y* and *u* and outputs 3, 4, 5 and 6 be plotted. Change the line types of the output 3 to be white, solid (w-), output 4 to be white, dashed (w--), output 5 to be white, dashed-dotted (w-.), and output to be white, dotted (w:). You have now finished setting up the plotting data and labeling the plots. It's now time to calculate the continuous-time simulation.

Pull down the **Window** menu from any simgui window and select the **Parameters** window. Select an integration time of 0.01 second by entering 0.01 into the **Integration Step Size** editable text, as seen in Figure 6-48.



muTools(2): Simulation Parameters
Window
Response Parameters
Identifiers
Export to Workspace
Initial Condition (Plant)
Initial Condition (Controller)
Initial Condition (Perturbation)
<Export Suffix>
<Simulation Name>

Final Time
Integration Step Size 0.01
Sample Time 1

◆ Open–Loop Nominal
◆ Open–Loop Perturbed
◆ Closed–Loop Nominal
◆ Closed–Loop Perturbed
◇ Plant
◇ Controller
◇ Perturbation
◇ Input Signal

Name	Meaning
Plant	Open-loop IC
K	Controller
Pert	Perturbation
Input	Input Signal
YOLN	OL Nominal
YOLP	OL Perturbed

Figure 6-48: Simulation Parameter Window

Pull down the **Window** menu in the **Parameter** window and hide the **Parameter** window. Press the **Compute** button in the main window to initiate the simulations. Immediately the message Computing appears in the message bar and the label on the **Compute** button changes to **Stop**. By pressing the **Stop** button, the time simulation will terminate at the next available execution of a break. After the Computing message appears in the message bar, and assuming that the **Stop** button was not pressed, one of two pieces of information will appear in the message bar. If each simulation is estimated to take less than 3 minutes to compute then a running tab of simulation as it progresses is shown. If a simulation is estimated to take more than 3 minutes to compute the message, then

```
Simulation will take approx. X seconds:
check Integration Step Size
```

appears in the message bar. The variable X is an estimate of the number of seconds it will take to calculate the given time simulation. After 1 second, this changes into a running tab of simulation as it progresses.

Each time simulation in this example should take less than 3 minutes each to calculate *unless* the default **Integration Step Size** was selected. The results of these simulations are shown in Figure 6-49. Note that the time simulations are calculated for *all* of the system outputs. Therefore other outputs, not initially selected, can be displayed by selecting their respective checkbox *without* having to resimulate the system. Similarly outputs can be deselected without deleting them. You should try to select other outputs, change the color and linetype of the plotted lines, and simulate the open-loop and closed-loop perturbed systems.

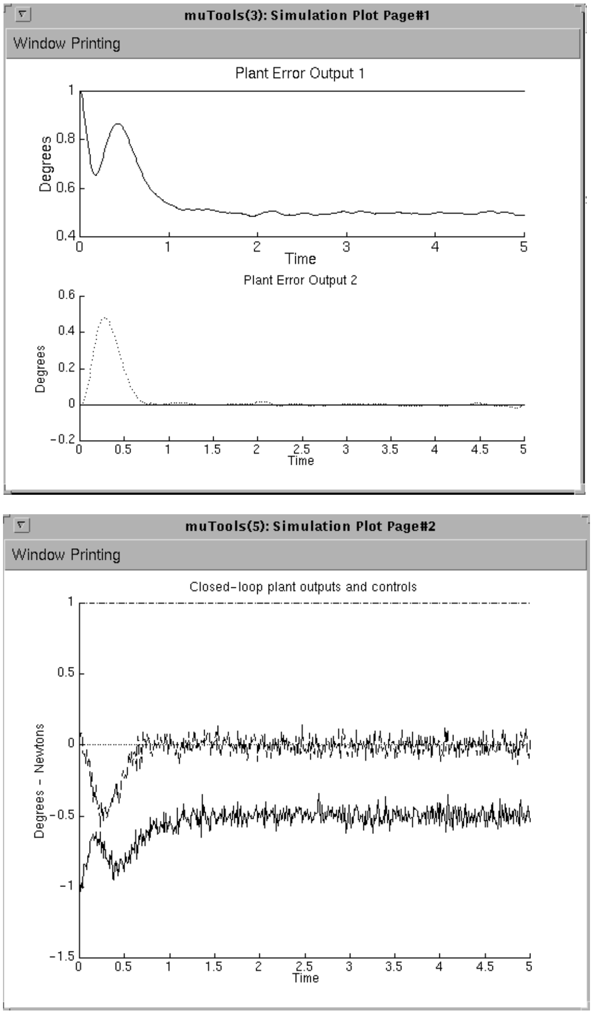


Figure 6-49: Simulation Plot Page 1 and 2 with Time Response

Let's compare these results to those with controller `K4ss` implemented. Return to the **Main Simulation** window and enter `K4ss` in the `Controller` editable text frame. Press the **Controller** push button. This loads `K4ss` into the `Controller` variable and enables the **Compute** button. The previous simulation data with `K1ss` implemented is deleted from the plot windows since it is no longer current. Simulate the open-loop and closed-loop nominal systems by pressing the **Compute** button. The time responses are shown in Figure 6-50. Comparing this response to the response of K1ss (Figure 6-49), you can see the improved performance with K4ss implemented.

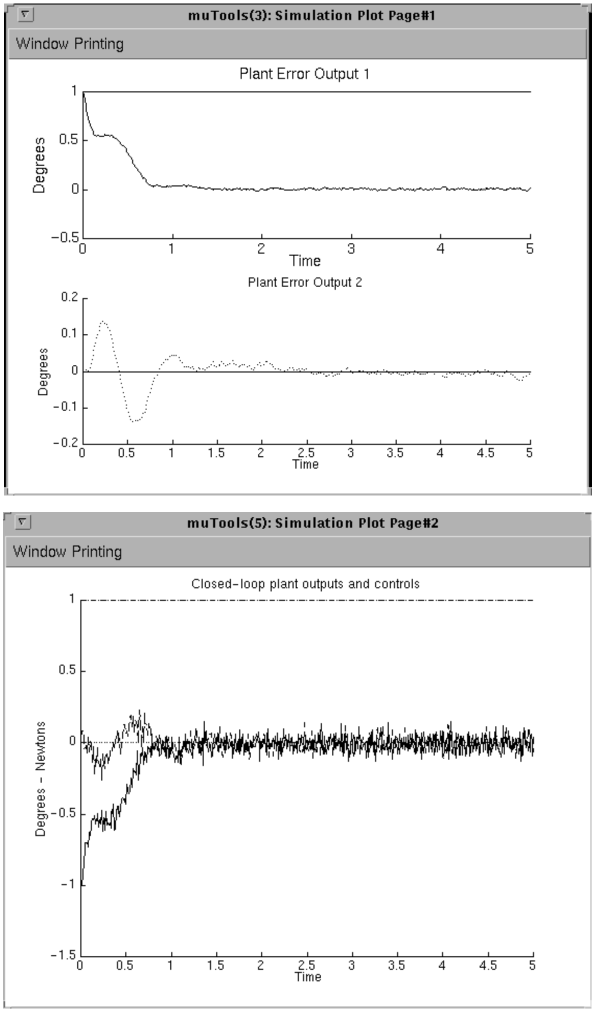


Figure 6-50: Simulation Plot Page 1 and 2 with Time Response

Printing Menu

Each **Simulation Plot Page** has a **Printing** menu for sending that figure to a printer or saving it to a file. Selecting the **Print** option from the **Printing** menu will pop up a **Print** dialog box, as shown in Figure 6-51. This window has three editable text boxes across from **Device**, **Options**, and **Filename**, and a **Print** and **Cancel** button. **Device**, **Options**, and **Filename** correspond to the exact same inputs you would provide to the standard MATLAB `print` command. Therefore, the exact string `-dps` and a filename have to be entered into the **Device** and **Filename** editable text respectively, to output the plot to a postscript file.

Leaving the three editable text boxes empty and pressing the **Print** button will send the current figure to the printer. This is similar to the MATLAB `print` command. Pressing the **Cancel** button will not execute any print command. After filling in any or all of the three editable text boxes, pressing the **Print** button will execute the MATLAB `print` command. Pressing either **Print** or **Cancel** executes that command and hides this window.

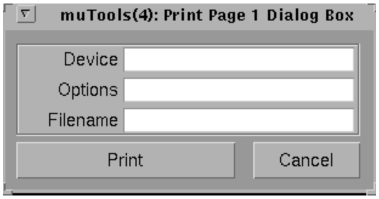


Figure 6-51: Printing Dialog Box

Loading and Saving Plot Information

The **File** menu of the main simulation window contains a **Load Setup**, **Save Setup**, and **Quit** as seen in Figure 6-52.

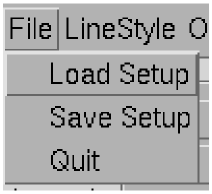


Figure 6-52: File Pull-Down Menu

Selecting the **Load Setup** from the menu will display a **Load Setup** dialog box, as shown in Figure 6-53. The **Load Setup** option is used to load plot setup information that was previously saved using the **Save Setup** option. You can enter a `Variable` name, (the default is `SAVESET`), and a `Filename`. If the `Filename` editable text is left empty, the variable is loaded from the current workspace. Pressing the **Load** button loads the data from the location described by the `Variable` and `Filename` data and hides the window. Pressing **Cancel** hides the window and loads no data. The data loaded includes all the **Plots and LineTypes** `Table` information, plotting information and labels, final time, integration time, sample time, initial conditions, export suffix, and simulation name if available. If an error occurs during the load operation, an error message will appear in the main window message bar.

Selecting **Save Setup** from the **File** menu will display a **Save Setup** dialog box, very similar to Figure 6-53 except with the **Save** button replacing the **Load** button. This option saves all the current plot and line type data along with the labels, final time, integration time, sample time, initial conditions, export suffix, and simulation name, if available, to the **Variable** string name.

The **Variable** and **Filename** editable text and the **Cancel** button operate in the exact same manner as in the **Load Setting** dialog box. Pressing the **Save** button saves the data to the current workspace in the **Variable** editable text string. If this is empty, the default variable name is set to `SAVESET`. The data is saved to the filename defined by the **Filename** editable text, or to the filename `SAVESET` if there is no `Filename` string.

The **Load Setup** and **Save Setup** options are extremely useful. They allow you to customize simulation plots for use with many plants, controllers, perturbations, and input signals.

The **Quit** button of the **File** menu exits the `simgui` tool and deletes all the data and windows opened by `simgui`.

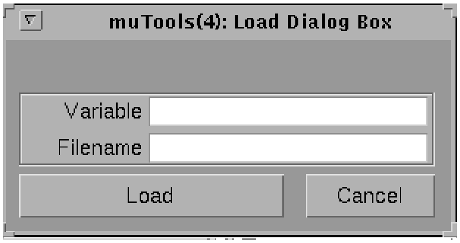


Figure 6-53: Load Setup Dialog Box

Simulation Type

You can select three different time simulations from the **Options** menu on the main window, as seen in Figure 6-54.

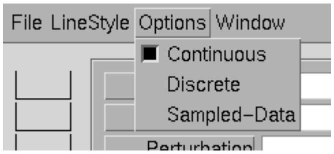


Figure 6-54: Options Pull-Down Menu

The default is continuous-time simulation, which assumes that the `Plant`, `Controller`, and `Perturbation` variables are either continuous-time SYSTEMs, CONSTANTs, or not entered. Continuous-time simulations are performed using the `trsp` command and make use of the `Final Time` and `Integration Step Size` if you have input them in the **Parameter** window. The *default final time* is the *last time data* in the `Input Signal`. An integration step size is calculated by `trsp` if you do not provided one. See `trsp` for more information.

Note The `trsp` command is conservative in its selection of an integration time; therefore, not inputting an `Integration Step Size` may lead to very time-consuming simulations.

Discrete-time simulations assume that the `Plant`, `Controller`, and `Perturbation` variables are either discrete-time SYSTEMs with the same sample rate, CONSTANTs, or not entered. Discrete-time simulations are performed using the `dtrsp` command and make use of the `Final Time` and `Sample Time` if you input them in the **Parameter** window. The default final time is the last time data in the `Input Signal`. The default sample time is 1. See `dtrsp` for more information.

A sample-data time simulation assumes that the `Plant` and `Perturbation` variables are either continuous-time SYSTEMs or CONSTANTs. The `Controller` variable is assumed to be either a discrete-time SYSTEM, CONSTANT, or not entered. Sample-data time simulations are performed using the `sdtrsp` command and make use of the `Final Time`, `Integration Step Size` and `Sample Time` if you input them in the **Parameter** window. The default final time and integration step size are the same as in the continuous-time case. The default sample time is the same as the discrete-time case. See `sdtrsp` for more information.

Progress of all these simulations is shown in the message bar of the main window during their computation.

Simulation Parameter Window

The initial **Parameter** window is shown in Figure 6-48. The **Simulation Data** frame contains data that is used in the calculation of the time responses. The **Final Time** data is used by all the simulations to define the final simulation time and overrides the final time of the **Input Signal** data. The default, an empty **Final Time** editable text string, is the final time of the **Input Signal** data. The `Integration Step Size` data defines the integration step size used by the continuous and sampled-data simulations. The default, an empty `Integration Step Size` editable text string, is to have the `trsp` calculate the integration step size. The **Sampled Time** is the sample rate used by the discrete-time and sampled-data simulations. The default **Sampled Time** is 1. All three variables must be positive, nonzero numbers.

Note that all the variables with editable text to the right of them are encased in < >. This notation denotes that these variables are optional and are not required. The `simgui` program either calculates the value, **Final Time** and **Integration Step Size**, or assigns them a default value.

The `Plant` open-loop plant, **Controller**, and **Perturbation** initial conditions may be entered in their respective editable text boxes. These must be a single, real column vector whose size is the same as the number of states of the respective variable. You can input any workspace variable or valid MATLAB expression to define the initial conditions. The default, with the editable text boxes left empty, is for all the initial conditions to be zero. Incorrect initial condition data will result in an error message in the main window message bar.

The second column of data in the **Parameter** window corresponds to the **Export Suffix**, **Simulation Tool Name** and the linkable variables frame. **Export Suffix** is used for data that is automatically exported to the MATLAB workspace upon computation. This string is appended to default names for the variables as they are exported to the workspace. The **Simulation Tool Name** modifies the title of each `simgui` figure. The linkable variable frame contains a list of `simgui` variables that can be dragged, using the mouse button, to other μ-Tools user interface tools. Table 6-1 contains a list of the linkable variables and their meaning. See "Dragging and Dropping Icons" for more information on how to drag and drop μ-Tools linkage variables.

Table 6-1: Linkable Variables for simgui

Linkable Variable	Meaning
`Plant`	Open-loop interconnection SYSTEM
`K`	The `Controller` variable
`Pert`	The `Perturbation` variable
`Input`	The `Input Signal` variable
`YOLN`	Open-Loop Nominal time response
`YOLP`	Open-Loop Perturbed time response
`YCLN`	Closed-Loop Nominal time response
`YCLP`	Closed-Loop Perturbed time response

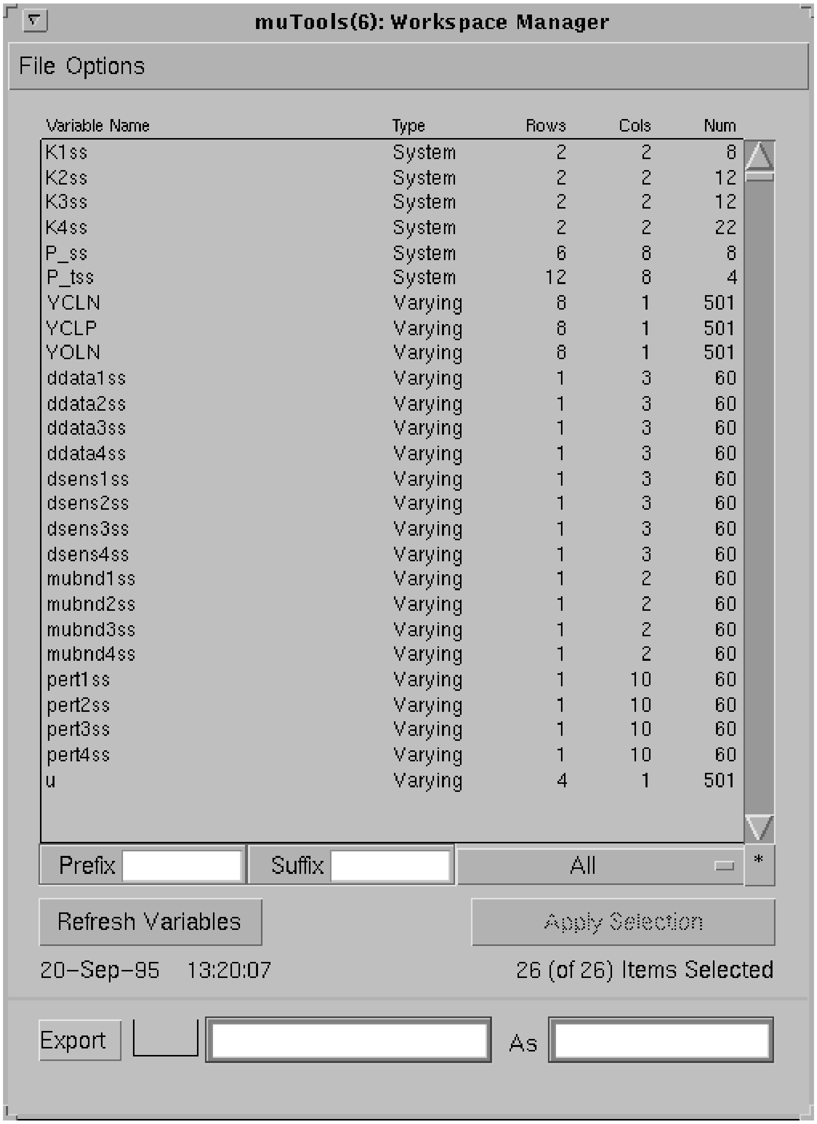
The **Export to Workspace** frame contains variables that you can export to the MATLAB workspace. Each variable is a radio button, which you can select or deselect. If a variable is selected, the variable is exported to the MATLAB workspace each time that simulation is performed. The default setting is to export `YOLN`, `YOLP`, `YCLN`, and `YCLP` every time they are calculated. The **Export to Workspace** names and variables are:

- **Open-Loop Nominal** time response, `YOLN`
- **Open-Loop Perturbed** time response, `YOLP`
- **Closed-Loop Nominal** time response, `YCLN`
- **Closed-Loop Perturbed** time response, `YCLP`
- **Plant**, the open-loop interconnection SYSTEM
- **Controller**
- **Perturbation**
- **Input Signal**

The **Iteration Suffix** string is appended to the end of all the output variables selected. The default is to export the time simulation data `YOLN`, `YOLP`, `YCLN`, and `YCLP` as they are computed.

Note These exported variables are overwritten with their new output responses after the individual time responses are calculated.

Returning to the example, press the **Refresh Variables** button in the Workspace Manager if it is currently open. We can see from the Workspace Manager that the open-loop nominal time response, `YOLN`, and the closed-loop nominal time response, `YCLN`, are saved in the MATLAB workspace (see Figure 6-56). This is based on the setting of the **Export to Workspace** radio buttons. `YOLP` and `YCLP` are not in the MATLAB workspace since the open-loop and closed-loop perturbed responses were not calculated in this example.



Variable Name	Type	Rows	Cols	Num
K1ss	System	2	2	8
K2ss	System	2	2	12
K3ss	System	2	2	12
K4ss	System	2	2	22
P_ss	System	6	8	8
P_tss	System	12	8	4
YCLN	Varying	8	1	501
YCLP	Varying	8	1	501
YOLN	Varying	8	1	501
ddata1ss	Varying	1	3	60
ddata2ss	Varying	1	3	60
ddata3ss	Varying	1	3	60
ddata4ss	Varying	1	3	60
dsens1ss	Varying	1	3	60
dsens2ss	Varying	1	3	60
dsens3ss	Varying	1	3	60
dsens4ss	Varying	1	3	60
mubnd1ss	Varying	1	2	60
mubnd2ss	Varying	1	2	60
mubnd3ss	Varying	1	2	60
mubnd4ss	Varying	1	2	60
pert1ss	Varying	1	10	60
pert2ss	Varying	1	10	60
pert3ss	Varying	1	10	60
pert4ss	Varying	1	10	60
u	Varying	4	1	501

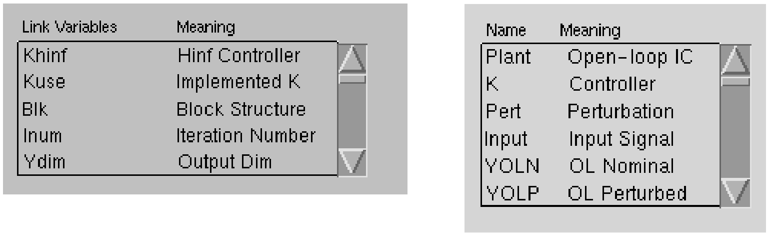
20–Sep–95 13:20:07

26 (of 26) Items Selected

Figure 6-55: Workspace Manager Window

Dragging and Dropping Icons

All the μ-Tools graphical user interface (GUI) programs, wsgui, dkitgui, and simgui have a drag and drop facility. The data icons that can be dragged come from a scroll table of linkable variable names in each program. In wsgui, the MATLAB workspace variable names currently in the main scroll table (see Figure 6-57) are the icons that are dragable. In dkitgui, the scroll table shown in Figure 6-56 contains the linkable variable names that are dragable. The simgui scroll table shown in Figure 6-56 contains the variables from simgui that are dragable. The left column shows the names of the link variables and the right column describes their meaning.

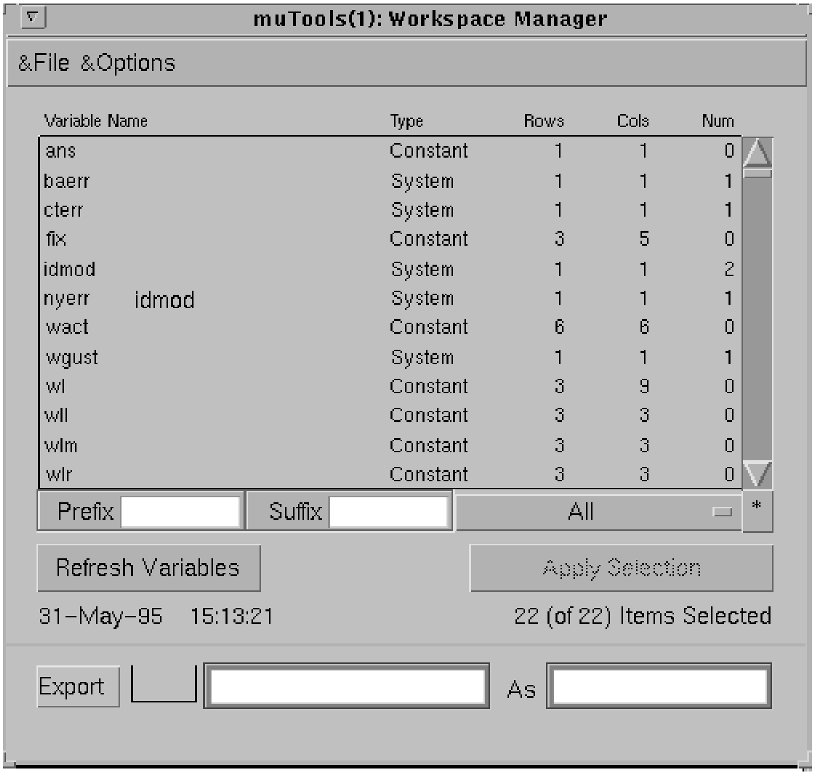


Link Variables	Meaning
Khinf	Hinf Controller
Kuse	Implemented K
Blk	Block Structure
Inum	Iteration Number
Ydim	Output Dim

Name	Meaning
Plant	Open-loop IC
K	Controller
Pert	Perturbation
Input	Input Signal
YOLN	OL Nominal
YOLP	OL Perturbed

Figure 6-56: dkitgui (left) and smgui (right) Linkable Variables Icon Tables for Dragging

To drag an icon, which is a variable name, position the mouse arrow over the variable name to be dragged. Press the left mouse button and hold the button down. You have now selected the variable name for dragging. For example, select the idmod variable for dragging from the wsgui scroll table and move the mouse button slightly. (Note: the wsgui workspace has just loaded the mk_wts file from the MATLAB workspace.) Your screen should look similar to Figure 6-57.



Variable Name	Type	Rows	Cols	Num
ans	Constant	1	1	0
baerr	System	1	1	1
cterr	System	1	1	1
fix	Constant	3	5	0
idmod	System	1	1	2
nyerr	System	1	1	1
wact	Constant	6	6	0
wgust	System	1	1	1
wl	Constant	3	9	0
wll	Constant	3	3	0
wlm	Constant	3	3	0
wlr	Constant	3	3	0

31–May–95 15:13:21

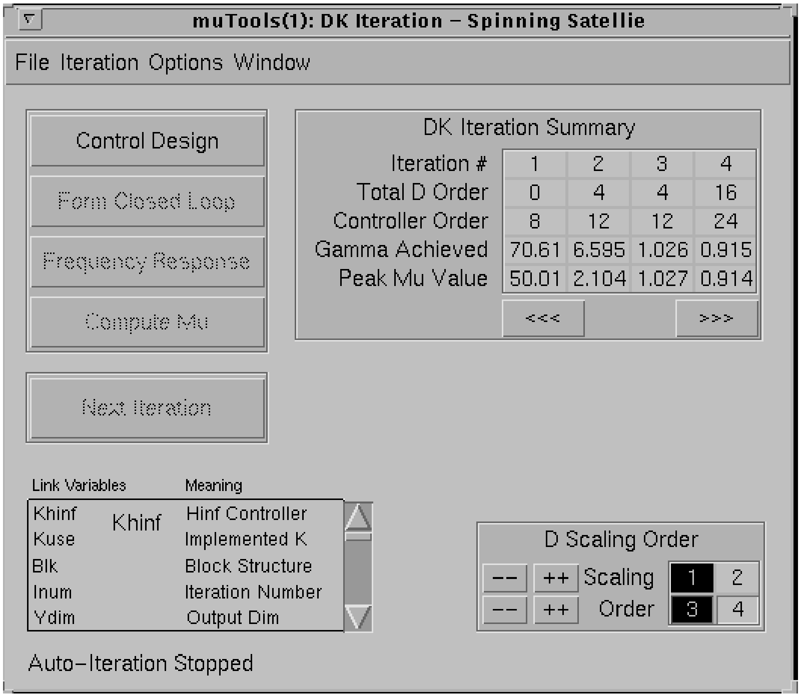
22 (of 22) Items Selected

Figure 6-57: Dragging the Controller Variable Name in dkitgui

There are several important facts regarding the dragging and dropping of icons that should be noted:

- Icons being dragged are only visible in windows, and they are not visible when the icons are over a frame or other MATLAB uicontrol objects.
- Dragging an icon may be slow on networked computers. This is due to the overhead of transferring the information about the icon being dragged.
- The drag and drop operation is only completed if the icon is deposited in a drop box.

Items being dragged need to be deposited directly inside a drop box. An example of a drop box is shown in Figure 6-59; it's to the left of the **Controller** push button in the **simgui** main window. To do this, position the icon being dragged in the center of a drop box and release the mouse button.



muTools(1): DK Iteration – Spinning Satellie

DK Iteration Summary

Iteration #	1	2	3	4
Total D Order	0	4	4	16
Controller Order	8	12	12	24
Gamma Achieved	70.61	6.595	1.026	0.915
Peak Mu Value	50.01	2.104	1.027	0.914

Link Variables	Meaning
Khinf	Hinf Controller
Kuse	Implemented K
Blk	Block Structure
Inum	Iteration Number
Ydim	Output Dim

Khinf

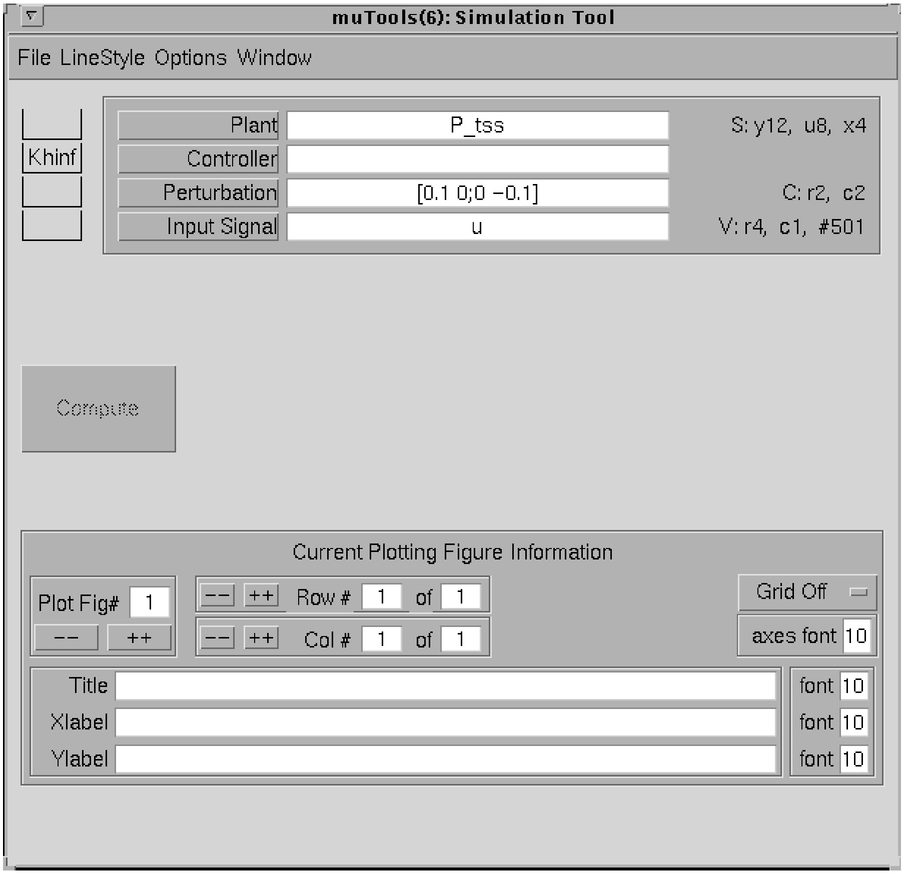
D Scaling Order

Scaling	1	2
Order	3	4

Auto–Iteration Stopped

Figure 6-58: Drag Khinf from dkitgui to simgui

For example, you can select to drag the fourth $D - K$ iteration controller from the spinning satellite example in the "D-K Iteration User Interface Tool: dkitgui" section to the LFT time simulation tool, `simgui`. Select the `Khinf` link variable in the `dkitgui` main window and move the mouse button slightly (Figure 6-57). Drag the `Khinf` link variable over to the main `simgui` window (Figure 6-58) and drop it in the `Controller` drop box, as shown in Figure 6-59. Upon successfully dragging and dropping the data, the string `grabvar(1,'Khinf')` appears in the main `simgui` window **Controller** editable text frame.



muTools(6): Simulation Tool

Plant: P_tss

Controller:

Perturbation: [0.1 0;0 −0.1]

Input Signal: u

S: y12, u8, x4

C: r2, c2

V: r4, c1, #501

Plot Fig# 1

Row # 1 of 1

Col # 1 of 1

Grid Off

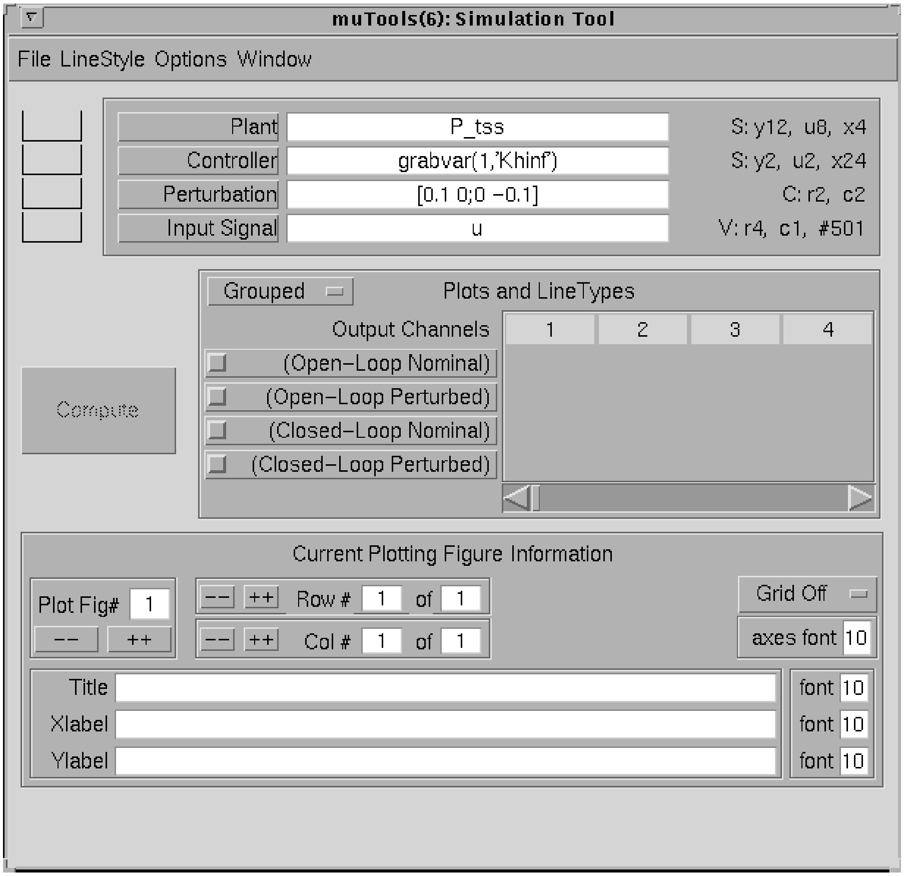
axes font 10

Title: font 10

Xlabel: font 10

Ylabel: font 10

Figure 6-59: Drop Khinf into simgui Controller Drop Box



Plant P_tss S: y12, u8, x4
Controller grabvar(1,'Khinf') S: y2, u2, x24
Perturbation [0.1 0;0 –0.1] C: r2, c2
Input Signal u V: r4, c1, #501

Plot Fig# 1
Row # 1 of 1
Col # 1 of 1
Grid Off
axes font 10
Title font 10
Xlabel font 10
Ylabel font 10

Figure 6-60: Evaluated Drop of Khinf

In `simgui` and `dkigui` the linkable variables are to be extracted from a large storage matrix. The program `grabvar` performs this operation. The inputs to the `grabvar` function are the μ-Tools GUI figure number and the variable name in the originating function. In this example, the variable dragged from `dkitgui` to `simgui` is `Kinf`. The `dkitgui` figure number is 1, which corresponds to the 1 in the `simgui` editable text: `grabvar(1,'Khinf')`, as seen in Figure 6-60. The drag and drop operation is responsible for entering the required input data and requires no additional input from you.

7

Robust Control Examples

This chapter has a number of examples to show how to apply the μ-Analysis and Synthesis Toolbox (μ-Tools) to robust control problems. These examples include:

- SISO Gain and Phase Margins
- MIMO Loop-at-a-Time Margins
- Analysis of Controllers for an Unstable System
- MIMO Margins Using m
- Space Shuttle Robustness Analysis
- HIMAT Robust Performance Design Example
- F–14 Lateral-Directional Control Design
- A Process Control Example: Two Tank System

SISO Gain and Phase Margins

Gain and phase margins play an important role in control by defining robustness margins for systems. Unfortunately these margins, by themselves, may be misleading in some cases.

Consider the plant

$$G(s) = \frac{s-2}{2s-1}$$

```
G = nd2sys([1 -2],[2 -1],-1);
minfo(G)
```

and four different controllers that stabilize the plant

$$K_1(s) = 1$$
$$K_2(s) = \frac{s+\beta}{\beta s+1} \quad (\beta > 0, \quad \text{as} \quad \beta \rightarrow 2)$$
$$K_3(s) = \frac{\beta s+1}{s+\beta} \quad (\beta > 0, \quad \text{as} \quad \beta \rightarrow 2)$$
$$K_4(s) = \frac{s+2.5}{2.5s+1}\frac{1.7s^2+1.5s+1}{s^2+1.5s+1.7}$$

Construct the controller K_1 and form the closed-loop using `formloop`.

```
k1 = 1;
T1 = formloop(G,k1,'neg','pos');
spoles(T1)
```

The command `formloop` produces a 2-input, 2-output system T_1, as shown in Figure 7-1. The third input argument `'neg'` denotes that there is negative feedback on the y_2 channel. The fourth input argument `'pos'` denotes that the feedback forward input u_2 is positive. These are the default values for `formloop`.

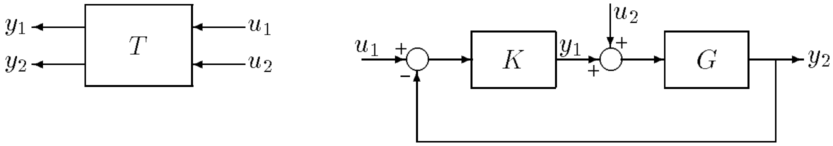


Figure 7-1 formloop Interconnection Structure

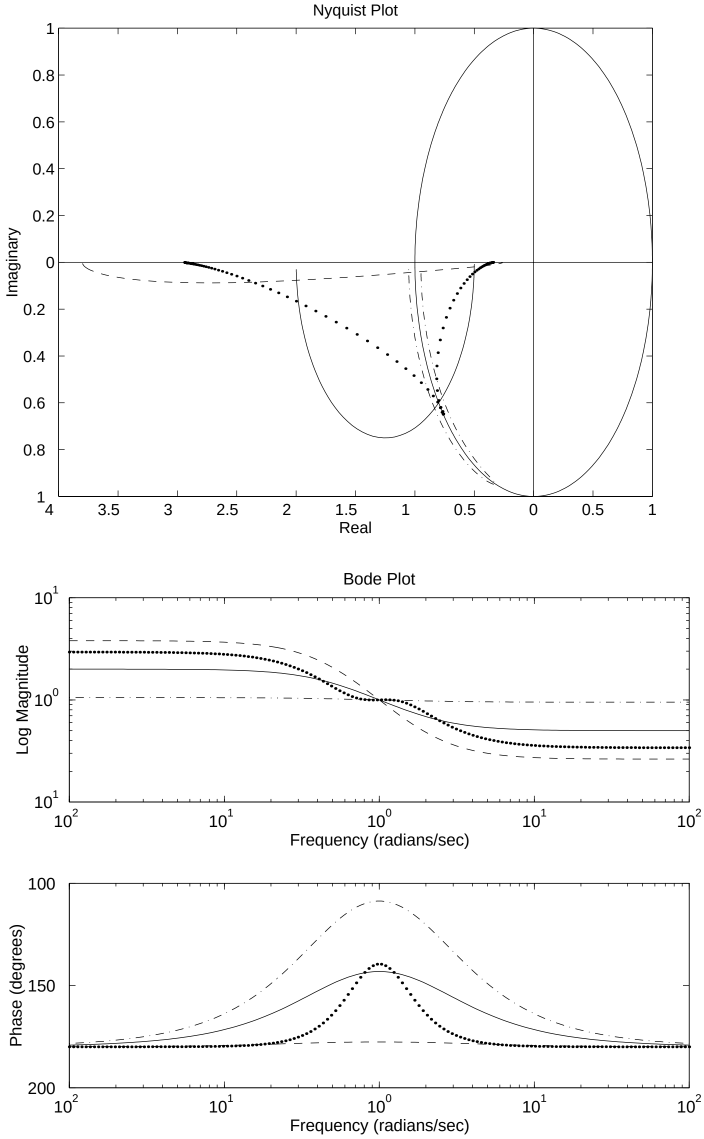
For the problem at hand, form the closed-loop system and calculate the poles of the closed-loop system.

```
K2 = nd2sys([1 1.9],[1.9 1]);
clp2 = formloop(G,K2);
spoles(clp2)
k3 = nd2sys([1.9 1],[1 1.9]);
clp3 = formloop(G,k3);
spoles(clp3)
k4 = nd2sys([1 2.5],[2.5 1]);
k4 = mmult(k4,nd2sys([1.7 1.5 1],[1 1.5 1.7]));
clp4 = formloop(G,k4);
spoles(clp4)
```

For gain and phase margins, we are interested in the loop transfer function L(s) := G(s) K(s). Select a frequency range of 0.01 rad/sec to 100 rad/sec with 100 points. Calculate the frequency response of the loop transfer function and plot the Bode and Nyquist diagrams of L. The command add_disk adds a unit disk to the Nyquist plot. The results are shown in Figure 7-2.

```
omega = logspace(-2,2,100);
G_g = frsp(G,omega);
k1_g = frsp(k1,omega);L1 = mmult(G_g,k1_g);
K2_g = frsp(K2,omega);L2 = mmult(G_g,K2_g);
k3_g = frsp(k3,omega);L3 = mmult(G_g,k3_g);
k4_g = frsp(k4,omega);L4 = mmult(G_g,k4_g);
vplot('nyq',L1,'-',L2,'--',L3,'-.',L4,'.'), add_disk
axis([-4 1 -1 1])
vplot('bode',L1,'-',L2,'--',L3,'-.',L4,'.'), grid
```

Controller K_1 provides a compromise between a large gain and phase margin. Controller K_2 optimizes the gain margin for this example, whereas controller K_3 optimizes the phase margin. Controller K_4 tries to optimize both gain and phase margins, resulting with about 50° phase margin, and a gain margin of 3 (it can tolerate increase or decrease in gain by a factor of 3). These margins, though, are very sensitive to small changes in the plant (a 50% gain change, to either 1.5 or 0.67, reduces the phase margin to less than 10°) illustrating that gain and phase margins are not always an accurate measure of the robustness of a control design even for single-input/single-output systems.



k1 (solid line) k2 (dashed line) k3 (dash-dotted line) k4 (dotted line)

Figure 7-2: Nyquist and Bode Plots of the Loop Gain with k1, k2, k3, and k4

MIMO Loop-at-a-Time Margins

This example is designed to illustrate that loop-at-a-time margins (gain, phase, and/or distance to –1) can be inaccurate measures of multivariable robustness margins. We will see that margins of the individual loops may be very sensitive to small perturbations within other loops.

The nominal closed-loop system considered here is shown in Figure 7-3.

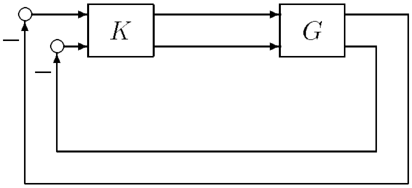


Figure 7-3: Nominal Closed-Loop System

G and K are 2 × 2 multi-input/multi-output (MIMO) systems, defined as

$$G := \frac{1}{s^2+\alpha^2}\begin{bmatrix} s-\alpha^2 & \alpha(s+1) \\ -\alpha(s+1) & s-\alpha^2 \end{bmatrix}, \qquad K = I_2$$

Set $\alpha := 10$, construct G in state-space form and compute its frequency response.

```
a = [0 10;-10 0];
b = eye(2);
c = [1 10;-10 1];
G = pck(a,b,c);
K = eye(2);
omega = logspace(-3,2,50);
Gg = frsp(G,omega);
minfo(Gg);
```

We want to consider perturbations to both input channels, as shown in Figure 7-4.

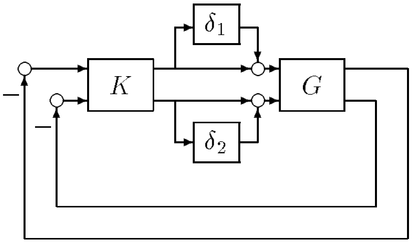


Figure 7-4: Perturbed Closed-Loop System

Break the loop at the place where one perturbation is, and compute the open-loop transfer function. This transfer function will be a function of the remaining perturbation. For instance, to check the margins in the first channel with perturbations in the second channel, consider the diagram in Figure 7-5.

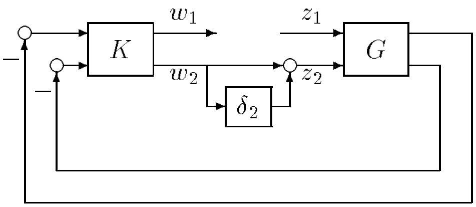


Figure 7-5: Closed-Loop System with Loop Broken in Channel 1 and Perturbation in Channel 2

In this particular example, *K* is the identity, so the loop in Figure 7-5 can be redrawn, as shown in Figure 7-6. This figure is easily constructed using `starp`.

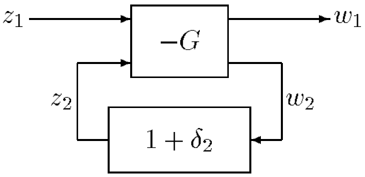


Figure 7-6: Redrawn Closed-Loop System with Loop Broken in Channel 1 and Perturbation in Channel 2

A Bode or Nyquist plot of the transfer function from z_1 to w_1 reveals the margin (nearness to +1 point in this case, since the negative sign is embedded in the transfer function from *z* to *w*) in the first channel as a function of perturbations in the second channel. Refer to Figure 7-6 and calculate the nominal transfer function obtained by breaking the loop in the 1st channel. A Bode plot of the transfer function is shown in Figure 7-7. Note that this loop is an integrator, so the single-loop margin is great. In a similar manner, calculate the nominal transfer function obtained by breaking the loop in the second channel. The results are shown in Figure 7-7.

```
nom_ch1_lb = starp(mscl(Gg,-1),1,1,1);
nom_ch2_lb = starp(1,mscl(Gg,-1),1,1);
vplot('bode',nom_ch1_lb,'-',nom_ch2_lb,'--');
```

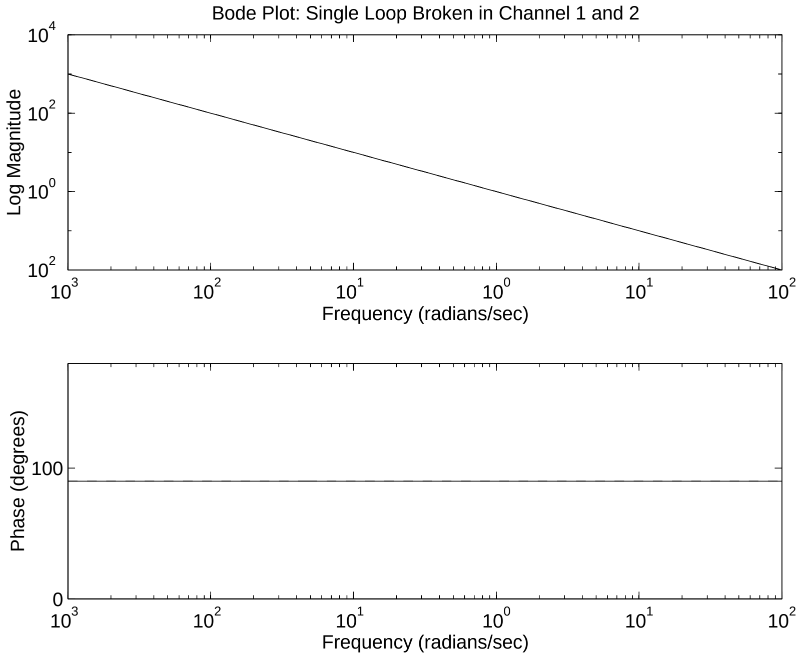


Figure 7-7: Bode Plot the Single-Loop Transfer Functions

The previous Bode plots show that each channel can tolerate large, loop-at-a-time, gain variations. Now though, consider a simultaneous 10% variation in each loop. Specifically, check the internal stability of the perturbed closed-loop shown in Figure 7-4, redrawn in Figure 7-8, with $\delta_1 = -1/\sqrt{101}$, and $\delta_2 = 1/\sqrt{101}$.

To do this, note that since $K = I_2$, the perturbed loop in Figure 7-4 is simply the star product (`starp`) of $-G$ and the matrix `[1+delta1 0; 0 1+delta2]`.

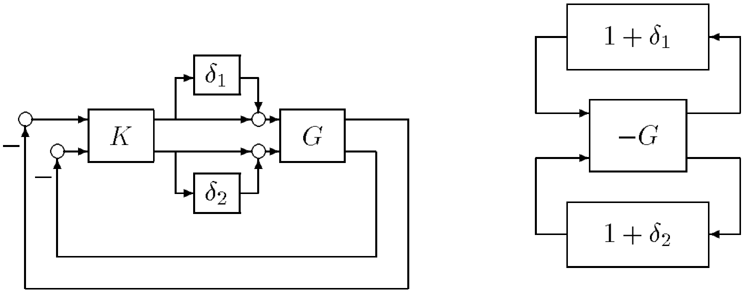


Figure 7-8: Equivalent Perturbed Closed-Loops

Note When two SYSTEM matrices are connected using `starp`, and all of the inputs of the top system are all of the outputs of the bottom system, and all of the inputs of the bottom system are all of the outputs of the top system, then the output of `starp` is the "A" matrix of the interconnection, stored as a regular MATLAB matrix.

The following code creates two perturbations of size $\sqrt{101}$, creates the perturbation matrix, forms the closed-loop system and calculates the closed-loop eigenvalues.

```
delta1 = -1/sqrt(101);
delta2 = 1/sqrt(101);
delta = [1+delta1 0 ; 0 1+delta2];
minfo(G)
minfo(delta)
clpAmat = starp(mscl(G,-1),delta,2,2)
minfo(clpAmat)
eig(clpAmat)
```

Notice that the eigenvalues of `clpAmat` are unstable. Hence, this small (10%), simultaneous perturbation to both channels causes instability (slightly larger values for δ_1 and δ_2 push the eigenvalues further into the right-half-plane).

We can see the effect of simultaneous perturbations with the help of a Nyquist plot. Refer back to Figure 7-6 and calculate the transfer function obtained by breaking the loop in the first channel, with a perturbation of δ_2 = 0.001. The single-loop margin in channel 1 is still quite large, since the Nyquist plot, solid line in Figure 7-9 intersects the real axis around 11. Try a slightly larger value for δ_2 (again, note that the +1 point is of interest due to the extra minus (–) sign on *G*).

```
delta2 = 0.001;
pert_ch1_lb1 = starp(mscl(Gg,-1),1+delta2,1,1);
vplot('nyq',pert_ch1_lb1);
delta2 = 0.01;
pert_ch1_lb2 = starp(mscl(Gg,-1),1+delta2,1,1);
vplot('nyq',pert_ch1_lb1,'-',pert_ch1_lb2,'--');
add_disk
```

Note the significant degradation in the margin due to a 1% variation in channel 2.

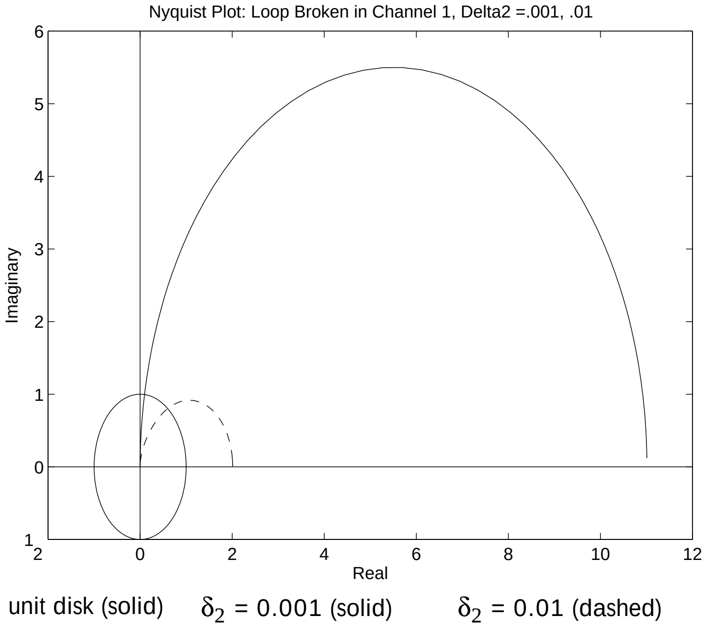


Figure 7-9: Nyquist Plot of Simultaneously Perturbed Closed-Loop System

It is easy to write a for loop (as in ex_ml1) that calculates the open-loop frequency response in channel 1 for several values of δ_2. Six δ_2 values are constructed from 0.001 to 0.1. A for loop calculates the perturbed loop transfer function in channel 1 for the six values of δ_2 and their responses are plotted on a Nyquist diagram.

file:ex_ml1.m

```
delta2values = logspace(-3,-1,6);
pert_ch1_lb = [];
for i=1:length(delta2values)
   delta2 = delta2values(i);
   pert_ch1_lb = ...
    sbs(pert_ch1_lb,starp(mscl(Gg,-1),1+delta2,1,1));
end
xa = vpck([-2; 12],[1 2]);
ya = vpck([-sqrt(-1); 6*sqrt(-1)],[1 2]);
vplot('nyq',pert_ch1_lb,xa,ya);
title('Loop Broken in Channel 1, Delta2 in [.001 .1]')
xlabel('Real')
ylabel('Imag')
```

Typing

```
ex_ml1
```

at the command line generates Figure 7-10. Note that the margin in channel 1 rapidly disappears due to small perturbations in channel 2.

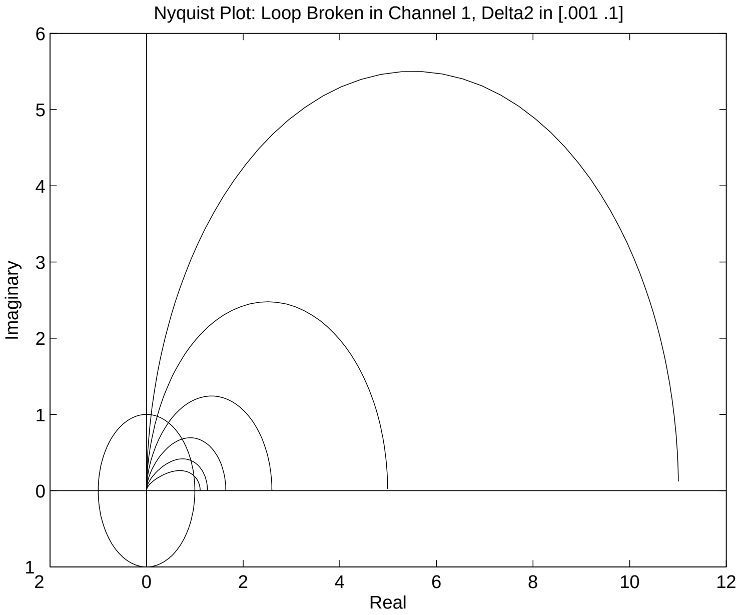


Figure 7-10: Nyquist Plot of Simultaneously Perturbed Closed-Loop System from δ_2 = 0.001 to δ_2 = 0.1

Exactly analogous results hold for breaking the loop in channel 2, and considering small perturbations in channel 1. This can be verified by closing the upper loop of $-G$ with $(1 + \delta_1)$. For example,

```
delta1 = -0.01;
pert_ch2_lb = starp(1+delta1,mscl(Gg,-1),1,1);
vplot('nyq',pert_ch2_lb);
add_disk
```

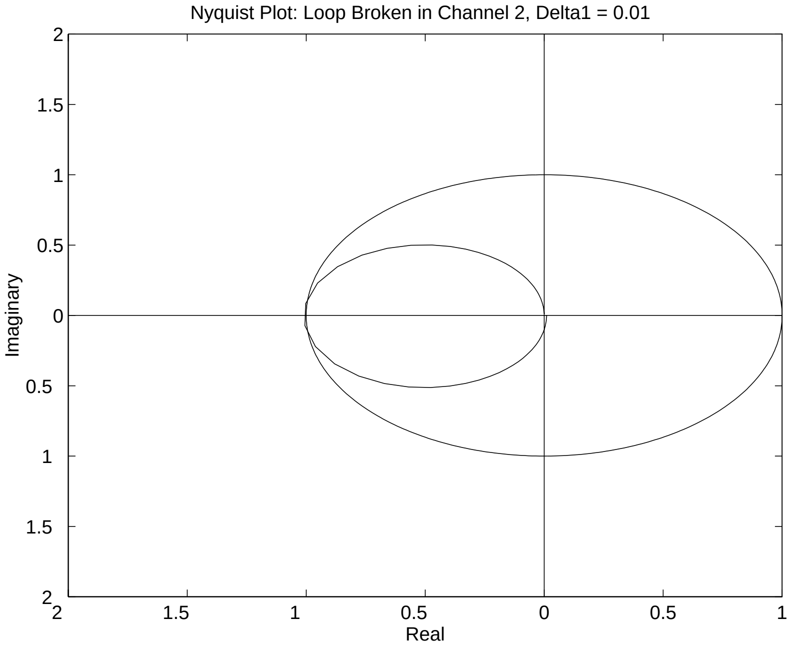


Figure 7-11: Nyquist Plot of Simultaneously Perturbed Closed-Loop System for δ_1 = –0.01

Hence, in a multivariable system the single loop margins may be good though simultaneous interaction of perturbations in each loop may lead to instability, even for small perturbations. The structured singular value, μ, can be used to detect the nonrobustness in this feedback system. To see this we will reanalyze this example using μ in the "MIMO Margins Using m" section in this chapter.

Analysis of Controllers for an Unstable System

Unstable example Recall from the “Unmodeled Dynamics” section in Chapter 4, the example with the nominal plant model

$$G := \frac{1}{s-1}$$

and the multiplicative input uncertainty model

$$W_u := \frac{\frac{1}{4}(\frac{1}{2}s+1)}{\frac{1}{32}s+1}$$

G and W_u describe a set $M(G, W_u)$ of plant models defined by

$$\mathcal{M}(G, W_u) := \{G(I+\Delta_G W_u) : \max_{\omega}|\Delta_G(j\omega)| \le 1\}. \tag{7-1}$$

with the restriction that the number of right-half plane poles of perturbed plant equal the number of right-half plane poles of G. The nominal model G, weighting function W_u, and the unknown transfer function Δ_G are used to parameterize all the possible models.

We are interested in the stability and performance of the closed-loop system for all possible plant models in the set $M(G, W_u)$. In this example, we choose the performance objective to be a stable, closed-loop system, and output disturbance rejection up to 0.6 rad/sec, with at least 100:1 disturbance rejection at DC.

This objective can approximately be represented as a weighted H_∞ norm constraint on the sensitivity function,

$$\left\|\frac{W_p}{1-\tilde{G}K}\right\|_\infty \le 1$$

for all $\tilde{G} \in \mathcal{M}(G, W_u)$, using the weighting function

$$W_p := \frac{\frac{1}{4}s + 0.6}{s} = s + 0.006$$

A block diagram of this uncertain closed-loop system illustrating the performance objective (closed-loop transfer function from $d \rightarrow e$) is shown in Figure 7-12.

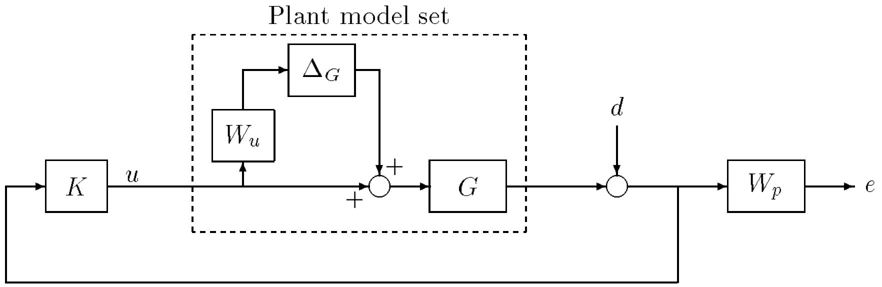


Figure 7-12: Uncertain Closed-Loop System

Grouping G, W_u, and W_p together (as P), the uncertain closed-loop system is redrawn in Figure 7-13.

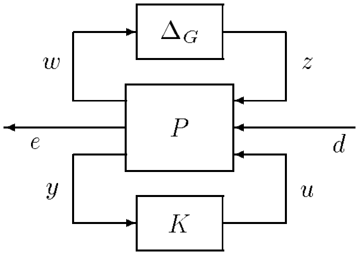


Figure 7-13: General Interconnection for Uncertain System

Type

```
ex_unic
```

to create the plant model, weighting functions and the open-loop interconnection P. The commands in `ex_unic` are as follows.

file: ex_unic.m

```
G = nd2sys(1,[1 -1]);
Wu = nd2sys([0.5 1],[0.03125 1],.25);
WP = nd2sys([.25 0.6],[1 0.006]);
systemnames = 'G Wu WP';
   sysoutname = 'P';
   inputvar = '[ z; d; u ]';
   input_to_p = '[ z + u ]';
   input_to_Wu = '[ u ]';
   input_to_WP = '[ G + d ]';
   outputvar = '[ Wu; WP; G + d ]';
   cleanupsysic = 'yes';
sysic
```

Next, consider two controllers, K_1 and K_2 which stabilize the nominal plant model G.

$$K_1 = -10\frac{0.9s+1}{s} \qquad K_2 = -1\frac{2.8s+1}{s}$$

For each controller, define $M := F_L(P,K)$. M_{22} is the nominal $d \rightarrow e$ transfer function, and hence, nominal performance is characterized by $\|M_{22}\|_\infty \leq 1$. The magnitude of the M_{22} transfer function with K_1 and K_2 implemented is shown in Figure 7-14. Both controllers easily achieve the nominal performance objective.

M_{11} is the transfer function that the perturbation Δ sees, so that robust stability over $\mathcal{M}(G, W_u)$ is equivalent to $\|M_{11}\|_\infty < 1$. The magnitude of the M_{11} transfer function is shown in Figure 7-15. Both controllers achieve the robust stability objective.

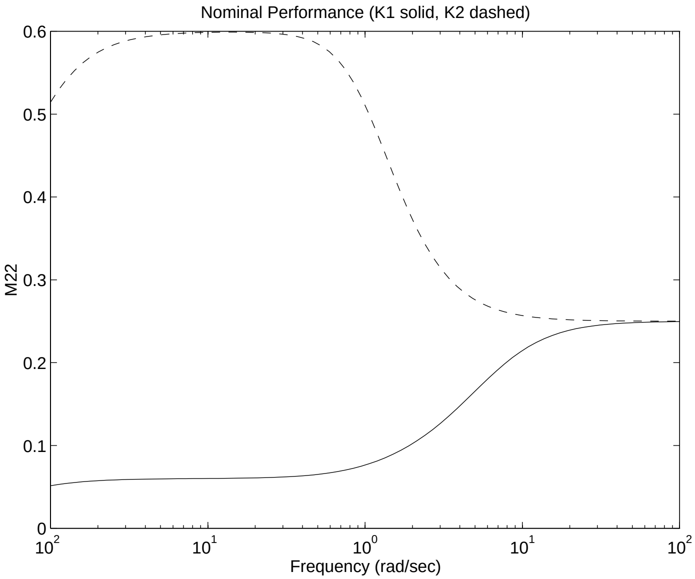


Figure 7-14: Nominal Performance Plots

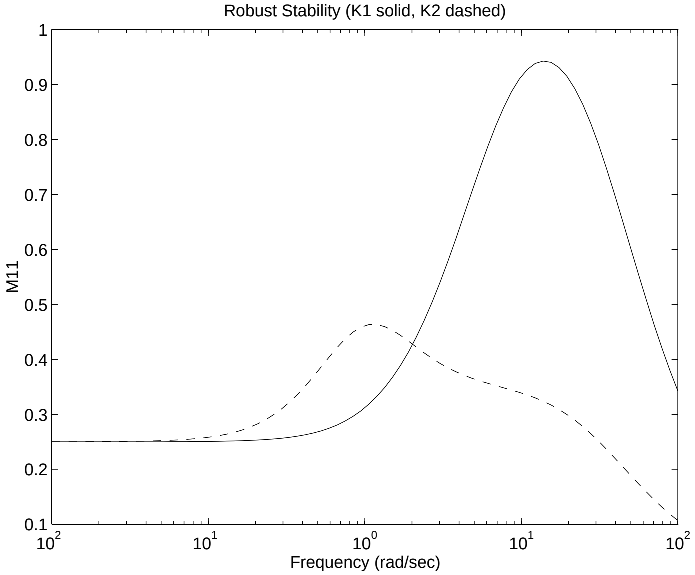


Figure 7-15: Robust Stability Plots

The actual goal is to achieve the performance objective for every plant model as defined by the dashed-box in Figure 7-12. This objective is defined as robust performance.We can use the structured singular value, μ, to analyze the closed-loop system to determine if robust performance is achieved. Achieving robust performance is mathematically equivalent to

$$\max_{\omega \in \mathbf{R}} \mu_{\Delta_P}(M(j\omega)) \leq 1$$

where Δ_P is the extended uncertainty set, consisting of the actual scalar unmodeled dynamics (i.e., a complex block) uncertainty block (representing Δ_G), and the fictitious uncertainty block used to reformulate the performance criterion as a stability problem (see section “Using m to Analyze Robust Performance” in Chapter 4 for details). The block structure used to evaluate robust performance is

$$\Delta_P := \{\text{diag}[\Delta,\Delta_F] : \Delta \in \mathbf{C}, \Delta_F \in \mathbf{C}\},$$

A plot of$\mu_{\Delta_P}(M(j\omega))$ is shown in Figure 7-16.

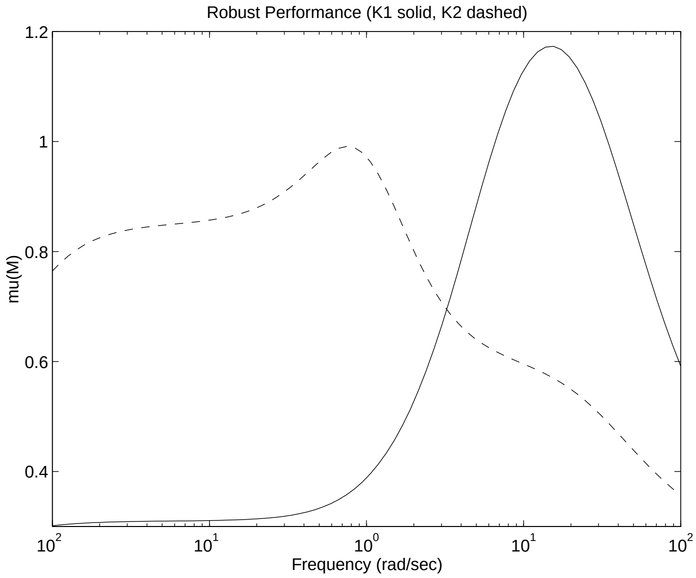


Figure 7-16: Robust Performance Plots

Note that while K_1 achieves better nominal performance than K_2, the closed-loop system with K_2 has better robust performance properties than with K_1. In fact, the controller K_2 does, just barely, achieve robust performance. Hence, for every plant $\tilde{G} \in \mathcal{M}(G, W_u)$, the closed-loop system, with K_2 is stable, and moreover

$$\left\| \frac{W_p}{1 - \tilde{G}K_2} \right\|_\infty < 1$$

This is not true for the closed-loop system with K_1, as the robust performance test, using μ is not satisfied for $F_L(P,K_1)$.

The script-file

```
ex_usrp
```

constructs the controllers, forms the closed-loop systems, calculates the closed-loop frequency responses, performs the analysis, and generates the plots. A listing of `ex_usrp` is as follows.

file: ex_usrp.m

```
K1 = nd2sys([.9 1],[1 0],-10);
K2 = nd2sys([2.8 1],[1 0],-1);
om = logspace(-2,2,80);
M1 = starp(P,K1);
M2 = starp(P,K2);
M1g = frsp(M1,om);
M2g = frsp(M2,om);
uncblk = [1 1];
fictblk = [1 1];
deltaset = [uncblk;fictblk];
bnds1 = mu(M1g,deltaset);
bnds2 = mu(M2g,deltaset);
vplot('liv,m',sel(M1g,2,2),'-',sel(M2g,2,2),'--')
xlabel('Frequency (rad/sec)')
ylabel('M22');
title('Nominal Performance (K1 solid, K2 dashed)');
vplot('liv,m',sel(M1g,1,1),'-',sel(M2g,1,1),'--')
xlabel('Frequency (rad/sec)')
ylabel('M11');
title('Robust Stability (K1 solid, K2 dashed)');
vplot('liv,m',sel(bnds1,1,1),'-',sel(bnds2,1,1),'--')
xlabel('Frequency (rad/sec)')
ylabel('mu(M)');
title('Robust Performance (K1 solid, K2 dashed)');
```

How does the weighted sensitivity transfer function degrade with uncertainty? This can be answered easily using the worst-case performance analysis, which shows the worst-case degradation of performance as a function of the size of the uncertainty.We also construct, for each closed-loop system, the worst-case perturbation of size (i.e., the worst case plant from $M(G, W_u)$) and use this later in time-domain simulations. The script file

```
ex_wcp
```

performs these calculations.

file: ex_wcp.m

```
[deltabad1,wcp_l1,wcp_u1] = wcperf(M1g,uncblk,1,8);
[deltabad2,wcp_l2,wcp_u2] = wcperf(M2g,uncblk,1,8);
vplot(wcp_l1,wcp_u1,wcp_l2,'--',wcp_u2,'--')
xlabel('Size of Delta_G')
ylabel('Weighted Sensitivity')
title('Performance Degradation (K1 solid, K2 dashed)')
```

The performance degradation curve is shown in Figure 7-17. The fourth argument to `wcperf`, in this case 8, means that there will be at least eight points in the performance degradation curve.

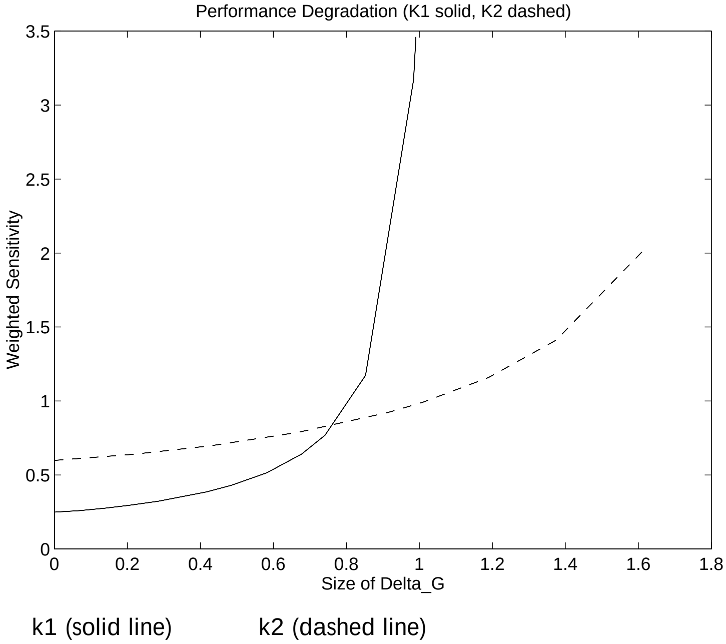


Figure 7-17: Performance Degradation

This clearly shows that for perturbations Δ_G satisfying

$$\max_{\omega}|\Delta_G(j\omega)| \text{ Š } 0.76,$$

the robust closed-loop performance using K_2 is better than that obtained using K_1. The perturbations `deltabad1` and `deltabad2` are the worst-case perturbations Δ_G, of size 1.They each cause the weighted sensitivity to degrade a maximal amount, over all perturbations of size 1. In the case of K_1, the perturbation `deltabad1` causes the weighted sensitivity to degrade to approximately 3.5, while in the case of K_2, `deltabad2` causes the weighted sensitivity to degrade to approximately 1.0. We will use these worst-case perturbations in time-domain simulations.

Based on the robust performance analysis results, K_2 will stabilize the nominal plant and the seven extreme plant models discussed in the "Unmodeled Dynamics" section in Chapter 4. Also, the performance degradation, in terms of the weighted sensitivity function, should be small. Using controller K_1, stability is also guaranteed. However, since $F_L(P,K_1)(j\omega)$ has a peak μ-value of about 1.2, it is clear that K_1 does not have robust performance over the set $\mathcal{M}(G,W_u)$ and we expect the performance degradation over the seven extreme plants to be worse than that using K_2.

These seven plants, all of which are members of the set $\mathcal{M}(G,W_u)$ and the worst-case plant from $\mathcal{M}(G,W_u)$, are

$$G_1 = \frac{1}{s-1}\left(\frac{6.1}{s+6.1}\right) \qquad G_2 = \frac{1.425}{s-1.425}$$

$$G_3 = \frac{.67}{s-0.67} \qquad G_4 = \frac{1}{s-1}\left(\frac{-0.07s+1}{0.07s+1}\right)$$

$$G_5 = \frac{1}{s-1}\left(\frac{70^2}{s^2+2\cdot 0.15\cdot 70s+70^2}\right) \qquad G_6 = \frac{1}{s-1}\left(\frac{70^2}{s^2+2\cdot 5.6\cdot 70s+70^2}\right)$$

$$G_7 = \frac{1}{s-1}\left(\frac{50}{s+50}\right)^6 \qquad G_{wc} = [G(1+\Delta_{wc}W_u)]$$

and are constructed via

```
ex_mkpl
```

The contents of `ex_mkpl` are listed below.

file: ex_mkpl.m

```
G1 = mmult(G,nd2sys([6.1],[1 6.1]));
G2 = nd2sys([1+0.425],[1 -1-0.425]);
G3 = nd2sys([1-0.33],[1 -1+0.33]);
G4 = mmult(G,nd2sys([-0.07 1],[0.07 1]));
G5 = mmult(G,nd2sys([70^2],  [1 2*0.14*70 70^2]));
G6 = mmult(G,nd2sys([70^2],  [1 2*5.6*70 70^2]));
Gt = nd2sys([50],[1 50]);
G7 = mmult(G,Gt,Gt,Gt,Gt,Gt,Gt);
Gwc1 = mmult(G,madd(1,mmult(deltabad1,Wu)));
Gwc2 = mmult(G,madd(1,mmult(deltabad2,Wu)));
```

The closed-loop system shown in Figure 7-18 is used for time simulations. Twenty time simulations are computed, for all combinations of K_1 and K_2, with the nominal plant G, the seven extreme plants $G1,...,G7$, and the two worst-case plants G_{wc1} and G_{wc2}. Rather than simulate an output disturbance, we manipulate the diagram, and simulate a unit-step reference command.

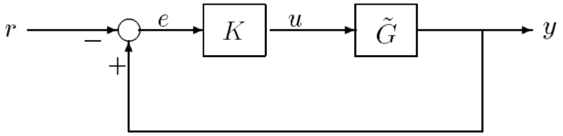


Figure 7-18: Simulation Block Diagram

This interconnection is most easily constructed using `formloop`, as described earlier. We use positive feedback (as that was the convention used in the design of the controllers) and invert the reference signal gain, to accomplish the tracking. The (2,1) entry of the MIMO system that `formloop` produces is the SYSTEM matrix from reference input r to plant output y.

Typing

```
ex_mkclp
```

forms the 20 SISO systems, as below.

file: ex_mkclp.m

```
clpg_k1 = sel(formloop(G,K1,'pos','neg'),2,1);
clpg1_k1 = sel(formloop(G1,K1,'pos','neg'),2,1);
clpg2_k1 = sel(formloop(G2,K1,'pos','neg'),2,1);
clpg3_k1 = sel(formloop(G3,K1,'pos','neg'),2,1);
clpg4_k1 = sel(formloop(G4,K1,'pos','neg'),2,1);
clpg5_k1 = sel(formloop(G5,K1,'pos','neg'),2,1);
clpg6_k1 = sel(formloop(G6,K1,'pos','neg'),2,1);
clpg7_k1 = sel(formloop(G7,K1,'pos','neg'),2,1);
clpgwc1_k1 = sel(formloop(Gwc1,K1,'pos','neg'),2,1);
clpgwc2_k1 = sel(formloop(Gwc2,K1,'pos','neg'),2,1);
clpg_K2 = sel(formloop(G,K2,'pos','neg'),2,1);
clpg1_K2 = sel(formloop(G1,K2,'pos','neg'),2,1);
clpg2_K2 = sel(formloop(G2,K2,'pos','neg'),2,1);
clpg3_K2 = sel(formloop(G3,K2,'pos','neg'),2,1);
clpg4_K2 = sel(formloop(G4,K2,'pos','neg'),2,1);
clpg5_K2 = sel(formloop(G5,K2,'pos','neg'),2,1);
clpg6_K2 = sel(formloop(G6,K2,'pos','neg'),2,1);
clpg7_K2 = sel(formloop(G7,K2,'pos','neg'),2,1);
clpgwc1_K2 = sel(formloop(Gwc1,K2,'pos','neg'),2,1);
clpgwc2_K2 = sel(formloop(Gwc2,K2,'pos','neg'),2,1);
```

These closed-loop systems are simulated with a unit-step reference input at t = 0.5. The time response with controllers K_1 (top) and K_2 (bottom) implemented are shown in Figure 7-19.

Typing

```
ex_ustr
```

runs the simulations, and generates the time response plots.

file: ex_ustr.m

```
stepref = step_tr([0 0.5 10],[0 1 1],.01,10);
Tfinal = 10;
tinc = 0.01;
yg_k1 = trsp(clpg_k1,stepref,Tfinal,tinc);
yg1_k1 = trsp(clpg1_k1,stepref,Tfinal,tinc);
yg7_k1 = trsp(clpg7_k1,stepref,Tfinal,tinc);
ygwc1_k1 = trsp(clpgwc1_k1,stepref,Tfinal,tinc);
ygwc2_k1 = trsp(clpgwc2_k1,stepref,Tfinal,tinc);
yg_k2 = trsp(clpg_k2,stepref,Tfinal,tinc);
yg1_k2 = trsp(clpg1_k2,stepref,Tfinal,tinc);
ygwc2_k2 = trsp(clpgwc2_k2,stepref,Tfinal,tinc);
subplot(211)
vplot(vdcmate(yg_k1,5),'+',yg1_k1,yg2_k1, yg3_k1,yg4_k1,...
     yg5_k1,yg6_k1,yg7_k1,ygwc1_k1,ygwc2_k1);
xlabel('Time (seconds)')
title('Closed-loop responses using k1')
subplot(212)
vplot(vdcmate(yg_k2,5),'+',yg1_k2,yg2_k2, yg3_k2,yg4_k2,...
     yg5_k2,yg6_k2,yg7_k2,ygwc1_k2,ygwc2_k2);
xlabel('Time (seconds)')
title('Closed-loop responses using K2')
```

Both controllers stabilize all plant models. The closed-loop performance provided by K_2 is very similar for all the plants. This is expected based on the robust performance analysis using μ, which showed that every plant in the set $\mathcal{M}(G,W_u)$ is stabilized and effectively controlled with K_2. The closed-loop performance provided by K_1 is not very robust. The large overshoot, undershoot and oscillations for some of the responses reinforces the notion that K_1 does not achieve robust performance, as defined by the uncertain plant model set $\mathcal{M}(G,W_u)$, and the weighted sensitivity transfer function.

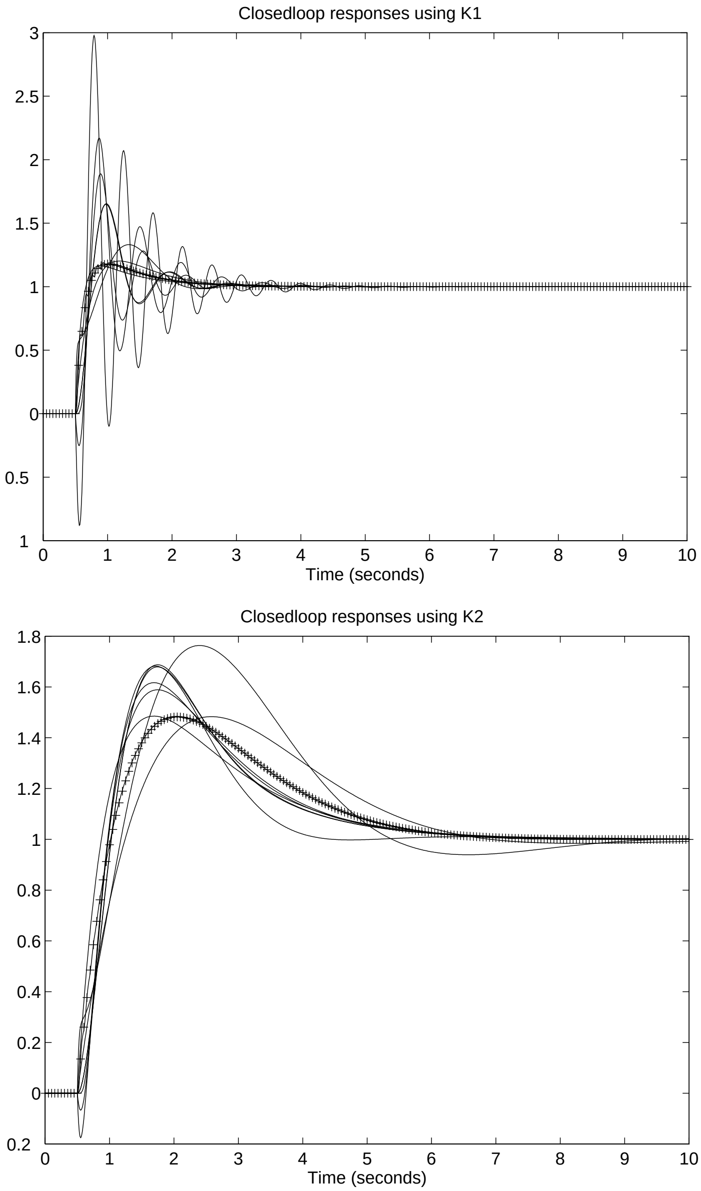
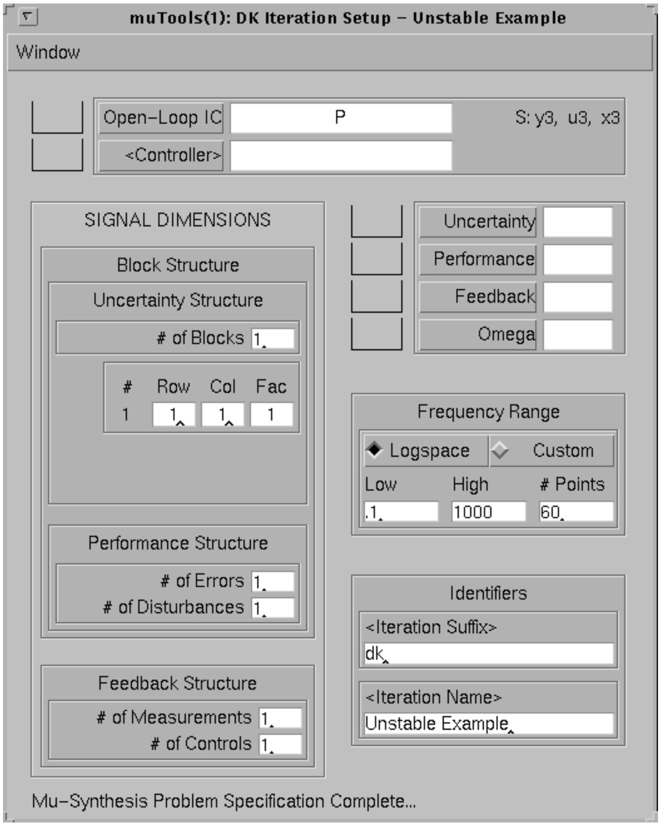


Figure 7-19: Time Response with K1 (top) and K2 (bottom) Implemented

Redesign Controllers Using D – K Iteration

The $D - K$ iteration, using `dkitgui`, can be used to design a robust controller for the uncertain plant model $M(G,W_u)$ and weighted sensitivity performance objective implied by the interconnection structure in Figure 7-12. Start `dkitgui`, and enter the data as shown in the `dkitgui` **Setup** window, Figure 7-20.



muTools(1): DK Iteration Setup – Unstable Example

Window

Open–Loop IC: P — S: y3, u3, x3

<Controller>

SIGNAL DIMENSIONS

Block Structure

Uncertainty Structure

# of Blocks: 1

#	Row	Col	Fac
1	1	1	1

Performance Structure

# of Errors: 1

# of Disturbances: 1

Feedback Structure

# of Measurements: 1

# of Controls: 1

Uncertainty

Performance

Feedback

Omega

Frequency Range

◆ Logspace ◇ Custom

Low	High	# Points
.1	1000	60

Identifiers

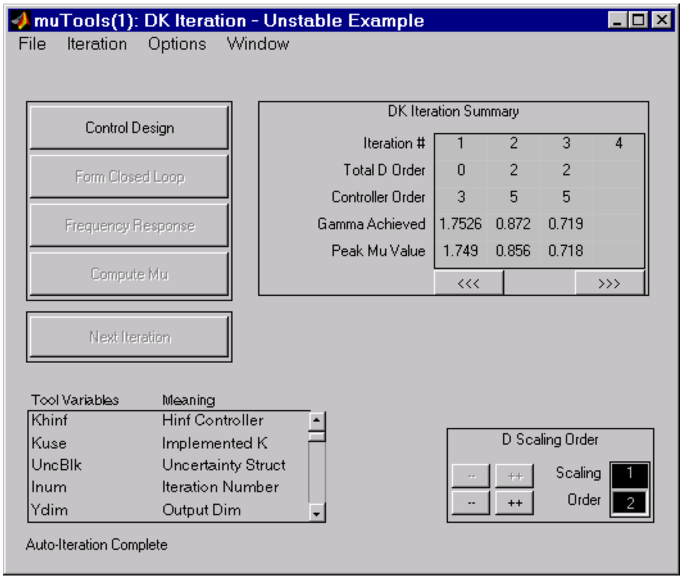
<Iteration Suffix>: dk

<Iteration Name>: Unstable Example

Mu–Synthesis Problem Specification Complete...

Figure 7-20: Unstable System dkitgui Setup Window

Return to the main Iteration window, as shown inFigure 7-21, and pull down the `Iteration` menu and select `3` from the `Auto Iterate` submenu. Figure 7-21 shows the main **Iteration** window after the third iteration has completed.



DK Iteration Summary

Iteration #	1	2	3	4
Total D Order	0	2	2	
Controller Order	3	5	5	
Gamma Achieved	1.7526	0.872	0.719	
Peak Mu Value	1.749	0.856	0.718	

Tool Variables	Meaning
Khinf	Hinf Controller
Kuse	Implemented K
UncBlk	Uncertainty Struct
Inum	Iteration Number
Ydim	Output Dim

D Scaling Order

Scaling 1

Order 2

Auto-Iteration Complete

Figure 7-21: Main dkitgui Window After Three Iterations

Controller K1dk corresponds to the first $D - K$ iteration H_∞ design and controller K3dk corresponds to the third $D - K$ iteration design.

Controller K1dk stabilizes all seven plant models, though the performance is poor, with large settling times, and large overshoot. This is not surprising, based on the robust performance analysis using μ. The value of μ with K1dk implemented was $1.55(> 1)$.

Controller K3dk achieves a closed-loop, robust performance μ-value of about 0.7, guaranteeing stability and good closed-loop performance for all seven plant models (as these are contained in the set $M(\mathrm{G}, W_u)$). The simulations agree with this conclusion. Note the improvement in overshoot and settling time (and greater insensitivity) over all of the previous closed-loop simulations (using K_1, K_2 and K_{1dk}). The nominal responses are denoted by the “+” symbol linetype. Typing

```
ex_ustrd
```

generates the plots shown inFigure 7-22.

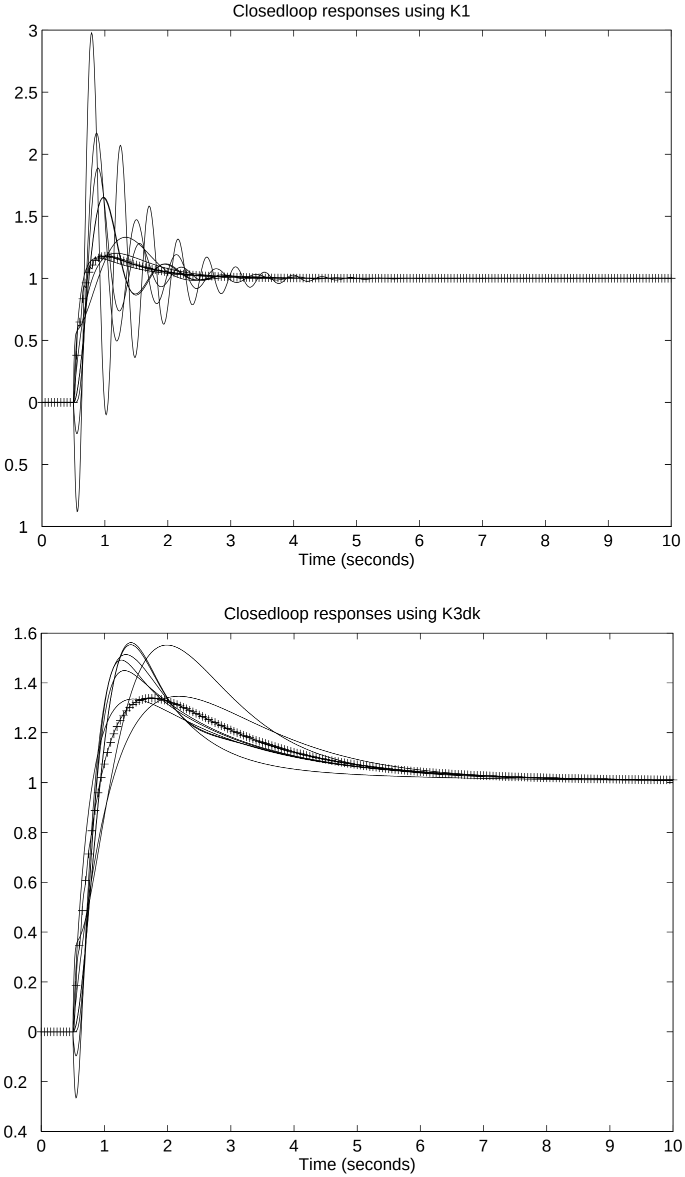


Figure 7-22: Time Response with K1dk (top) and K3dk (bottom) Implemented

Near the H_∞ optimal solution, the controller tends to have high bandwidth, providing some argument against using the H_∞ norm as the sole measure of performance. For this reason, we often back off from optimality, leading to a lower bandwidth, and more reasonable controller. In the **Parameter** window, change `Gamma Min` to 0.78, and change `Gamma Max` to 0.78. This fixes the next control design to be at $\gamma = 0.78$, which is approximately 10% above the optimal value of 0.71. Then, in the **Iteration** window, press **Control Design** to produce `K4dk`. This controller will sacrifice some (about 10%) H_∞ performance, but have much lower bandwidth. You should repeat the time-domain simulations for this controller.

You can also do simple model reduction on this controller, using `sysbal` and `strunc`. In the **Setup** window, type

```
reducek(
```

in the `<Controller>` data entry box. Then, drag, from the **Iteration** window, the linkable variable `Khinf` into the **Setup** window drop box for the controller. Finally, type

```
,3)
```

in the controller data entry box, which should now read

```
reducek(grabvar(2,'Khinf'),3)
```

The `2` corresponds to the MATLAB figure number associated with the **Iteration** window. In your case, it may not be 2. However, the dragging action will put the correct number in this place. Next, press the **<Controller>** pushbutton. This will grab the current H_∞ controller, balance it, and reduce it to third order using a simple M-file `reducek`, and place the reduced order controller back into `Kuse`. `Kuse` is the controller for which all analysis in `dkitgui` takes place (in `dkitgui`, the variable `Kuse` is automatically set to `Khinf` immediately after `Khinf` is designed). This reenables the **Form Closed Loop** pushbutton, which will implement the controller stored in `Kuse`. Pressing the **Frequency Response** and **Compute Mu** pushbuttons, you can determine the effectiveness of the reduced order controller, and repeat the process for different choice of controller order. In this example, a third order controller is suitable.

Time responses using this suboptimal, reduced order controller are shown in Figure 7-23.

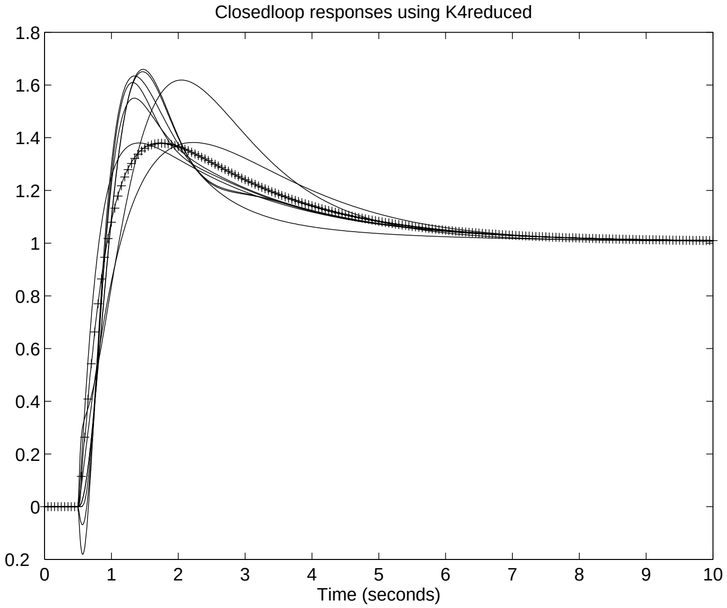


Figure 7-23: Time Response with K4dkreduced Implemented

MIMO Margins Using μ

Recall the conclusions of the “MIMO Loop-at-a-Time Margins” section concerning loop-at-a-time margins in MIMO systems. These margins can be inaccurate measures of multivariable robustness margins. In this exercise, we compare two controllers for a weighted robust stability problem similar to that in the“MIMO Loop-at-a-Time Margins” section, and see that the structured singular value, μ, accurately predicts their robust stability characteristics.

Consider an uncertainty model consisting of multiplicative uncertainty at the plant input in each channel. A block diagram of the model of the open-loop plant, with perturbations, is shown in Figure 7-24.

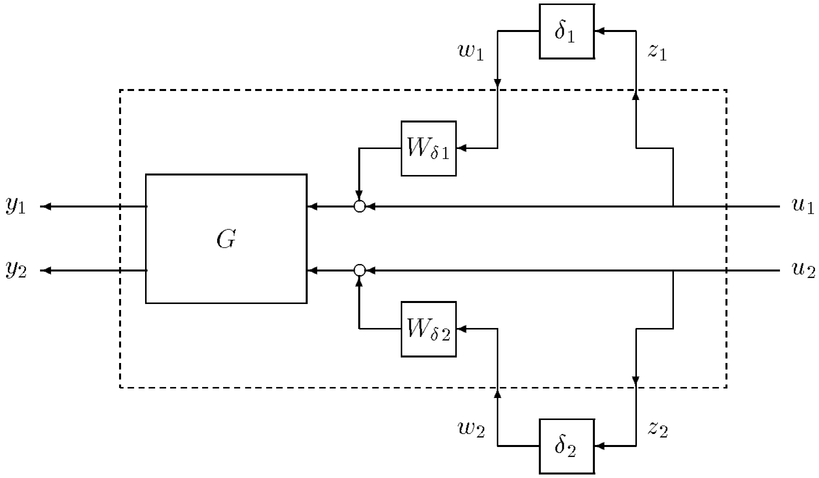


Figure 7-24: Plant Model with Multiplicative Uncertainty

The linear, time-invariant perturbations, represented by transfer functions δ_1 and δ_2, are used to model the uncertainty in the plant, and are assumed to satisfy

- $\delta_i(s)$ is stable
- $\max\limits_{\omega \in \mathbf{R}} |\delta_i(j\omega)| < 1$,

but otherwise be unknown. Everything inside the dashed box, in particular, the transfer functions G, $W_{\delta 1}$ and $W_{\delta 2}$, along with the specific interconnection, are assumed given and known. Since the uncertainty represented by the δ_i is normalized to 1, the transfer functions *W*δi reflect the frequency-dependent size of the uncertainty.

The first input channel is modeled with 10% uncertainty at low frequency, increasing to 100% uncertainty at approximately 20 rads/sec, and larger uncertainty for higher frequencies. In channel 2, the low frequency uncertainty is 20%, rising to 100% at approximately 45 rads/sec. Type in the following data for the problem.

```
a = [0 10;-10 0];
b = eye(2);
c = [1 10;-10 1];
G = pck(a,b,c);
wdel1 = nd2sys([1 2],[1/60 20]);
wdel2 = nd2sys([1 9],[1/40 45]);
omega = logspace(-3,2,50);
wdel1_g = frsp(wdel1,omega);
wdel2_g = frsp(wdel2,omega);
vplot('liv,lm',wdel1_g,wdel2_g)
title('Multiplicative Uncertainty Weights')
```

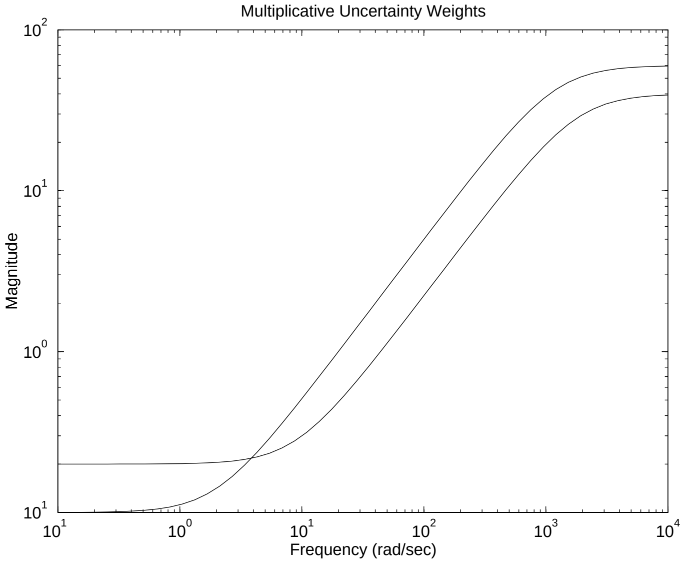
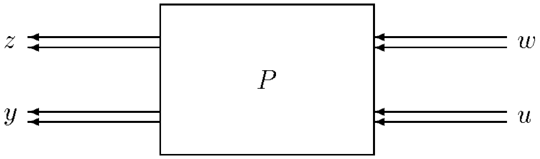


Figure 7-25: Multiplicative Uncertainty Weights

To use the μ theory, we need to isolate the perturbations from the known components. We do this by constructing a four input, four output system, with two fictitious input/output pairs (*w*,*z*), as well as two actual input/output pairs (*u*,*y*). The fictitious inputs and outputs are used to model the multiplicative uncertainty at the plant input. The system is called *P*, and is shown below.



The internal structure of *P* is shown in Figure 7-26.

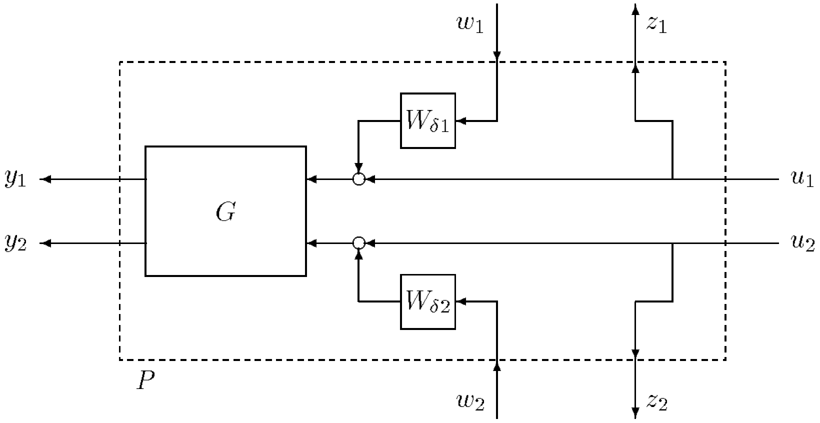


Figure 7-26: Plant Model with Multiplicative Uncertainty Blocks Pulled Out

The SYSTEM matrix *P* can be constructed easily from the components, using the SYSTEM interconnection program, sysic.

```
systemnames = 'G wdel1 wdel2';
inputvar = '[w{2}; u{2}]';
input_to_G = '[u(1) + wdel1; u(2) + wdel2]';
input_to_wdel1 = '[w(1)]';
input_to_wdel2 = '[w(2)]';
outputvar = '[u; G]';
sysoutname = 'P';
sysic;
```

The feedback controller for this problem will measure the variables y, and generate control signals, u. In a block diagram, the perturbed closed-loop system appears as shown in Figure 7-27. In terms of P, the perturbed closed-loop system appears as Figure 7-28.

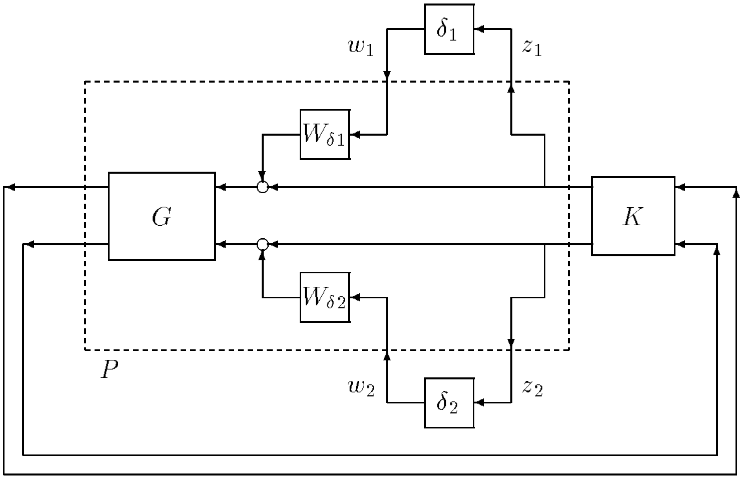


Figure 7-27: Closed-Loop System with δ_1 and δ_2 Uncertainty

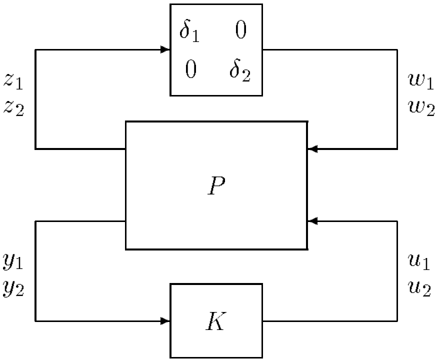


Figure 7-28: Closed-Loop System with Uncertainty

Next, we need to construct the transfer function that the 2 × 2 perturbation matrix Δ sees, namely $M := F_L(P,K)$. This transfer function is computed by closing the measurement/control feedback loop of the system P with the controller. For this example, we will compare two controllers, K_a, a constant gain 2×2 matrix, and K_b, a four-state, two input, two output SYSTEM. The command ex_mmmk makes up controllers K_a and K_b.

```
ex_mmmk;
ka
ka =
   -1     0
    0    -1
seesys(kb)
```

```
–1.9e+01   2.6e+01  –4.1e+01  –2.4e+01 | –5.4e+00  –2.6e+00

–8.9e+00  –4.5e+01   9.4e+00   8.3e+01 |  5.1e+00  –6.5e+00

 4.1e+01   2.2e–02  –7.4e+01  –6.7e+01 |  4.1e+00   3.5e+00

 1.6e+01  –8.4e+01   1.2e+01  –3.4e+02 |  4.9e+00  –5.0e+00

---------------------------------------|-------------------

–1.1e+00  –7.0e+00  –2.2e+00   5.2e+00 |  0.0e+00   0.0e+00

 5.9e+00  –4.5e+00   5.0e+00   4.8e+00 |  0.0e+00   0.0e+00
```

Next, close the lower loop of P with the controller (K_a or K_b) yielding M, which casts the robust stability problem as that depicted in Figure 4-13. Use starp to compute the closed-loop system M. A block diagram of this is shown in Figure 7-29.

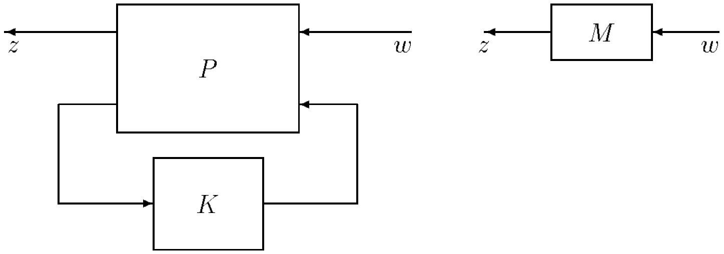


Figure 7-29: Closed-Loop Interconnection Block Diagrams

```
  Ma = starp(P,ka,2,2);
  Mb = starp(P,kb,2,2);
  spoles(Ma)

      real          imaginary        frequency          damping

     –1.8000e+03      0.0000e+00      1.8000e+03      1.0000e+00

     –1.2000e+03      0.0000e+00      1.2000e+03      1.0000e+00

     –1.0000e+00      0.0000e+00      1.0000e+00      1.0000e+00

     –1.0000e+00      0.0000e+00      1.0000e+00      1.0000e+00


  spoles(Mb)

real             imaginary        frequency        damping

     –1.5959e+01      0.0000e+00      1.5959e+01      l.0000e+00

     –2.7899e+01     –1.6622e+01      3.2475e+01      8.5908e–01

     –2.7899e+01      1.6622e+01      3.2475e+01      8.5908e–01

     –4.7231e+01     –6.8509e+00      4.7725e+01      9.8964e–01

     –4.7231e+01      6.8509e+00      4.7725e+01      9.8964e–01

     –3.1101e+02      0.0000e+00      3.1101e+02      1.0000e+00

     –1.2000e+03      0.0000e+00      1.2000e+03      1.0000e+00

     –1.8000e+03      0.0000e+00      1.8000e+03      1.0000e+00
```

Loop-at-a-Time Robustness

The loop-at-a-time robustness measures are computed by separately plotting the transfer function that each δ_i sees. This corresponds to the (1,1) and (2,2) entries of M. Recall that, if $\delta_2 \equiv 0$, then the perturbed closed-loop system is stable for all $\delta_1(s)$ satisfying

$$\|\delta_1\|_\infty < \frac{1}{\|M_{11}\|_\infty}$$

and that there is a perturbation $\delta_1(s)$, with $\|\delta_1\|_\infty = 1/\|M_{11}\|_\infty$

that causes instability. This is the robustness test for perturbations in channel 1, with no uncertainty in channel 2. Similar comments apply for the robustness test for perturbations in channel 2, with no uncertainty in channel 1.

The results for Ma_g (solid) and Mb_g (dashed) are computed below, and shown in Figure 7-30.

```
Ma_g = frsp(Ma,omega);
Mb_g = frsp(Mb,omega);
vplot('liv,m',sel(Ma_g,1,1),'-',sel(Mb_g,1,1),'--');
vplot('liv,m',sel(Ma_g,2,2),'-',sel(Mb_g,2,2),'--');
```

With respect to these uncertainty weights, the closed-loop system associated with controller K_a (solid) has better loop-at-a-time margins than the closed-loop system associated with controller K_b (dashed) since the peak is lower. Note that this holds for both channels.

The robustness measure for simultaneous perturbations in channels 1 and 2 requires the structured singular value, μ. Recall from Chapter 4 that the structured singular value of the matrix that the perturbation sees gives a nonconservative bound on the allowable simultaneous perturbations.

In this example there are two, scalar, unmodelled dynamics perturbations (i.e., complex blocks), so the uncertainty structure is

$$\Delta := \left\{ \begin{bmatrix} \delta_1 & 0 \\ 0 & \delta_2 \end{bmatrix} : \delta_1 \in \mathbf{C}, \delta_2 \in \mathbf{C} \right\}$$

The μ-Tools representation of the uncertainty set, along with the μ calculation syntax are given as follows.

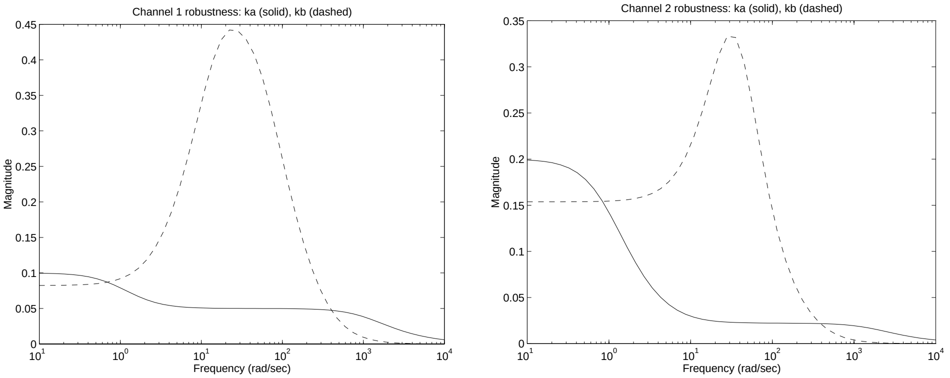


Figure 7-30: Loop-at-a-Time Robustness Tests

```
deltaset = [1 1;1 1];
[bnds_a,dvec_a,sens_a,pvec_a] = mu(Ma_g,deltaset);
[bnds_b,dvec_b,sens_b,pvec_b] = mu(Mb_g,deltaset);
vplot('liv,m',bnds_a,'-',bnds_b,'--');
```

Plots of $\mu_{\Delta}(M(j\omega))$ for both closed-loop systems are shown in Figure 7-31. Note that bnds_a and bnds_b each contain upper and lower bounds for μ, hence there are a total of four plots in Figure 7-31. In the case of two complex uncertainties, the upper and lower bound for μ are guaranteed to be equal, so it appears that there are only 2 plots in the figure.

The peak associated with controller ka is much larger than the peak associated with controller kb. Again, the size of the smallest block-diagonal perturbation which causes instability is equal to

$$\frac{1}{\max_{\omega}\mu_{\Delta}(M(j\omega))}$$

Hence, although the margins to loop-at-a-time perturbations are better with controller ka, the margins with respect to simultaneous perturbations are significantly better using controller kb.

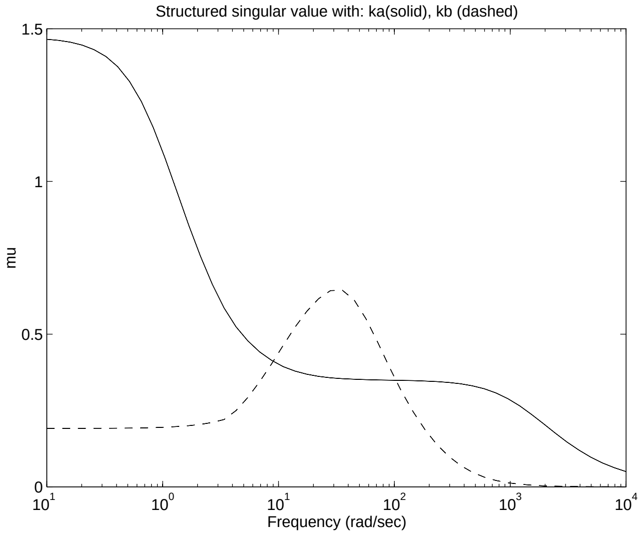


Figure 7-31: Structured Singular Value Plot

Constructing Destabilizing Perturbations

Constructing the smallest *destabilizing* perturbation for a closed-loop system is quite useful. This perturbation can provide physical insight to the types of variations for which the closed-loop system is very sensitive. The command `dypert` is used to construct the smallest destabilizing perturbation from the output of `mu`. The program `dypert` is a block-structured version of `sisorat`. The data required from the `mu` calculation is the perturbation vector, `pvec`, the block structure, `deltaset`, and the bounds for μ, `bnds`. Based on this data, `dypert` creates a real-rational, block structured perturbation that causes instability. The $\|\cdot\|_\infty$ norm of the block diagonal perturbation is equal to 1/(peak value μ) from the associated μ plot. From the μ plot in Figure 7-31, the peak lower bound μ value for `Ma_g` is about 1.47 and for `Mb_g`, 0.64. Therefore, the H_∞ norm of the smallest destabilizing perturbation for `Ma_g` is $\frac{1}{1.47} \approx 0.68$, while for `Mb_g` we get $\frac{1}{0.64} \approx 1.55$. The following commands illustrate this fact.

```
perta = dypert(pvec_a,deltaset,bnds_a);
pertb = dypert(pvec_b,deltaset,bnds_b);
[pkvnorm(sel(bnds_a,1,2)) 1/pkvnorm(sel(bnds_a,1,2))]
ans =
   1.4657  0.6823
[pkvnorm(sel(bnds_b,1,2)) 1/pkvnorm(sel(bnds_b,1,2))]
ans =
   0.6435  1.5540
hinfnorm(perta)
norm between 0.6823 and 0.6829
achieved near 0
hinfnorm(pertb)
norm between 1.554 and 1.556
achieved near 2.182
```

The destabilizing perturbations, perta and pertb, have the same structure as Δ, block-diagonal. Therefore, the diagonal entries of perta and pertb will have norms of 0.68 and 1.55, respectively, with zero off diagonal entries. The following commands verify this fact. The output of these functions is not shown.

```
hinfnorm(sel(perta,1,1))
hinfnorm(sel(perta,1,2))
hinfnorm(sel(perta,2,1))
hinfnorm(sel(perta,2,2))
hinfnorm(pertb)
hinfnorm(sel(pertb,1,1))
hinfnorm(sel(pertb,1,2))
hinfnorm(sel(pertb,2,1))
hinfnorm(sel(pertb,2,2))
```

The perturbed closed-loop can be formed with `starp`. Each perturbation results in a pair of imaginary axis eigenvalues at the frequency associated with the peak (across frequency) of $\mu_\Delta(M(j\omega))$.

```
pertclpa = starp(perta,Ma,2,2);
pertclpb = starp(pertb,Mb,2,2);
rifd(eig(pertclpa))

   real          imaginary         frequency          damping

–1.0201e–04       0.0000e+00        1.0201e–04        1.0000e+00

 1.0394e–14      –1.0000e–01        1.0000e–01       –1.0394e–13

 1.0394e–14       1.0000e–01        1.0000e–01       –1.0394e–13

–1.4313e+00       0.0000e+00        1.4313e+00        1.0000e+00

–1.0808e+03       0.0000e+00        1.0808e+03        1.0000e+00

–1.9334e+03       0.0000e+00        1.9334e+03        1.0000e+00


rifd(eig(pertclpb))

   real          imaginary         frequency          damping

-1.0550e+00       0.0000e+00        1.0550e+00        1.0000e+00

-8.0315e+00      -1.1012e+01        1.3630e+01        5.8927e-01

-8.0315e+00       1.1012e+01        1.3630e+01        5.8927e-01

 2.2808e-12      -3.5565e+01        3.5565e+01       -6.4132e-14

 2.2808e-12       3.5565e+01        3.5565e+01       -6.4132e-14

-9.6571e+01       0.0000e+00        9.6571e+01        1.0000e+00

-3.3746e+02      -9.4865e+01        3.5054e+02        9.6268e-01

-3.3746e+02       9.4865e+01        3.5054e+02        9.6268e-01

-1.2000e+03       0.0000e+00        1.2000e+03        1.0000e+00

-1.8000e+03       0.0000e+00        1.8000e+03        1.0000e+00
```

Space Shuttle Robustness Analysis

This section outlines a robust stability and robust performance analysis of the Space Shuttle lateral axis flight control system during re-entry. It serves as a general illustration of the usefulness of the real and complex μ analysis methods.

The system is a simplified model of the Space Shuttle, in the final stages of landing, as it transitions from supersonic to subsonic speeds. The material in this chapter is based on the paper:

Doyle, J., K. Lenz, and A. Packard, “Design Examples Using μ Synthesis: Space Shuttle Lateral Axis FCS During Re-entry,” *NATO ASI Series, Modelling, Robustness, and Sensitivity Reduction in Control Systems*, vol. 34, Springer-Verlag, 1987.

The analysis procedure involves several steps:

1 Build uncertain model of plant.

2 Define performance specifications and uncertainty bounds.

3 Construct open-loop interconnection.

4 Close feedback loop with controller.

5 Perform a variety of real and complex μ analysis tests on the closed-loop system, and explore the impact of the uncertainty model (real vs. complex) on the robust stability and robust performance requirements.

6 Construct worst-case perturbations, and see their effect on the closed-loop system in the frequency and time domain.

Aircraft Model: Rigid-Body

The rigid body model for the aircraft at Mach 0.9 is a four-state system, with states

$$x = \begin{bmatrix} \beta \\ p \\ r \\ \phi \end{bmatrix} = \begin{bmatrix} \text{sideslip angle (rad)} \\ \text{roll rate (rad/s)} \\ \text{yaw rate (rad/s)} \\ \text{bank angle (rad)} \end{bmatrix}$$

An input/output block diagram of the aircraft is shown in Figure 7-32.

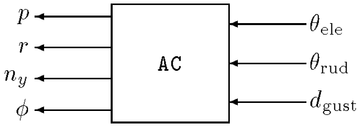


Figure 7-32: Input-Output Description of the Aircraft

The three inputs to the aircraft are denoted by u,

$$u = \begin{bmatrix} \theta_{\text{ele}}\text{(rad)} \\ \theta_{\text{rud}}\text{(rad)} \\ d_{\text{gust}}\text{(ft/sec)} \end{bmatrix}$$

The first input is the actual angular deflection of the elevon surface. The second is the actual deflection of the rudder surface. Finally, there is a lateral wind gust disturbance input, due to the winds that occur at this altitude.

There are four output variables of the aircraft. Three of these are states, while the fourth is the lateral acceleration at the pilot's location, denoted n_y (units of n_y are ft/sec^2).

$$y = \begin{bmatrix} p \\ r \\ n_y \\ \phi \end{bmatrix}$$

All variables in y are measured with inertial devices (gyroscopes and accelerometers) whose individual noise characteristics are discussed later.

Aircraft Model: Aerodynamic Uncertainty

The major source of uncertainty in the aircraft model (AC) is in the aerodynamic coefficients. These are standard aerodynamic parameters which express incremental forces and torques generated by incremental changes in sideslip, elevon, and rudder angles. This is a linear relationship, expressed as

$$\begin{bmatrix} \text{side force} \\ \text{yawing moment} \\ \text{rolling moment} \end{bmatrix} = \begin{bmatrix} c_{y\beta} & c_{ya} & c_{yr} \\ c_{\eta\beta} & c_{\eta a} & c_{\eta r} \\ c_{l\beta} & c_{la} & c_{lr} \end{bmatrix} \begin{bmatrix} \beta \\ \theta_{\text{ele}} \\ \theta_{\text{rud}} \end{bmatrix}$$

The coefficients $c_{\bullet\bullet}$ are typically estimated based on theoretical predictions, numerical calculations, experiments in wind tunnels, and flight tests. At Mach 0.9, the shuttle is in a transonic regime involving a combination of subsonic and supersonic flows. Theoretical, computational, and wind tunnel techniques are inaccurate at this flight condition, so with extremely limited flight data (early in the shuttle program), the coefficient uncertainty for the shuttle model is unusually large.

Uncertainty in these coefficients is modeled as a nominal value, plus a perturbation.

$$\begin{bmatrix} c_{y\beta} & c_{ya} & c_{yr} \\ c_{\eta\beta} & c_{\eta a} & c_{\eta r} \\ c_{l\beta} & c_{la} & c_{lr} \end{bmatrix} = \begin{bmatrix} \bar{c}_{y\beta} & \bar{c}_{ya} & \bar{c}_{yr} \\ \bar{c}_{\eta\beta} & \bar{c}_{\eta a} & \bar{c}_{\eta r} \\ \bar{c}_{l\beta} & \bar{c}_{la} & \bar{c}_{lr} \end{bmatrix} + \begin{bmatrix} r_{y\beta}\delta_{y\beta} & r_{ya}\delta_{ya} & r_{yr}\delta_{yr} \\ r_{\eta\beta}\delta_{\eta\beta} & r_{\eta a}\delta_{\eta a} & r_{\eta r}\delta_{\eta r} \\ r_{l\beta}\delta_{l\beta} & r_{la}\delta_{la} & r_{lr}\delta_{lr} \end{bmatrix}$$

where the values of the $r_{\bullet\bullet}$ are

$$\begin{bmatrix} r_{y\beta} & r_{ya} & r_{yr} \\ r_{\eta\beta} & r_{\eta a} & r_{\eta r} \\ r_{l\beta} & r_{la} & r_{lr} \end{bmatrix} = \begin{bmatrix} 2.19 & -1.33 & -0.37 \\ -1.52 & 1.35 & 0.87 \\ -0.72 & 0.52 & 0.24 \end{bmatrix}$$

and the perturbations $\delta_{\bullet\bullet}$ are assumed to be fixed, unknown, real parameters, with each satisfying $|\delta_{\bullet\bullet}| \leq 1$. We use the notation $r_{\bullet\bullet} .* \delta_{\bullet\bullet}$ to denote the 3 × 3 perturbation matrix in the model for the aero coefficients, $c_{\bullet\bullet}$.

The aircraft model acnom has the nominal aerodynamic coefficients absorbed into the state-space data. In addition to the inputs μ and outputs y described earlier, acnom has three fictitious inputs and outputs such that the uncertain behavior of the aircraft AC is given by the linear fractional transformation in Figure 7-33.

The state-space model for acnom is created by the M-file mk_acnom. A listing of state-space model acnom is given in “Shuttle Rigid Body Model” at the end of this section.

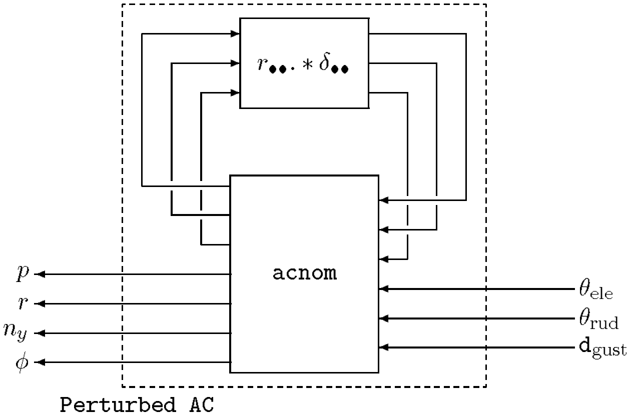
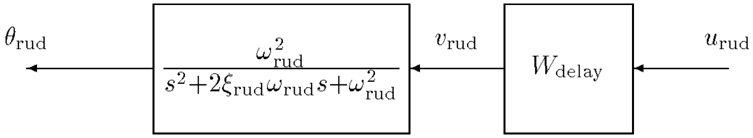


Figure 7-33: Uncertain Aircraft Model

Actuator Models

The aircraft has two controlled inputs, rudder command, and elevon command. Each actuator is modeled with a second order transfer function, as well as a second order delay approximation to model the effects of the digital implementation.

The model for the rudder is


$\theta_{\rm rud}$
$\frac{\omega_{\rm rud}^2}{s^2+2\xi_{\rm rud}\omega_{\rm rud}s+\omega_{\rm rud}^2}$
$v_{\rm rud}$
$W_{\rm delay}$
$u_{\rm rud}$

Here, u_{rud} is the electrical command that the controller will generate to move the rudder. The transfer function W_{delay} is a second order approximation of a delay, to model the effects of the digital implementation of the control system. In particular

$$W_{\text{delay}}(s) = \frac{1 - 2\xi_{\text{del}}(s/\omega_{\text{del}}) + (s^2/\omega_{\text{del}}^2)}{1 + 2\xi_{\text{del}}(s/\omega_{\text{del}}) + (s^2/\omega_{\text{del}}^2)}$$

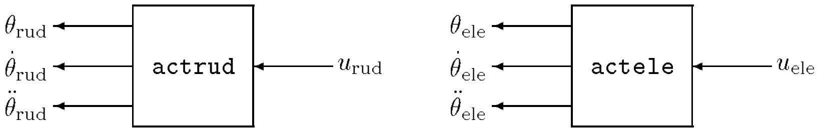
with ω_{del} = 173 rad/s, and ξ_{del} = 0.866. The transfer function

$$\frac{\omega_{\text{rud}}^2}{s^2 + 2\xi_{\text{rud}}\omega_{\text{rud}}s + \omega_{\text{rud}}^2}$$

models the physical devices (motors, inertias, etc.) involved in actually moving the rudder. The variable θ_{rud} is the actual deflection of the rudder surface, while u_{rud} represents the command to the rudder system. The values of the parameters are ξ_{rud} = 0.75, ω_{rud} =21 rads/sec.

A similar model is used for the elevon actuation system. The parameters in that case are ξ_{ele} = 0.72, ω_{ele} =14 rads/sec, with an identical second order delay model.

The state-space models for the actuators are created by the M-file `mk_act`. Since the closed-loop performance objectives include penalties on the deflections, rates, and accelerations of the control surfaces, the state-space models created in mk_act each have three outputs, as shown below.



θ_{rud}
$\dot{\theta}_{\mathrm{rud}}$
$\ddot{\theta}_{\mathrm{rud}}$
actrud
u_{rud}
θ_{ele}
$\dot{\theta}_{\mathrm{ele}}$
$\ddot{\theta}_{\mathrm{ele}}$
actele
u_{ele}

Exogenous Disturbances, Noises, and Commands

There are three sources of exogenous signals:

- Wind gusts
- Sensor noise
- Pilot bank-angle command

In the H_∞ framework, all time domain signals are modeled as the unit ball in L_2, filtered by problem dependent weighting functions which reflect typically occurring signals in the application. In addition to the L_2 gain, the H_∞ norm also has an interpretation in terms of gain from sinusoids to sinusoids. Now, suppose h represents one of the exogenous signals, and W_h is the associated stable weighting function. Then, the signal h is assumed to be any signal from the set

$$h \in \{W_h \eta_h : \|\eta_h\|_2 \le 1\}$$

By choosing the form of $W_h(s)$, the spectral content of such signals h can be shaped.

- **Lateral Wind Gusts:** The set of lateral wind gusts is modeled as

$$d_{\mathrm{gust}} \in \left\{ W_{\mathrm{gust}} \eta_{\mathrm{gust}} : W_{\mathrm{gust}} = 30\frac{1 + s/2}{1 + s}, \left\|\eta_{\mathrm{gust}}\right\|_2 \le 1 \right\}.$$

The set on the right-hand side of the equation models the typical wind gusts that the shuttle will encounter at this flight condition.

- **Sensor Noise:** Each measurement is corrupted with sensor noise which becomes more severe with increasing frequency. Since p and r are measured with comparable gyroscopes, their sensor noise weights are identical,

$$W_P = W_r = 0.0003\frac{1 + s/0.01}{1 + s/0.5}$$

These weighting functions imply a low frequency measurement error in p and r of 0.0003 rads/sec, and a high frequency error of 0.015 rads/sec. The model of the measured value of p, denoted p_{meas}, is given by

$$p_{\text{meas}} = p + W_p\eta_p$$

where η_p is an arbitrary signal, with $\|\eta_p\|_2 \le 1$. This type of weighted, additive L_2 sensor noise is assumed for each of the 4 measured variables.

The measurement of ϕ is obtained from a navigation package at a reduced sample rate, so its weight is chosen to be

$$W_\phi = 0.0007\frac{1 + s/0.01}{1 + s/2}$$

which is relatively large in the mid-to-high frequency range. The sensor noise weight on the n_y accelerometer is

$$W_{n_y} = 0.25\frac{1 + s/0.05}{1 + s/10}$$

For the variables r, ϕ, and n_y, we have

$$\begin{aligned} r_{\text{meas}} &= r + W_r\eta_r \\ \phi_{\text{meas}} &= \phi + W_\phi\eta_\phi \\ n_{y_{\text{meas}}} &= n_y + W_{n_y}\eta_{n_y} \end{aligned}$$

- **Pilot Bank-Angle Command:** In this problem, the pilot (or autopilot) takes the shuttle through a series of sweeping “S” turns to slow the vehicle down.

These require accurate tracking of a bank-angle command. Typical bank-angle commands are modeled as

$$\phi_{\mathrm{cmd}} = W_{\phi\mathrm{cmd}}\eta_{\phi\mathrm{cmd}},$$

where $\eta_{\phi\mathrm{cmd}}$ is assumed to be an arbitrary signal with $\|\eta_{\phi\mathrm{cmd}}\|_2 \leq 1$. In this example, the weight on the bank-angle command is chosen as

$$W_{\phi\mathrm{cmd}} := 0.5\frac{1 + s/2}{1 + s/0.5}$$

The particular choice roughly implies that the bank-angle commands are dominated by low frequency signals, with a maximum magnitude of approximately 0.5 radians.

The noise weighting functions are denoted by $W_{\mathrm{noise}} = \mathrm{diag}\{W_p, W_r, W_\phi, W_{n_y}\}$ in the control block diagram.

Errors

There are several variables that are to be kept small in the face of the exogenous signals listed in the previous section. In this context, these variables will be considered *errors*.

Actuator signal levels: the angular position, angular rates, and angular accelerations of the rudder ($\cdot_{\mathrm{rud}}$) and elevon ($\cdot_{\mathrm{ele}}$) surfaces should remain reasonably small in the face of the exogenous signals. The signals are weighted to give an actuator error vector of

$$e_{\mathrm{act}} = \begin{bmatrix} e_e \\ e_{\dot{e}} \\ \ddot{e}_e \\ e_r \\ e_{\dot{r}} \\ \ddot{e}_r \end{bmatrix} := \begin{bmatrix} 4\ \theta_{\mathrm{ele}} \\ \dot{\theta}_{\mathrm{ele}} \\ 0.005\ \ddot{\theta}_{\mathrm{ele}} \\ 2\ \theta_{\mathrm{rud}} \\ 0.2\ \dot{\theta}_{\mathrm{rud}} \\ 0.009\ \ddot{\theta}_{\mathrm{rud}} \end{bmatrix}$$

This performance specification can be loosely interpreted as a requirement that the closed-loop system should, under the excitation of the modeled exogenous signals, maintain θ_{ele} to below 0.25 radians, $\dot{\theta}_{ele}$ to below 1 rad/sec, $\ddot{\theta}_{ele}$ to below 200 rads/sec^2, and so on for the rudder variables. For notational purposes, let W_{act} be the 6 × 6 constant matrix so that

$$e_{act} = W_{act}\begin{bmatrix} \theta_{ele} \\ \dot{\theta}_{ele} \\ \ddot{\theta}_{ele} \\ \theta_{rud} \\ \dot{\theta}_{rud} \\ \ddot{\theta}_{rud} \end{bmatrix}$$

- **Performance variables:**
  - The ideal bank angle response (ϕ_{ideal}) of the shuttle to a bank-angle command (ϕ_{cmd}) is

    $$\phi_{ideal} := \frac{1}{1 + 2 \cdot \xi(s/\omega) + (s/\omega)^2}\phi_{cmd}$$

    where $\omega = 1.2$ rad/sec, and $\xi = 0.7$. The bank-angle tracking error is defined as $\phi - \phi_{ideal}$.
  - Turn coordination: in an ideal turn, the bank angle, and the yaw rate are related. For this aircraft, a turn coordination error is defined as

    $r_p := r - 0.037\phi$
  - In a turn, it is desired that the pilot feel very little lateral acceleration, hence, the lateral acceleration variable, n_y, is an error.

    These error signals are weighted by frequency dependent weights to give a performance error vector as

$$e_{\text{perf}} := \begin{bmatrix} 0.8\frac{1+s}{1+s/0.1} & 0 & 0 \\ 0 & 500\frac{1+s}{1+s/0.01} & 0 \\ 0 & 0 & 250\frac{1+s}{1+s/0.01} \end{bmatrix} \begin{bmatrix} n_y \\ r - 0.037\phi \\ \phi - \phi_{\text{ideal}} \end{bmatrix}$$

For notational purposes, let W_{perf} be a 3 × 5 transfer function matrix so that

$$e_{\text{perf}} = W_{\text{perf}} \begin{bmatrix} p \\ r \\ n_y \\ \phi \\ \phi_{\text{ideal}} \end{bmatrix}$$

The error weight on the lateral acceleration indicates a tolerance for low frequency accelerations of 1.25 ft/sec^2, which is relaxed at high frequency, allowing accelerations up to 12.5 ft/sec^2. Again, these specifications correspond to n_y errors produced by the exogenous signal set (wind gusts, measurement noises, and bank angle commands). Similar interpretation is given to the other performance variables.

LFT Aero-Coefficient Uncertainty

The perturbations in the aero-coefficients can be written as an LFT (linear fractional transformation) on a structured uncertainty matrix. Define constant matrices $W_L \in \mathbf{R}^{3\times 9}$ and $W_R \in \mathbf{R}^{9\times 3}$ such that

$$W_L \cdot \operatorname{diag}[\delta_{y\beta}, \delta_{\eta\beta}, \delta_{l\beta}, \delta_{ya}, \ldots, \delta_{lr}] \cdot W_R = \begin{bmatrix} r_{y\beta}\delta_{y\beta} & r_{ya}\delta_{ya} & r_{yr}\delta_{yr} \\ r_{\eta\beta}\delta_{\eta\beta} & r_{\eta a}\delta_{\eta a} & r_{\eta r}\delta_{\eta r} \\ r_{l\beta}\delta_{l\beta} & r_{la}\delta_{la} & r_{lr}\delta_{lr} \end{bmatrix}$$

for all $\delta_{\ldots}$. This is easily done with the permutation matrices W_L and W_R shown below.

$$W_R^T = \begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \end{bmatrix}$$

$$W_L = \begin{bmatrix} 2.19 & 0 & 0 & -1.33 & 0 & 0 & -0.37 & 0 & 0 \\ 0 & -1.52 & 0 & 0 & 1.35 & 0 & 0 & 0.87 & 0 \\ 0 & 0 & -0.72 & 0 & 0 & 0.52 & 0 & 0 & 0.24 \end{bmatrix}$$

Create Open-Loop Interconnection

The aircraft model, actuator models, and weighting functions discussed in the previous sections can be constructed from M-files.

```
mk_acnom;
mk_act;
mk_wts;
```

The open-loop interconnection structure, which includes the uncertainty model and the performance objectives, is shown in Figure 7-34.

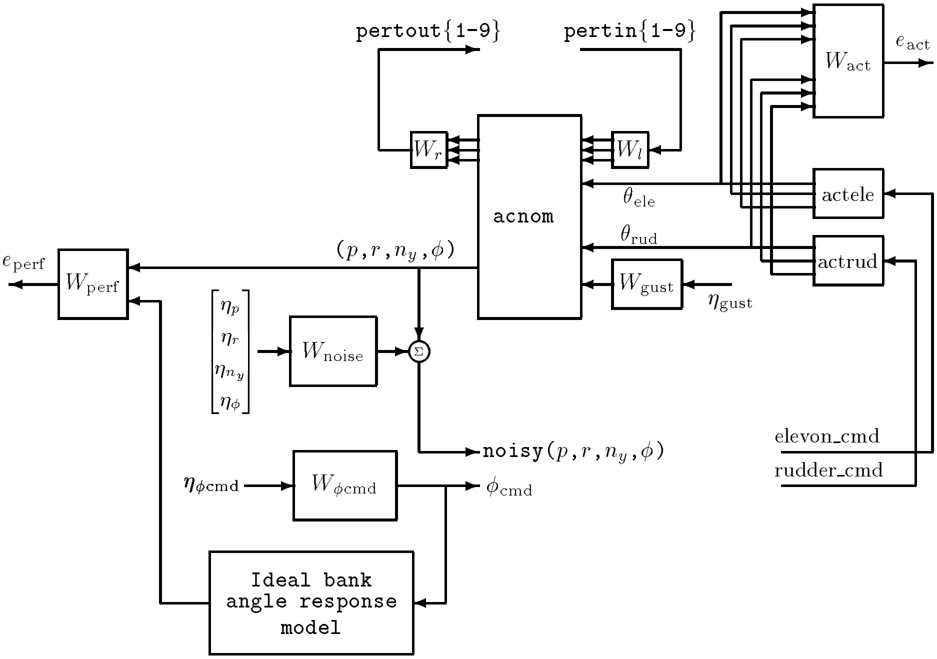


Figure 7-34: Shuttle Interconnection Structure

The M-file `mk_olic` uses the `sysic` command to create a SYSTEM matrix description of the open-loop interconnection structure. In the workspace, the open-loop system is denoted by `olic`, and has 23 states, 23 outputs, and 17 inputs.

```
mk_olic;
minfo(olic)
```

A schematic diagram, with the specific input/output ordering for `olic`, is shown in Figure 7-35.

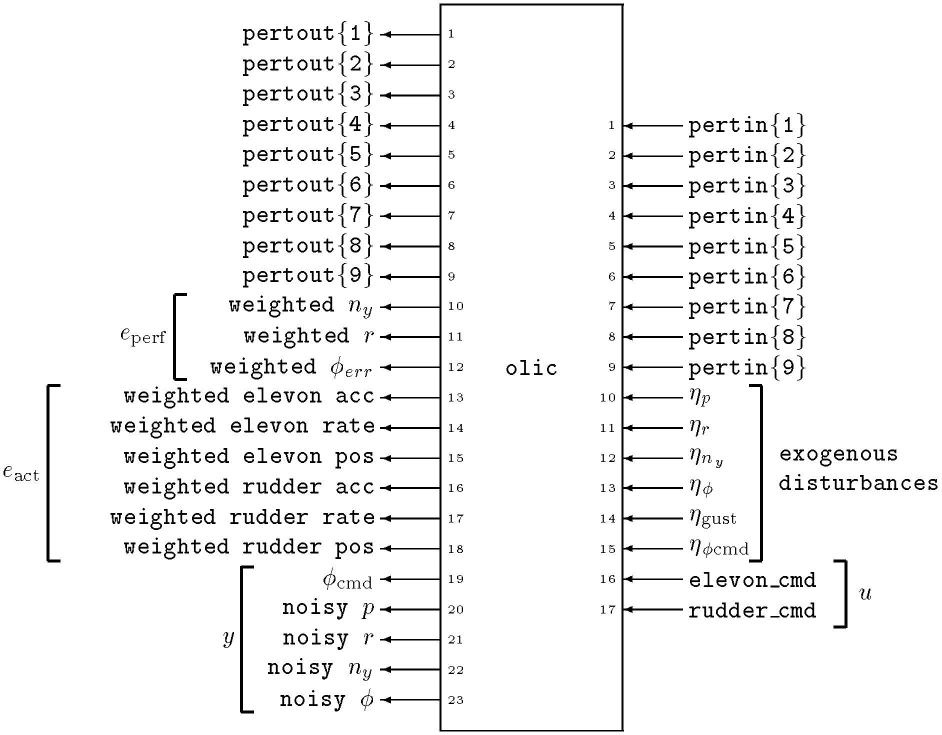
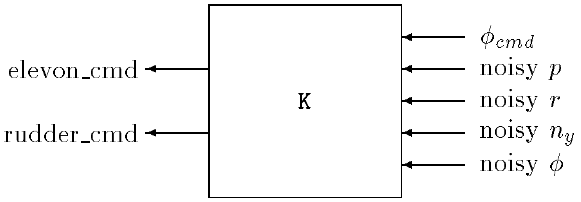


Figure 7-35: Schematic Diagram of Space Shuttle olic

Controllers

In this section, the robustness properties of three different controllers are analyzed using μ. The controllers receive four sensor measurements along with the ϕ command signal and produce two control signals for the elevon and rudder commands. The controller block diagram is shown below.



In this example, each controller has different characteristics:

- k_h is designed to optimize H_∞ performance, under the assumption that there is no model uncertainty;
- k_mu is designed with the $D - K$ iteration approach to μ -synthesis;
- k_x is constructed to be a tradeoff between the two controllers

The μ-Tools commands to design the H_∞ optimal controller, k_h, are:

```
olic_h = sel(olic,[10:23],[10:17]);
minfo(olic_h)
k_h = hinfsyn(olic_h,5,2,0,5,0.1);
```

The first command, sel, removes the aero-coefficient uncertainty channels, leaving only the exogenous signals and errors, and feedback signals. The third command, hinfsyn, designs a suboptimal H_∞ controller for the open-loop system olic_h. This controller measures five signals, and generates two control signals.

It is simple to check some characteristics of the controller and the closed-loop system

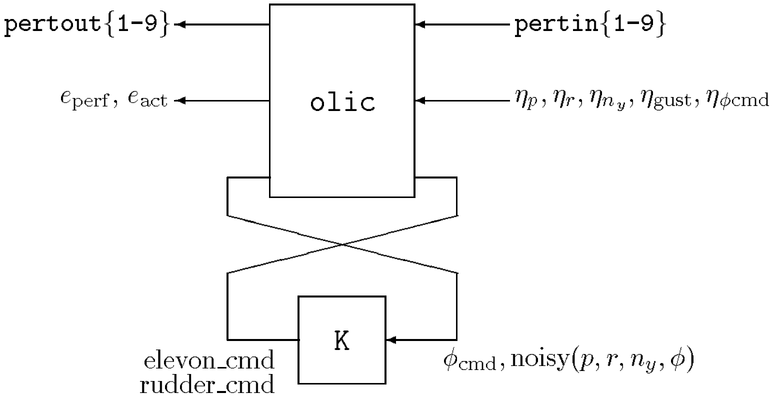
```
minfo(k_h)
clp_h = starp(olic,k_h,5,2);
rifd(spoles(clp_h))
rifd(spoles(k_h))
```

The two other controllers have already been designed and stored in the file `shutcont.mat`.

```
load shutcont
minfo(k_x)
minfo(k_mu)
```

Nominal Frequency Responses

The closed-loop system is constructed using the star product command `starp`.


pertout{1-9}
pertin{1-9}
$e_{\mathrm{perf}}, e_{\mathrm{act}}$
olic
$\eta_p, \eta_r, \eta_{n_y}, \eta_{\mathrm{gust}}, \eta_{\phi\mathrm{cmd}}$
K
elevon_cmd
rudder_cmd
ϕ_{cmd}, noisy(p, r, n_y, ϕ)

In the closed-loop system, there are six exogenous signals (the six η signals: four sensor noises, wind gust, bank angle command) and nine errors (weighted performance error vector and the weighted actuator error vector). The nominal performance objective is that this multivariable transfer function matrix should have an H_∞ norm less than 1. Using μ-Tools, it is easy to evaluate this performance criterion. Simply form the closed-loop system, calculate its frequency response, and plot the norm of the appropriate transfer function versus frequency.

```
omega = logspace(-2,3,30);
clp_h = starp(olic,k_h,5,2);
clp_hg = frsp(clp_h,omega);
clp_x = starp(olic,k_x,5,2);
minfo(clp_x)
clp_xg = frsp(clp_x,omega);
minfo(clp_xg)
clp_mu = starp(olic,k_mu,5,2);
clp_mug = frsp(clp_mu,omega);
```

Note that the closed-loop systems have additional inputs and outputs from the nine aero-perturbation channels. The relevant exogenous signals and errors are selected (using sel) before calculating the maximum singular value (vnorm).

```
np_hg = sel(clp_hg,[10:18],[10:15]);
np_xg = sel(clp_xg,[10:18],[10:15]);
np_mug = sel(clp_mug,[10:18],[10:15]);
vplot('liv,m',vnorm(sel(clp_hg,10:18,10:15)),...
   vnorm(sel(clp_xg,10:18,10:15)),...
   vnorm(sel(clp_mug,10:18,10:15)))
title('NOMINAL PERFORMANCE: ALL CONTROLLERS')
```

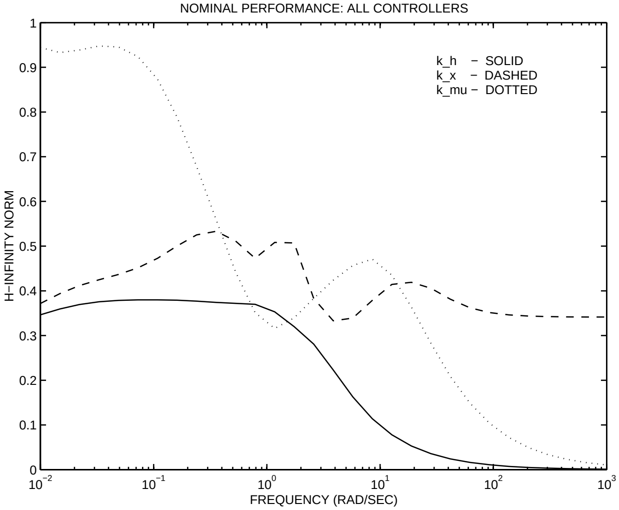


Figure 7-36: Nominal Performance of k_h**,** k_x**, and** k_mu

Note that the best nominal performance is achieved by controller k_h, as seen in Figure 7-36. This is not surprising, since it was designed specifically with these disturbances and errors in mind. Relatively, the performance from k_mu is poor, though it does meet the nominal performance objective. In later calculations, it will become clear that the degradation in nominal performance is offset by a much greater insensitivity to variations in the aerodynamic coefficients.

Robust Stability

Using μ, the robust stability characteristics of each closed-loop system can be evaluated. The uncertain parameters ($\delta_{y\beta}$,. . .,δ_{lr}) can be assumed to be real, representing uncertainty in the constant aerodynamic coefficients. However, the flow around the vehicle is very complex, and the quasi-steady implication of constant aerodynamic coefficients is somewhat simplistic. Consequently, for a more conservative analysis, the uncertain parameters can be treated as complex. In this section, both models of uncertainty will be analyzed, and compared. Refer to Chapter 4, “Modeling and Analysis of Uncertain Systems” for more detail on the interpretations.

This motivates two separate representations of the uncertainty set,

$$\Delta_{\mathbf{C}} := \{\mathrm{diag}[\delta_1,\delta_2,\ldots,\delta_9] : \delta_i \in \mathbf{C}\}$$

$$\Delta_{\mathbf{R}} := \{\mathrm{diag}[\delta_1,\delta_2,\ldots,\delta_9] : \delta_i \in \mathbf{R}\}$$

where the perturbations represent uncertainty in the aero-coefficients. Note that the first set contains all complex perturbations, while the second set includes only real perturbations.

In the μ-Tools syntax, these are represented as

```
delsetrs_C = ones(9,2);
delsetrs_R = [-ones(9,1) zeros(9,1)];
```

Here, the lower case rs refers to robust stability (as opposed to robust performance, rp, which will be addressed later).

The perturbation inputs/outputs from the frequency responses are selected for a robust stability μ test. The input/output channels associated with the performance criterion are not used in the robust stability μ test. A diagram of the closed-loop system is shown in Figure 7-37.

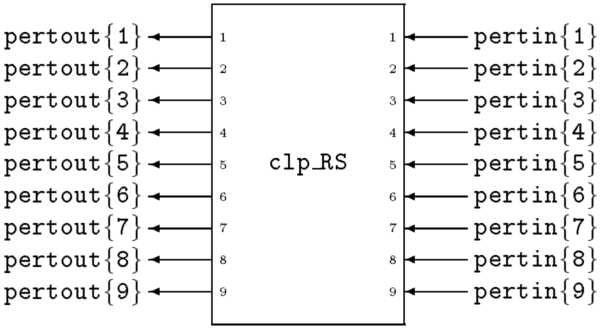


Figure 7-37: Schematic Design of clp_RS

```
clp_hgRS = sel(clp_hg,1:9,1:9);
clp_xgRS = sel(clp_xg,1:9,1:9);
clp_mugRS = sel(clp_mug,1:9,1:9);
```

Calculate μ across frequency, and look at μ plots. Start with the complex uncertainty structure.

```
[bnds_h,dv_h,sens_h,rp_h]=mu(clp_hgRS,delsetrs_C);
[bnds_x,dv_x,sens_x,rp_x]=mu(clp_xgRS,delsetrs_C);
[bnds_mu,dv_mu,sens_mu,rp_mu]=mu(clp_mugRS,delsetrs_C);
vplot('liv,d',bnds_h,'-',bnds_x,'--',bnds_mu,'-.')
title('ROBUST STABILITY OF CLOSED-LOOP: COMPLEX')
```

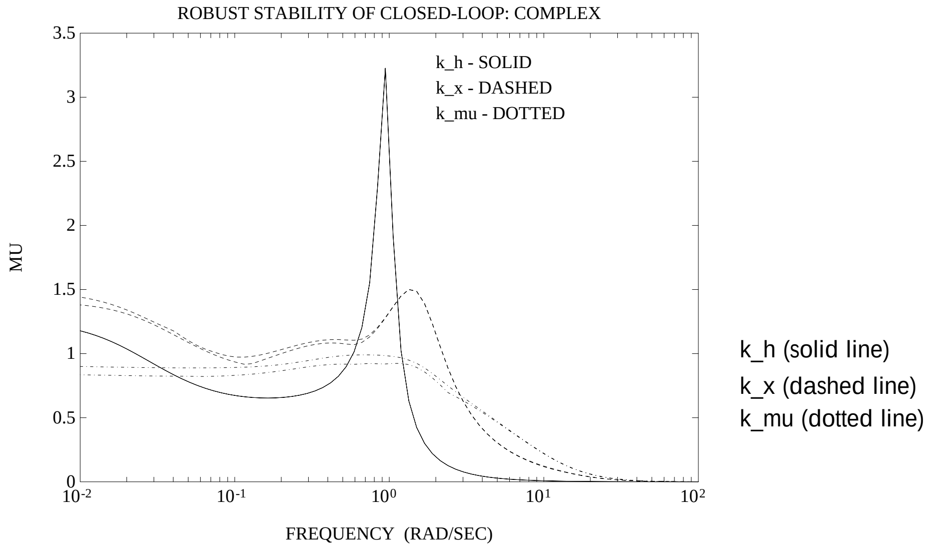


Figure 7-38: Complex Robust Stability μ Analysis of k_h, k_x, and k_mu

According to Figure 7-38, the k_mu controller has the best robust stability properties when the perturbations are treated as complex (dynamic). The peak of the lower bound, 0.9, implies that there is a diagonal complex perturbation of size, $\frac{1}{0.9}$, that causes instability. The peak of the upper bound, approximately 0.99, implies that for diagonal perturbations smaller than $\frac{1}{0.99}$, the closed-loop system remains stable. The gap between the upper and lower bound can be reduced by using the “c” option in the mu command. Without this option, the upper bound from mu is a computational approximation to

$$\inf_{D \in \mathbf{D}} \bar{\sigma}(DMD^{-1})$$

that can be refined (option “c”) at the expense of slower execution. Using the “c” option reduces the upper bound peak to 0.9, so that the complex μ analysis gives a tight estimate on the size of the smallest destabilizing perturbation.

Similar interpretations are possible for the closed-loop systems with controllers k_h and k_x, though, since the μ plots have larger peaks, the bound on allowable perturbations is smaller. Hence, the closed-loop system with the controller k_mu achieves robust stability to complex perturbations, whereas the other controllers do not.

The closed-loop system is also analyzed treating the aerodynamic uncertainty as real perturbations with the μ-Tools mu command.

```
[rbnds_h,rrp_h] = mu(clp_hgRS,delsetrs_R);
[rbnds_x,rrp_x] = mu(clp_xgRS,delsetrs_R);
[rbnds_mu,rrp_mu] = mu(clp_mugRS,delsetrs_R);
vplot('liv,d',rbnds_h,'-',rbnds_x,'--',rbnds_mu,'-.')
title('ROBUST STABILITY OF CLOSED-LOOP: REAL')
```

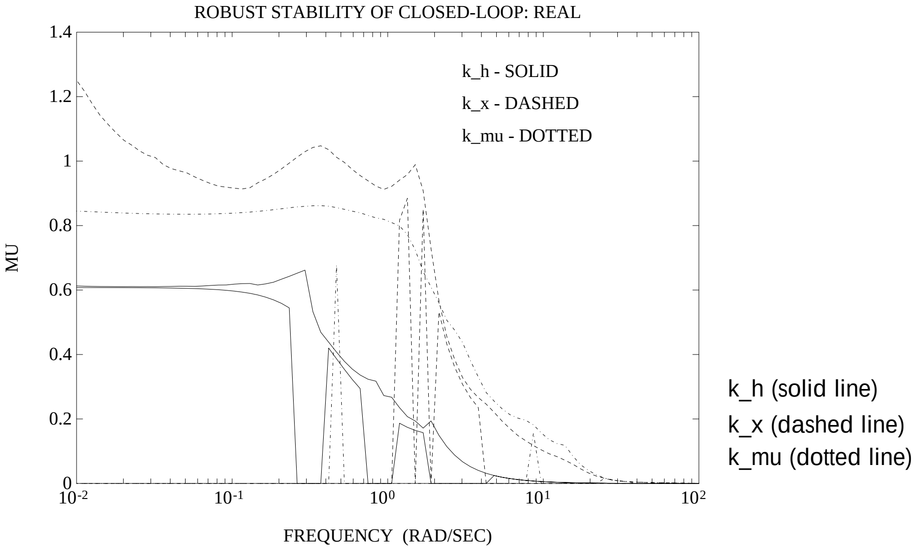


Figure 7-39: Real Robust Stability μ Analysis of k_h, k_x, and k_mu

The k_h controller has the best robust stability properties, when the perturbations are treated as real, as seen in Figure 7-39. This is in contrast to the robust stability analysis with complex perturbation where k_mu exhibited the best properties. The peak of the upper bound, approximately 0.66, implies that for diagonal, real perturbations smaller than $\frac{1}{0.66}$, the closed-loop system remains stable. The lower bound in Figure 7-39 is often 0 and does not converge for all values of frequency, leading to a large gap between the upper and lower bound. This gap can be reduced by adding a small amount of complex perturbation to the pure real perturbation. A detailed discussion of this approach can be found in the “Mixed Real/Complex Structured Singular Value” section in Chapter 4.

Robust Performance

Using μ, the robust performance characteristics of each closed-loop system can be evaluated. The uncertain parameters are treated as real parameters in this analysis. These parameters can also be treated as complex perturbations, though this is not done in this section.

The appropriate block structure for the robust performance test is

$$\Delta_P := \{\mathrm{diag}[\delta_1,\delta_2,...,\delta_9,\Delta_{10}] : \delta_i \in \mathbf{R}, \Delta_{10} \in \mathbf{C}^{6\times 9}\}$$

which is simply an augmentation of the original real robust stability uncertainty set, Δ_R, with a complex 6 × 9 full block to include the performance objectives. Recall from the “Using m to Analyze Robust Performance” section in Chapter 4: H_∞ performance objectives are always represented with a full, complex block. Hence,

```
delsetrp_R = [delsetrs_R;6 9]
```

The μ calculations are performed on the entire 18 × 15 closed-loop matrix, which includes the perturbation channels and the exogenous signals and errors. The command mu is called with both real and complex blocks.

```
[bnds_h,ph] = mu(clp_hg,delsetrp_R);
[bnds_x,px] = mu(clp_xg,delsetrp_R);
[bnds_mu,p_mu] = mu(clp_mug,delsetrp_R);
vplot('liv,d',bnds_h,'-',bnds_x,'--',bnds_mu,'-.')
title('ROBUST PERFORMANCE OF ALL CONTROLLERS')
```

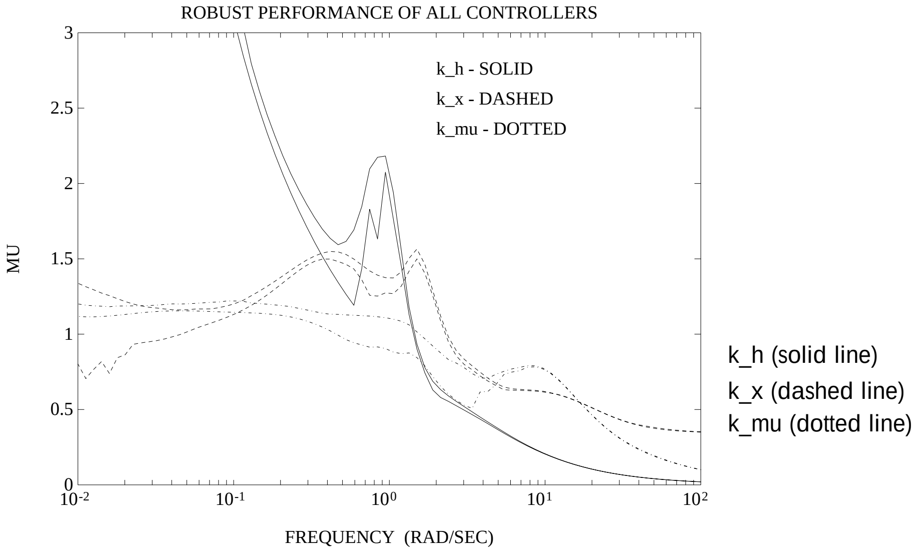


Figure 7-40: Robust Performance μ Plots of k_h**,** k_x**, and** k_mu

The axis is selected in Figure 7-40 to show a comparison of controllers k_x and k_mu. At low frequency, the closed-loop robust performance with k_h implemented gets as bad as 14. The closed-loop system using controller k_x achieves a robust performance μ value of 1.56, while controller k_mu achieves a robust performance μ value of 1.22.

Worst-case Perturbations

Using a μ calculation, we have seen that all controllers achieve robust-stability to the 9 × 9 real uncertainty matrix which represents uncertainty in the aero-coefficients. However, the performance of each closed-loop system degrades differently under LFT real, diagonal perturbations. We use wcperf to compute the worst-case performance degradation as well as the worst-case, norm 1, perturbation. The worst-case perturbation of norm 1 will be used in the next section for uncertain time-domain simulations.

```
[deltabadh,wcp_lowh,wcp_upph] = wcperf(clp_hg,delsetrs_R,.05,4);
[deltabadx,wcp_lowh,wcp_upph] = wcperf(clp_xg,delsetrs_R,.05,10);
[deltabadmu,wcp_lowmu,wcp_uppmu]= wcperf(clp_mug,delsetrs_R,.05,10);
vplot(wcp_lowh,wcp_upph,wcp_lowx,wcp_uppx,wcp_lowmu,wcp_uppmu)
```

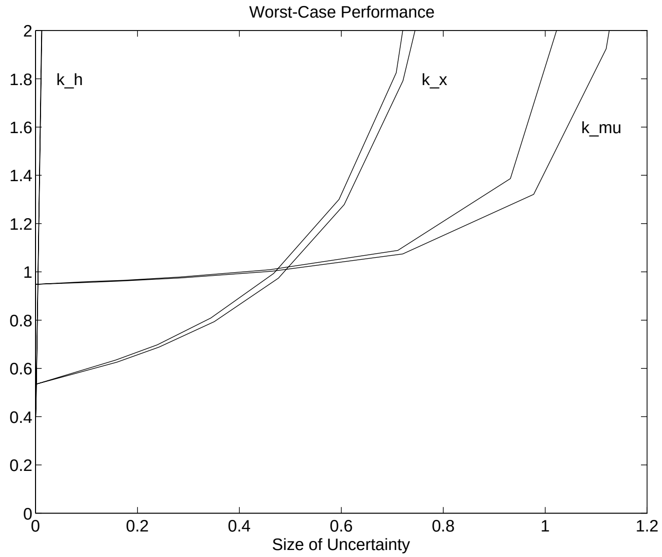


Figure 7-41: Performance Degradation of Closed-Loop

Using k_h, it is clear that the closed-loop performance degrades rapidly and severely. It would not be an acceptable controller in the real aircraft.

Note Optimizing the H_∞ norm of some closed-loop transfer function *does not, in any way, guarantee robustness* to perturbations at other points in the feedback loop.

Using k_x and k_mu, reasonable robustness properties (on the order of the original specifications) are attained. The controller k_x achieves better nominal performance (i.e., at $\|\Delta\| = 0$), at the expense of more rapid potential performance degradation under uncertainty. Both closed-loop systems potentially degrade to unacceptable (performance norm > 1) performance with less than one-half of the original modeled uncertainty. At that level of uncertainty, the closed-loop system with k_mu degrades more gracefully. This type of tradeoff curve illustrates some of the differences between the two controllers, and can be helpful in understanding the tradeoffs involved.

Time Simulations

The open-loop simulation interconnection, Figure 7-42, is similar to `olic`, but contains none of the weighting functions. It is used exclusively for nominal and perturbed time-domain simulations, where unweighted time signals will be calculated and plotted.

```
mk_olsim;
minfo(olsim)
```

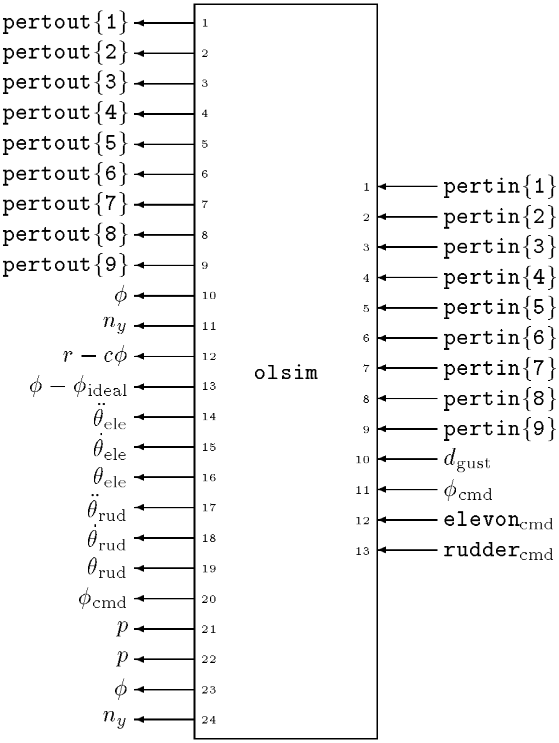
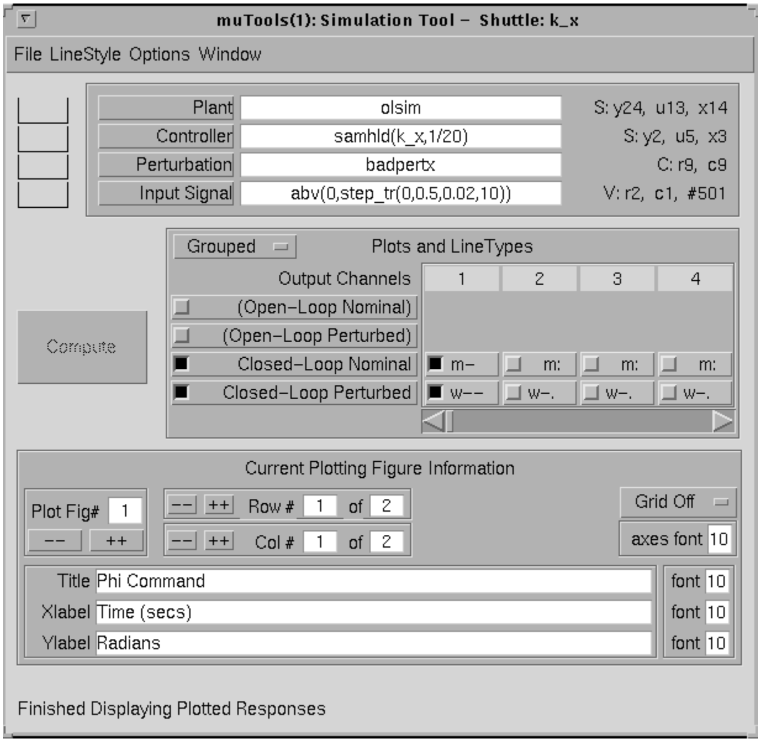


Figure 7-42: Open-Loop Simulation Model

For the purposes of this exercise, the four sensor noises have been eliminated from the simulation model. It is easy to modify `mk_olsim.m` to include these if desired.

The LFT time simulation GUI, `simgui`, is used to simulate the nominal and perturbed time response of the three controllers. For more details on `simgui` see the “LFT Time Simulation User Interface Tool: simgui” section in Chapter 6.

The main performance objective is bank angle tracking, so the response to a 0.5 radian step input for ϕ_{cmd} is investigated. The gust input is set to zero in these simulations. This data is entered into the `simgui` Main window `Input Signal` editable text. Note that ϕ_{cmd} is the 11th input of `olsim` and the second non-perturbation input. The output signals of interest are ϕ, n_y, $r - c\phi$, and $\phi - \phi_{ideal}$, which are outputs 10 through 13 of `olsim` or the first through fourth outputs after the perturbation has been included. In the `simgui` Main window, input `olsim` into the `Plant` editable text. Figure 7-43 shows the main simulation window for the nominal and perturbed response of controller `k_x`.


File LineStyle Options Window
Compute
Plots and LineTypes
Current Plotting Figure Information
-- ++

muTools(1): Simulation Tool – Shuttle: k_x

Plant	olsim	S: y24, u13, x14
Controller	samhld(k_x,1/20)	S: y2, u5, x3
Perturbation	badpertx	C: r9, c9
Input Signal	abv(0,step_tr(0,0.5,0.02,10))	V: r2, c1, #501

Grouped

Output Channels	1	2	3	4
□ (Open–Loop Nominal)				
□ (Open–Loop Perturbed)				
■ Closed–Loop Nominal	■ m–	□ m:	□ m:	□ m:
■ Closed–Loop Perturbed	■ w--	□ w–.	□ w–.	□ w–.

Plot Fig# 1

Row # 1 of 2

Col # 1 of 2

Grid Off

axes font 10

Title Phi Command font 10

Xlabel Time (secs) font 10

Ylabel Radians font 10

Finished Displaying Plotted Responses

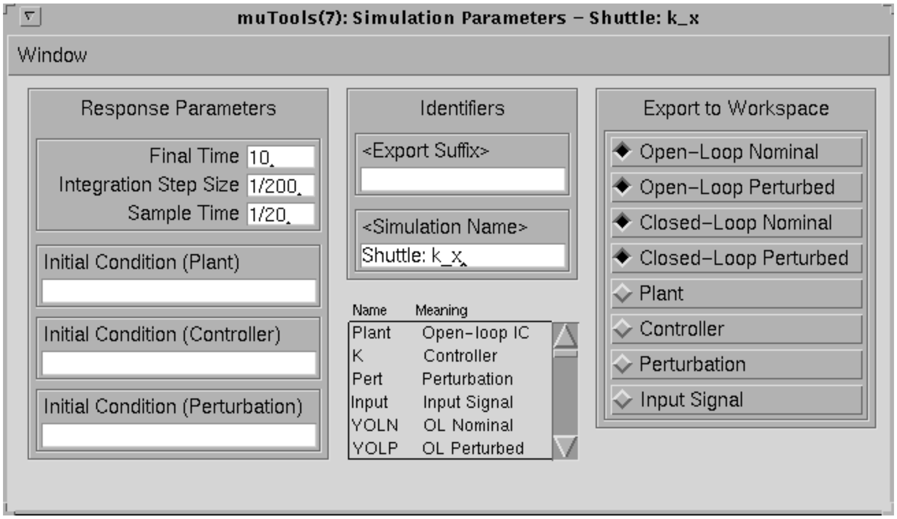
Figure 7-43: simgui Main Window for Shuttle Time Simulation

We are interested in the nominal and perturbed closed-loop response in the presence of a worst-case, real perturbation. This corresponds to the top nine channels of `olsim`, the aero-coefficient perturbations. Therefore, worst-case real perturbations of size 1 in the aero-coefficients for `k_x` and `k_mu` are calculated using the `wcperf` command. These perturbations are:

$$\begin{array}{lllllllllll} \texttt{badpertx} & = & \mathrm{diag}([\ 1 & 1 & 1 & -.87 & 1 & 1 & -.28 & -1 & -1 &]) \\ \texttt{badpertmu} & = & \mathrm{diag}([\ -1 & -1 & -1 & 1 & 1 & 1 & -1 & -1 & -1 &]) \end{array}$$

`badpertx` is used in the perturbed response for controller `k_x` and `badpertmu` is used as the worst-case real perturbation for controllers `k_h` and `k_mu`. This data is input into the `Perturbation` editable text in the `simgui` main window.

The controllers are implemented in discrete-time at a sample-rate of 20Hz on the shuttle. To replicate the same implementation, a sample-data time simulation is performed. This simulation is available under the **Options** menu in the Main simulation window. Therefore, the continuous-time controllers, `k_x`, `k_h`, and `k_mu`, must be discretized for the sampled-data time simulation. The continuous-time plant, `olsim`, is simulated at 200Hz and the controllers at 20Hz as seen in the `simgui` parameter window, Figure 7-44.



muTools(7): Simulation Parameters – Shuttle: k_x

Window

Response Parameters

Final Time 10

Integration Step Size 1/200

Sample Time 1/20

Initial Condition (Plant)

Initial Condition (Controller)

Initial Condition (Perturbation)

Identifiers

<Export Suffix>

<Simulation Name> Shuttle: k_x

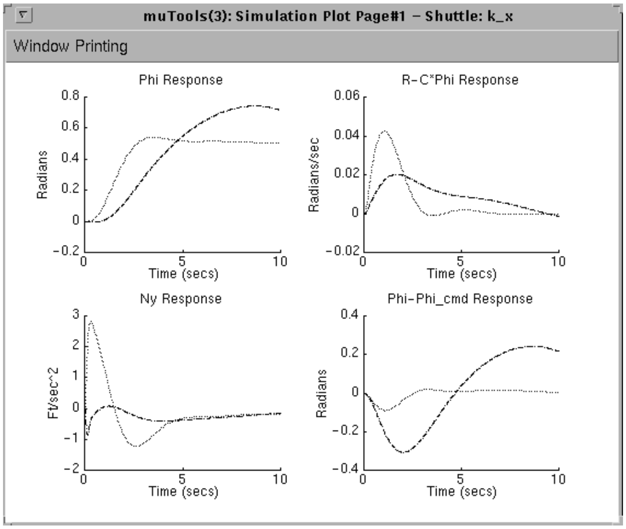
Name	Meaning
Plant	Open-loop IC
K	Controller
Pert	Perturbation
Input	Input Signal
YOLN	OL Nominal
YOLP	OL Perturbed

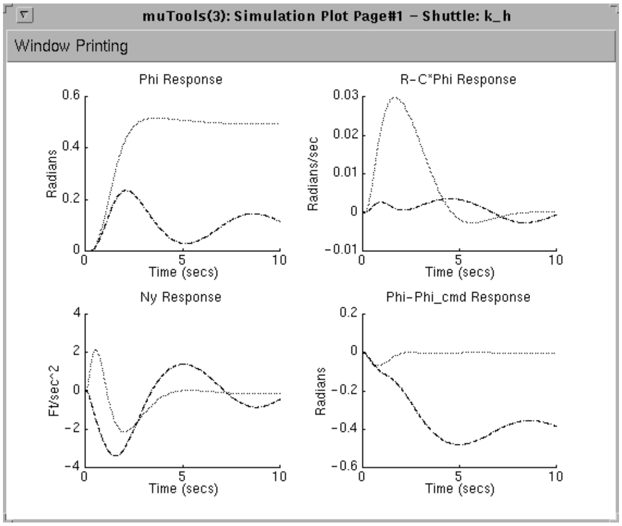
Export to Workspace

◆ Open–Loop Nominal
◆ Open–Loop Perturbed
◆ Closed–Loop Nominal
◆ Closed–Loop Perturbed
◇ Plant
◇ Controller
◇ Perturbation
◇ Input Signal

Figure 7-44: simgui Parameter Window for Shuttle Time Simulation

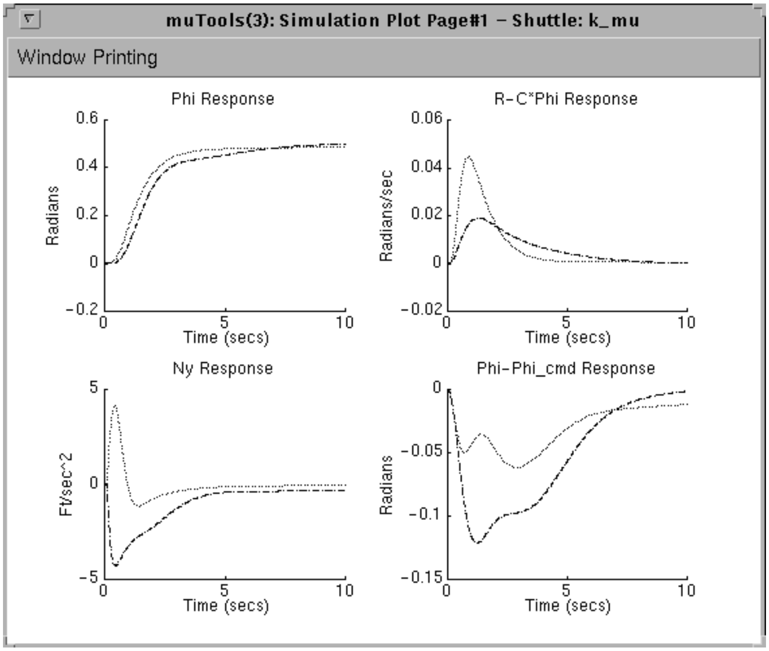
The nominal and perturbed closed-loop responses with `k_x`, `k_h`, and `k_mu` implemented are shown in Figure 7-45 and 7-46. As expected, the time domain simulations reinforce the conclusions that were reached in the frequency domain analysis. The nominal performance associated with `k_h` is superb, but degrades significantly with the aerodynamic uncertainty. In that respect, the controller `k_mu` performs the best, nearly achieving all of the robust performance objectives. The nominal and perturbed time response of other performance variables can also be easily investigated.





nominal (solid line)   perturbed (dashed line)

Figure 7-45: Closed-Loop Nominal and Perturbed Time Response, k_x **(top) and** k_h



Nominal (solid line)   Perturbed (dashed line)

Figure 7-46: Closed-Loop Nominal and Perturbed Time Response, `k_mu`

Conclusions

This exercise illustrated the use of the μ-Tools software to analyze the robust stability and robust performance objectives on a complicated, uncertain plant model.

There is an important feature of the `mu` software that cannot be overlooked or overemphasized. These algorithms calculate both upper *and* lower bounds for μ, and produce worst-case perturbations which provide the lower bound. The perturbations, and their effects, can be analyzed in both the frequency domain and time domain. In practice, the bad perturbations are also used in high fidelity, nonlinear simulations of the closed-loop system to discover limitations and unforeseen problems.

Although this problem did not have repeated uncertain parameters (each δ.. appeared only once), the algorithms and software do handle these cases, and the reader is referred back to the “Complex Structured Singular Value” section in Chapter 4 for details.

Space Shuttle References

Doyle, J., K. Lenz, and A. Packard, “Design Examples Using μ Synthesis: Space Shuttle Lateral Axis FCS During Reentry,” *NATO ASI Series, Modelling, Robustness, and Sensitivity Reduction in Control Systems*, vol. 34, Springer-Verlag, 1987.

Shuttle Rigid Body Model

The perturbed, state-space rigid body model of the aircraft, `acnom`, is shown in the following figure.

$$\text{acnom} = \left[\begin{array}{c|c} \text{A\_acnom} & \text{B\_acnom} \\ \hline \text{C\_acnom} & \text{D\_acnom} \end{array}\right]$$

$$\text{A\_acnom} = \begin{bmatrix} -9.5e-2 & 1.4e-1 & -9.9e-1 & 3.6e-2 \\ -3.6e+0 & -4.3e-1 & 2.8e-1 & 0 \\ 4.0e-1 & -1.3e-2 & -8.1e-2 & 0 \\ 0 & 1 & -1.4e-1 & 0 \end{bmatrix}$$

$$\text{B\_acnom} = \begin{bmatrix} 1.3e-2 & 0 & 0 & -1.2e-2 & 1.0e-2 & -1.1e-7 \\ 0 & -3.1e-2 & -3.1e+0 & 6.6e+0 & 1.3e+0 & -4.1e-6 \\ 0 & -1.9e-1 & -6.4e-2 & 3.8e-1 & -2.6e-1 & 4.5e-7 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\text{C\_acnom} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -6.8e+1 & -1.7e+0 & -4.1e+0 & -3.7e-5 \\ 0 & 0 & 0 & 1.0e+0 \end{bmatrix}$$

$$\text{D\_acnom} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 1.2e-6 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1.1e+1 & -1.1e+1 & -1.1e+1 & 2.7e+1 & -3.0e+0 & -7.8e-5 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

HIMAT Robust Performance Design Example

This section contains an idealized example of μ-synthesis as applied to the design of a pitch axis controller of an experimental highly maneuverable airplane, HIMAT. The airplane model is taken from aerodynamic data for the HIMAT vehicle. The problem is posed as a robust performance problem, with multiplicative plant uncertainty at the plant input and plant output weighted sensitivity function as the performance criterion. The design procedure presented in this section involves several steps:

1. Specification of closed-loop feedback structure.
2. Specification of model uncertainty and performance objectives in terms of frequency-dependent weighting matrices.
3. Construction of open-loop interconnection for control synthesis routines.
4. Loop shape controller design for the open-loop interconnection.
5. H_∞ optimal controller design for the open-loop interconnection.
6. Analysis of robust performance properties of the resulting closed-loop systems using the structured singular value, μ (μ-analysis).
7. Use of frequency dependent similarity scalings, obtained in the μ-analysis step, to scale the open-loop interconnection, and redesign H_∞ controller (iterating on steps 5, 6, and 7 constitutes the approach to μ-synthesis called “$D – K$ iteration,” which is described in detail in Chapter 5).

The main objective of this section is to illustrate μ-synthesis design methods (steps 1, 2, 3, 5, 6, 7). The loop shape controller (step 4) is included to illustrate that robust stability and nominal performance do not necessarily imply robust performance.

Many of the command outputs are not displayed in the text, since it is assumed that the reader is simultaneously working through the example on a computer.

HIMAT Vehicle Model and Control Objectives

The HIMAT vehicle model and control objectives are taken from a paper by Safonov *et al*. (1981). The interested reader should consult this paper as well as Hartman *et al*. (1979) and Merkel and Whitmoyer (1976) for more details. The HIMAT vehicle was a scaled, remotely piloted vehicle (RPV) version of an advanced fighter, which was flight tested in the late 1970s. The actual HIMAT vehicle is currently on display in the Smithsonian National Aerospace Museum in Washington, D.C. The design example will consider only the longitudinal dynamics of the airplane. These dynamics are assumed to be uncoupled from the lateral-directional dynamics. Linearized models for a collection of flight conditions can be found in [HartBG]. The state vector consists of the vehicle’s basic rigid body variables.

$$x^{\mathrm{T}} = (\delta v, \alpha, q, \theta)$$

representing the forward velocity, angle-of-attack, pitch rate, and pitch angle, respectively. The flight path angle (γ) is defined as $\gamma = \theta - \alpha$. The state variables used to describe motions in the vertical plane are given below.

δv — perturbations along the velocity vector

α — angle between velocity vector and aircraft's longitudinal axis

q — rate-of-change of aircraft attitude angle

θ — aircraft attitude angle

The control inputs are the elevon (δ_e) and the canard (δ_c). The variables to be measured are α and θ.

There are three longitudinal maneuvers to be considered.

Vertical Translation: Control the vertical velocity at a constant θ (α varies). This implies that the attitude is held constant as the velocity vector rotates.

Pitch Pointing: Control the attitude at a constant flight path angle (i.e., $\theta - \alpha$ = constant). In this case the velocity vector does not rotate.

Direct Lift: Control the flight path angle at constant angle-of-attack (i.e., $\gamma = \theta$). This maneuver produces a normal acceleration response without changing the angle-of-attack.

These control objectives are accounted for within the performance specification.

Closed-Loop Feedback Structure

A diagram for the closed-loop system, which includes the feedback structure of the plant and controller, and elements associated with the uncertainty models and performance objectives, is shown inFigure 7-47.

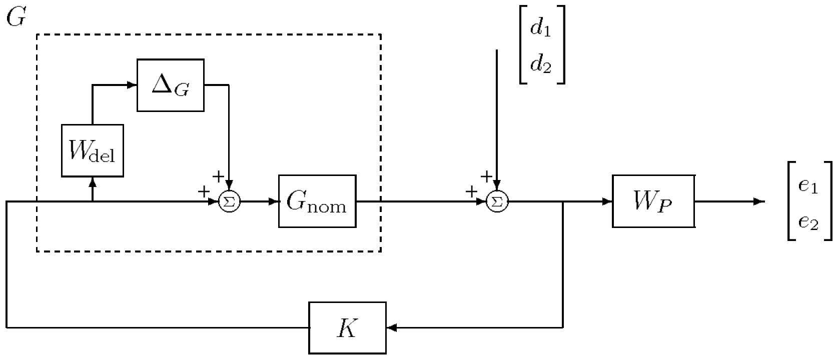

G
Δ_G
W_{del}
+
+
Σ
G_{nom}
$\begin{bmatrix} d_1 \\ d_2 \end{bmatrix}$
+
+
Σ
W_P
$\begin{bmatrix} e_1 \\ e_2 \end{bmatrix}$
K

Figure 7-47: HIMAT Closed-Loop Interconnection Structure

The dashed box represents the true airplane, with associated transfer function G. Inside the box is the nominal model of the airplane dynamics, G_{nom}, and two elements, w_{del} and Δ_G, which parametrize the uncertainty in the model. This type of uncertainty is called multiplicative uncertainty at the plant input, for obvious reasons. The transfer function w_{del} is assumed known, and reflects the amount of uncertainty in the model. The transfer function Δ_G is assumed to be stable and unknown, except for the norm condition, $\|\Delta_G\|_\infty < 1$. The performance objective is that the transfer function from d to e be small, in the $\|\cdot\|_\infty$ sense, for all possible uncertainty transfer functions Δ_G. The weighting function W_P is used to reflect the relative importance of various frequency ranges for which performance is desired.

The control design objective is to design a stabilizing controller K such that for all stable perturbations $\Delta_G(s)$, with $\|\Delta_G\|_\infty < 1$, the perturbed closed-loop system remains stable, and the perturbed weighted sensitivity transfer function,

$$S(\Delta_G) := W_P(I + P(I + \Delta_G W_{\text{del}})K)^{-1}$$

has $\|S(\Delta_G)\|_\infty < 1$ for all such perturbations. Recall that these mathematical objectives exactly fit in the structured singular value framework.

Uncertainty Models

The airplane model we consider has two inputs: elevon command (δ_e) and canard command (δ_c); and two measured outputs: angle-of-attack (α) and pitch angle (θ).

A first principles set of uncertainties about the aircraft model would include:

- Uncertainty in the canard and the elevon actuators. The electrical signals that command deflections in these surfaces must be converted to actual mechanical deflections by the electronics and hydraulics of the actuators. This is not done perfectly in the actual system, unlike the nominal model.
- Uncertainty in the forces and moments generated on the aircraft, due to specific deflections of the canard and elevon. As a first approximation, this arises from the uncertainties in the aerodynamic coefficients, which vary with flight conditions, as well as uncertainty in the exact geometry of the airplane. An even more detailed view is that surface deflections generate the forces and moments by changing the flow around the vehicle in very complex ways. Thus there are uncertainties in the force and moment generation that go beyond the quasi-steady uncertainties implied by uncertain aerodynamic coefficients.
- Uncertainty in the linear and angular accelerations produced by the aerodynamically generated forces and moments. This arises from the uncertainty in the various inertial parameters of the airplane, in addition to neglected dynamics such as fuel slosh and airframe flexibility.
- Other forms of uncertainty that are less well understood.

In this example, we choose not to model the uncertainty in this detailed manner, but rather to lump all of these effects together into one full-block uncertainty at the input of a four-state, nominal model of the aircraft rigid body. This nominal model has no (i.e., perfect) actuators and only quasi-steady dynamics. The nominal model for the airplane is loaded from the `mutools/subs` directory.

The simple model of the airplane has four states: forward speed (v), angle-of-attack (α), pitch rate (q) and pitch angle (θ); two inputs: elevon command (δ_e) and canard command (δ_c); and two measured outputs: angle-of-attack (α) and pitch angle (θ).

```
mkhimat;
minfo(himat)
seesys(himat,'%9.le')
```

```
 -2.3e-02  -3.7e+01  -1.9e+01  -3.2e+01 |   0.0e+00   0.0e+00
  0.0e+00  -1.9e+00   9.8e-01   0.0e+00 |  -4.1e-01   0.0e+00
  1.2e-02  -1.2e+01  -2.6e+00   0.0e+00 |  -7.8e+01   2.2e+01
  0.0e+00   0.0e+00   1.0e+00   0.0e+00 |   0.0e+00   0.0e+00
  ------------------------------------- |  -----------------
  0.0e+00   5.7e+01   0.0e+00   0.0e+00 |   0.0e+00   0.0e+00
  0.0e+00   0.0e+00   0.0e+00   5.7e+01 |   0.0e+00   0.0e+00
```

The partitioned matrix represents the `[A B; C D]` state space data. Given this nominal model `himat` (i.e., $G_{nom}(s)$) we also specify a stable, 2×2 transfer matrix $W_{del}(s)$, called the uncertainty weight. These two transfer matrices parametrize an entire set of plants, $\mathcal{G}$, which must be suitably controlled by the robust controller K.

$$\mathcal{G} := \{G_{nom}(I + \Delta_G W_{del}) : \Delta_G \text{ stable}, \|\Delta_G\|_\infty \le 1\}.$$

All of the uncertainty in modeling the airplane is captured in the normalized, unknown transfer function Δ_G. The unknown transfer function $\Delta_G(s)$ is used to parametrize the potential differences between the nominal model $G_{nom}(s)$, and the actual behavior of the real airplane, denoted by G. The dependence on frequency of the uncertainty weight indicates that the level of uncertainty in the airplane's behavior depends on frequency.

In this example, the uncertainty weight W_{del} is of the form $W_{del}(s) := w_{del}(s)I_2$, for a given scalar valued function $w_{del}(s)$. The fact that the uncertainty weight is diagonal, with equal diagonal entries, indicates that the modeled uncertainty is in some sense a round ball about the nominal model G_{nom}. The scalar weight associated with the multiplicative input uncertainty is constructed using the command nd2sys. The weight chosen for this problem is $w_{\text{del}} = \frac{50(s+100)}{s+10000}$.

```
wdel = nd2sys([1 100],[1 10000],50);
```

The set of plants that are represented by this uncertainty weight is

$$\mathcal{G} := \left\{ G_{nom}\left(I_2 + \frac{50(s+100)}{s+10000}\Delta_G(s)\right) : \Delta_G(s) \text{ stable, } \|\Delta_G\|_\infty \le 1 \right\}$$

The weighting function is used to normalize the size of the unknown perturbation Δ_G. At any frequency ω, $|\omega_{\text{del}}(j\omega)|$ can be interpreted as the percentage of uncertainty in the model at that frequency.

```
omega = logspace(0,5,50);
wdel_g = frsp(wdel,omega);
vplot('liv,lm',wdel_g)
title('Multiplicative Uncertainty Weighting Function')
xlabel('Frequency (rad/s)')
```

The particular uncertainty weight chosen for this problem indicates that at low frequency, there is potentially a 50% modeling error, and at a frequency of 173 rad/sec, the uncertainty in the model is up to 100%, and could get larger at higher and higher frequencies. A frequency response of w_{del} is shown in Figure 7-48.

Specifications of Closed-Loop Performance

The performance of the closed loop system will be evaluated using the output sensitivity transfer function, $(I + GK)^{-1}$. Good performance will be characterized in terms of a weighted H_∞ norm on this transfer function. Given a 2 × 2 stable, rational transfer matrix W_P, we say that nominal performance is achieved if $\|W_P(I + GK)^{-1}\|_\infty < 1$. As in the uncertainty modeling, the weighting function W_p is used to normalize specifications, in this case, to define performance as whether a particular norm is less than 1.

In this problem, we choose a simple weight of the form $W_P(s) = w_p(s)I_2$, where $w_p(s) = \frac{0.5(s+3)}{s+0.03}$.

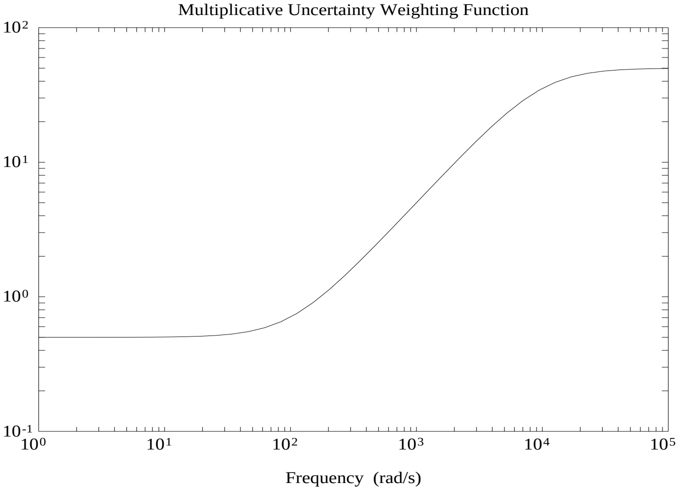


Figure 7-48: HIMAT Multiplicative Uncertainty Weighting Function

```
wp = nd2sys([1 3],[1 0.03],0.5);
```

For performance to be achieved, $\|W_P(I + GK)^{-1}\|_\infty < 1$, and since W_P is a scalar (times a 2×2 identity), the maximum singular value plot of the sensitivity transfer function $(I + GK)^{-1}$ must lie below the plot of $\frac{1}{|w_p|}$ at every frequency. That is, $\|W_P(I + GK)^{-1}\|_\infty < 1$, if and only if at all frequencies, $\bar{\sigma}[(I + GK)^{-1}(j\omega)] < |1/w_p(j\omega)|$.

```
omega = logspace(-3,2,50);
wp_g = frsp(wp,omega);
vplot('liv,lm',minv(wp_g))
title('Inverse of Performance Weighting function')
xlabel('Frequency (rad/s)')
```

This sensitivity weight indicates that at low frequency, the closed-loop (both nominal and perturbed) should reject disturbances at the output by a factor of 50-to–1. Expressed differently, steady-state tracking errors in both channels, due to reference step-inputs in either channel should be on the order of 0.02 or smaller. This performance requirement gets less and less stringent at higher and higher frequencies. The closed-loop system should perform better than open-loop for frequencies up to 1.73 radians/second, and for higher frequencies, the closed-loop performance should degrade gracefully, always lying underneath the inverse of the weight, wp. The frequency response of $\frac{1}{w_p}$ is shown in Figure 7-49.

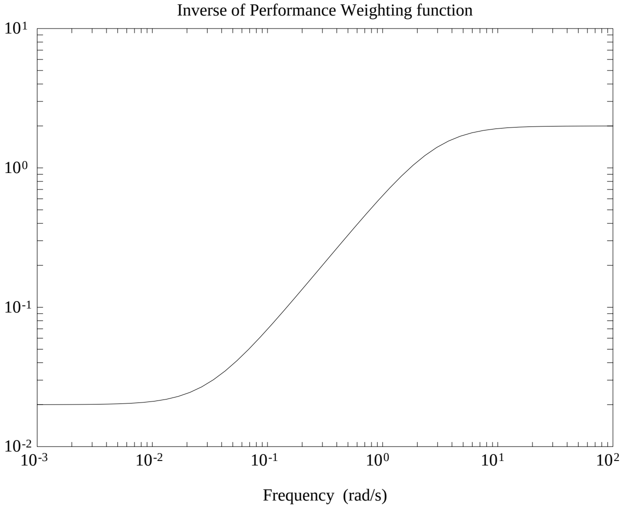


Figure 7-49: Inverse of the HIMAT Performance Weight

The 2 × 2 weighting matrices in the interconnection involve the scalar functions we have discussed, and identity matrices of dimension 2. We can build these matrices using the command daug, which stands for diagonal augmentation. Each new weight has two states, two inputs and two outputs as one can see using minfo.

```
wdel = daug(wdel,wdel);
wp = daug(wp,wp);
minfo(wdel)
minfo(wp)
```

The engineering motivation for a performance specification like this might come from the desire to have independent tracking of the angle of attack and pitch angle. This allows the vehicle to be pointed in pitch independently from vertical motions. We would expect this to be difficult to achieve, given that it is obviously easier for the vehicle to simultaneously pitch up and accelerate up than it is to simultaneously pitch down and accelerate up.

Robust Stability, Nominal Performance, Robust Performance

The phrases robust stability, nominal performance, and robust performance are used in this framework extensively.

Nominal Performance. The closed-loop system achieves nominal performance if the performance objective is satisfied for the nominal plant model, G_{nom}.

In this problem, that is equivalent to:

$$\text{Nominal Performance} \Leftrightarrow \|W_P(I + G_{\text{nom}}K)^{-1}\|_\infty < 1$$

Robust Stability. The closed-loop system achieves robust stability if the closed loop system is internally stable for all of the possible plant models $G \in \mathcal{G}$.

In this problem, that is equivalent to a simple norm test on a particular nominal closed-loop transfer function.

$$\text{Robust Stability} \Leftrightarrow \|W_{\text{del}}KG_{\text{nom}}(I + KG_{\text{nom}})^{-1}\|_\infty < 1$$

Robust Performance. The closed-loop system achieves robust performance if the closed-loop system is internally stable for all $G \in \mathcal{G}$, and in addition to that, the performance objective,

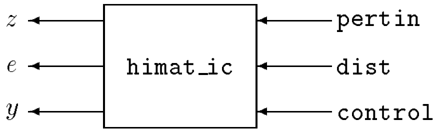
$$\|W_P(I + GK)^{-1}\|_\infty < 1,$$

is satisfied for every $G \in \mathcal{G}$. The property of robust performance is equivalent to a structured singular value test (a generalization of the two H_∞ norm tests in the previous conditions) on a particular, nominal closed- loop transfer function. This is discussed further in Chapter 4, “Modeling and Analysis of Uncertain Systems”.

Building the Open-Loop Interconnection

The command `sysic` is used to construct the open-loop interconnection. We will often refer to this open-loop system as the generalized plant. In this particular example, we store the system in the MATLAB variable `himat_ic`. The command `sysic` will build any specified interconnection of smaller subsystems, provided the correct information about the interconnection is in the MATLAB workspace.

A six-input, six-output SYSTEM matrix, himat_ic, (also referred to as *P*)



has internal structure shown in Figure 7-50. The variables control, pertin, dist, and y are two element vectors.

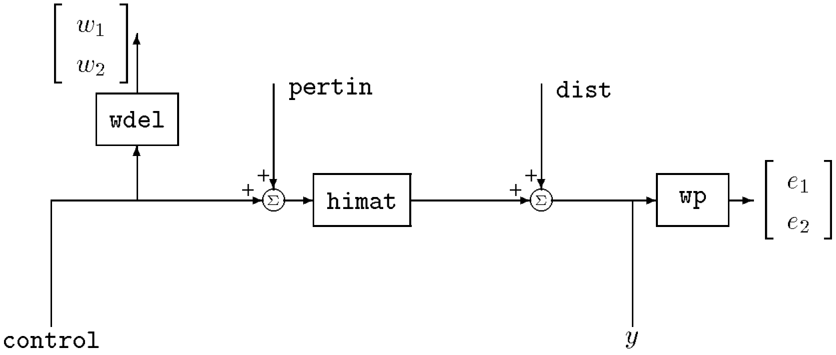


Figure 7-50: HIMAT Open-Loop Interconnection Structure

This can be produced with nine MATLAB commands, listed below. The first eight lines describe the various aspects of the interconnection, and may appear in any order. The last command, sysic, produces the final interconnection. The commands can be placed in an M-file, or executed at the command line.

```
systemnames = ' himat wp wdel ';
inputvar = '[ pertin{2} ; dist{2} ; control{2} ]';
outputvar = '[ wdel ; wp ; himat + dist ]';
input_to_himat = '[ control + pertin ]';
input_to_wdel = '[ control ]';
input_to_wp = '[ himat + dist ]';
sysoutname = 'himat_ic';
cleanupsysic = 'yes';
sysic;
```

Since the system himat_ic is still open-loop, its poles are simply the poles of the various components that make up the interconnection.

```
minfo(himat_ic)
spoles(himat_ic)
spoles(himat)
spoles(wdel)
spoles(wp)
```

μ-synthesis and D – K Iteration

For notational purposes, let $P(s)$ denote the transfer function of the six-input, six-output open-loop interconnection, himat_ic. Define a block structure Δ as

$$\Delta := \left\{ \begin{bmatrix} \Delta_1 & 0 \\ 0 & \Delta_2 \end{bmatrix} : \Delta_1 \in \mathbf{C}^{2 \times 2}, \Delta_2 \in \mathbf{C}^{2 \times 2} \right\} \subset \mathbf{C}^{4 \times 4} .$$

The first block of this structured set corresponds to the full-block uncertainty Δ_G used in section to model the uncertainty in the airplane's behavior. The second block, Δ_2 is a fictitious uncertainty block, used to incorporate the H_∞ performance objectives on the weighted output sensitivity transfer function into the μ-framework.

Using theorem 4.5 from the "Robust Performance" section in Chapter 4, a stabilizing controller K achieves closed-loop, robust performance if and only if for each frequency $\omega \in [0, \infty]$, the structured singular value

$$\mu_\Delta[F_L(P,K)(j\omega)] < 1$$

Using the upper bound for μ, (recall that in this case, two full blocks, the upper bound is exactly equal to μ) we can attempt to minimize the peak closed-loop μ value by posing the optimization problem

$$\min_{\substack{K \\ \text{stabilizing}}} \ \min_{\substack{d(s) \\ \text{stable,min-phase}}} \left\| \begin{bmatrix} d(s)I_2 & 0 \\ 0 & I_2 \end{bmatrix} F_L(P,K) \begin{bmatrix} d^{-1}(s)I_2 & 0 \\ 0 & I_2 \end{bmatrix} \right\|_\infty$$

Finding the value of this minimum and constructing controllers K that achieve levels of performance arbitrarily close to the optimal level is called μ-synthesis. A more detailed discussion of $D – K$ iteration is given in Chapter 5.

Before plunging into the $D – K$ iteration design procedure, we begin with a controller designed via basic MIMO loop shaping methods.

Loop Shaping Control Design

One approach to control design for the HIMAT model is to synthesize a loop shaping controller. We want the loop shape controller, K_{loop}, to make the open-loop gain act as an integrator at low frequency and at crossover. At high frequencies, we won't worry too much about the details of the roll-off, provided that it is at least first order. To achieve this, we'll roughly invert the plant $G(s)$ (G has only 1 MIMO finite zero, at $s \approx –0.026$; it also has zeros at $s = \infty$, so our inverse is only approximate) and augment the desired loop gain dynamics to the controller. The series of commands below constructs one such controller and plots the open-loop gain (broken at the input to the controller), as seen in Figure 7-51. The interested reader may want to explore various alternative schemes for constructing loop shape controllers discussed in Freudenberg and Looze (1988).

```
[a,b,c,d] = unpck(himat);
cn = c*a*a + 1000*c*a;
dn = c*a*b + 1000*c*b;
kloop = mscl(minv(pck(a,b,cn,dn)),-9000);
L = mmult(himat,kloop);
omega = logspace(-1,4,50);
Lg = frsp(L,omega);
vplot('liv,lm',vsvd(Lg))
title('Loop Gain Plot with Loop Shape Controller')
xlabel('Frequency (rad/s)')
```

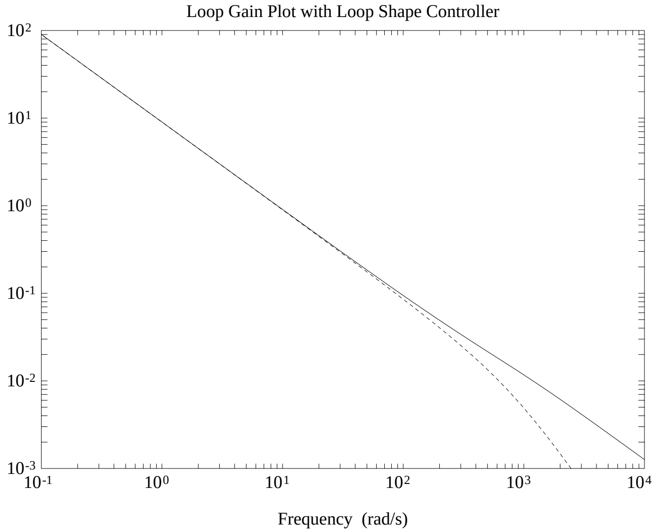


Figure 7-51: Loop Gain of the Loop Shaping Controller

The open-loop gain plot satisfies both the low frequency performance objective and the high frequency robustness goals. We have only plotted the singular values of GK_{loop}, but $K_{loop}G$ looks similar. Hence, you would expect the controller to satisfy the robust stability and nominal performance requirements.

The two 2 × 2 transfer functions associated with robust stability and nominal performance can be evaluated for the loop shaping controller. Simply close the open-loop interconnection P (`himat_ic`) with the loop shaping controller, K_{loop} (`kloop`) and evaluate the pertinent transfer functions using the command `sel`.

In using `sel`, the desired outputs (or rows) are specified first, followed by the desired inputs (or columns). The results are seen in Figure 7-52.

```
clp = starp(himat_ic,kloop,2,2);
spoles(clp)
rs_loop = sel(clp,1:2,1:2);
np_loop = sel(clp,3:4,3:4);
rs_loopg = frsp(rs_loop,omega);
np_loopg = frsp(np_loop,omega);
vplot('liv,m',vnorm(rs_loopg),vnorm(np_loopg))
tmp1 = 'ROBUST STABILITY (solid) and';
tmp2 = ' NOMINAL PERFORMANCE (dashed)';
title([tmp1 tmp2])
xlabel('Frequency(rad/s)')
```

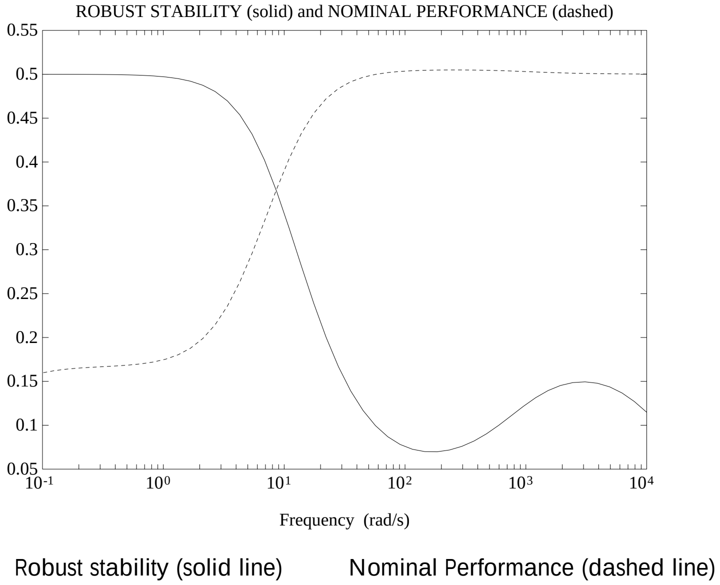


Figure 7-52: Robust Stability and Nominal Performance Plots for the Loop Shaping Controller

The interpretation of the plots in Figure 7-52 is as follows:

- The controlled system (with loop shaping controller) achieves nominal performance. This conclusion follows from the singular value plot of the nominal weighted output sensitivity function, which has a peak value of 0.50.
- The controlled system (with loop shaping controller) achieves robust stability. This conclusion stems from the singular value plot of the nominal weighted input complementary sensitivity function, which has a peak value of 0.50.

H_∞ Design on the Open-Loop Interconnection

In this section, we carry out the first step of the $D - K$ iteration, which is an H_∞ (sub)optimal control, design for the open-loop interconnection, `himat_ic`. In terms of the iteration, this amounts to holding the d variable fixed (at 1), and minimizing the $\|\cdot\|_\infty$ norm of $F_L(P,K)$, over the controller variable K. Recall that $F_L(P,K)$ is the nominal closed loop transfer function from the perturbation inputs and disturbances (sysic variables `pertin` and `dist`) to the perturbation outputs and errors (z and e), shown in Figure 7-53.

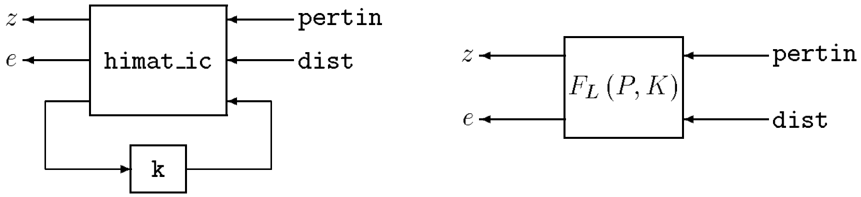


Figure 7-53: Closed-Loop Linear Fractional Transformation

The function `hinfsyn` designs a (sub)optimal H_∞ control law based on the open-loop interconnection structure provided. Syntax, input and output arguments of `hinfsyn` are

```
[k,clp] = hinfsyn(p,nmeas,ncon,glow,ghigh,tol)
```

The arguments are as follows.

Inputs

open-loop interconnection (SYSTEM matrix)	p
number of measurements	nmeas
number of controls	ncons
lower bound for bisection	glow
upper bound for bisection	ghigh
absolute tolerance for bisection method	tol

Outputs

controller (SYSTEM matrix)	k
closed-loop (SYSTEM matrix)	clp

In this example, the open-loop interconnection is himat_ic, with two measurements, two control inputs, and the bisection algorithm will search for the optimal achievable closed-loop norm, to an absolute tolerance of 0.06, between lower and upper limits of 0.8 and 6.0, respectively. Since we are planning on performing several iterations of the $D - K$ iteration procedure, we label the resulting controller k1. The resulting closed loop system (4-input, 4-output), from [pertin;dist] to [z;e] is labeled clp1.

```
[k1,clp1] = hinfsyn(himat_ic,2,2,0.8,6.0,.06);
```

The controller is stable, and its Bode plot is shown in Figure 7-54.

Properties of Controller

```
minfo(k1)
omega = logspace(-1,4,50);
spoles(k1)
k1_g = frsp(k1,omega);
vplot('bode',k1_g)
subplot(211), title('Frequency Response of k1')
```

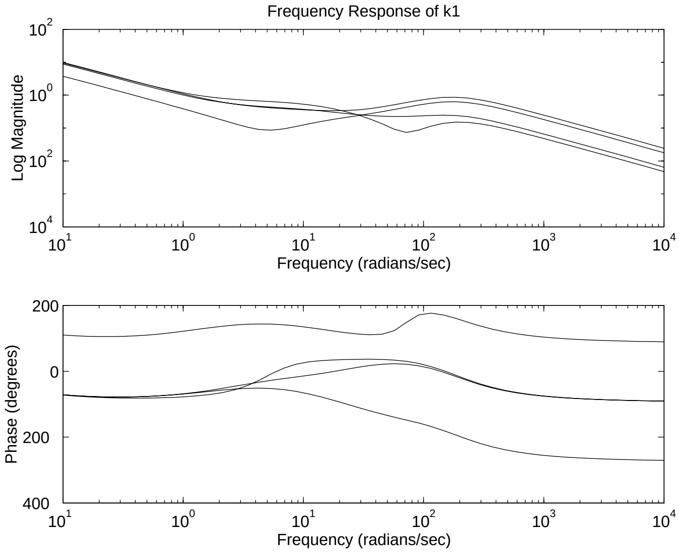


Figure 7-54: Bode Plot of k1

Figure 7-55 shows the singular values of the closed-loop system `clp1`. Although `clp1` is 4×4, at each frequency it only has rank equal to 2, hence only two singular values are nonzero.

Closed-Loop Properties

```
rifd(spoles(clp1))
clp1g = frsp(clp1,omega);
clp1gs = vsvd(clp1g);
vplot('liv,m',clp1gs)
title('Singular Value Plot of clp1')
xlabel('Frequency (rad/s)')
```

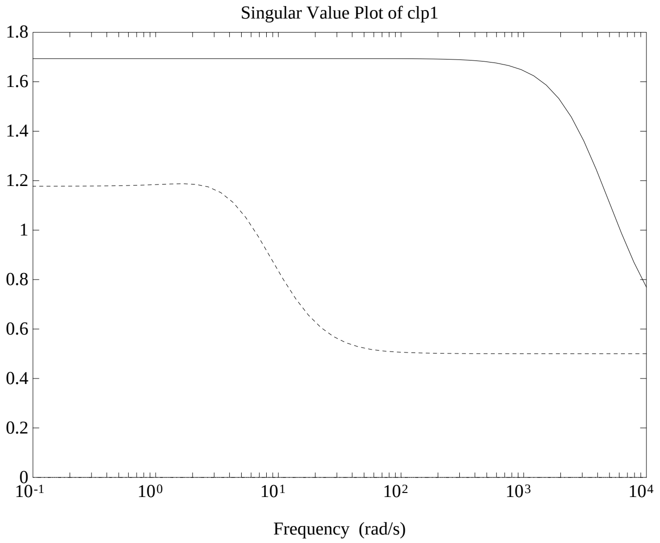


Figure 7-55: Singular Value Plot of the Closed-Loop System with k1

The two 2 × 2 transfer functions associated with robust stability and nominal performance may be evaluated separately, using the command `sel`. Recall that the robust stability test is performed on the upper 2 × 2 transfer function in `clp1`, and the nominal performance test is on the lower 2 × 2 transfer function in `clp1`. Since a frequency response of `clp1` is already available, (in `clp1g`) we simply perform the `sel` on the frequency response, and plot the norms.

```
rob_stab = sel(clp1g,[1 2],[1 2]);
nom_perf = sel(clp1g,[3 4],[3 4]);
minfo(rob_stab)
minfo(nom_perf)
vplot('liv,m',vnorm(rob_stab),vnorm(nom_perf))
tmp1 = 'ROBUST STABILITY (solid) and';
tmp2 = ' NOMINAL PERFORMANCE (dashed)';
title([tmp1 tmp2])
xlabel('Frequency (rad/s)')
```

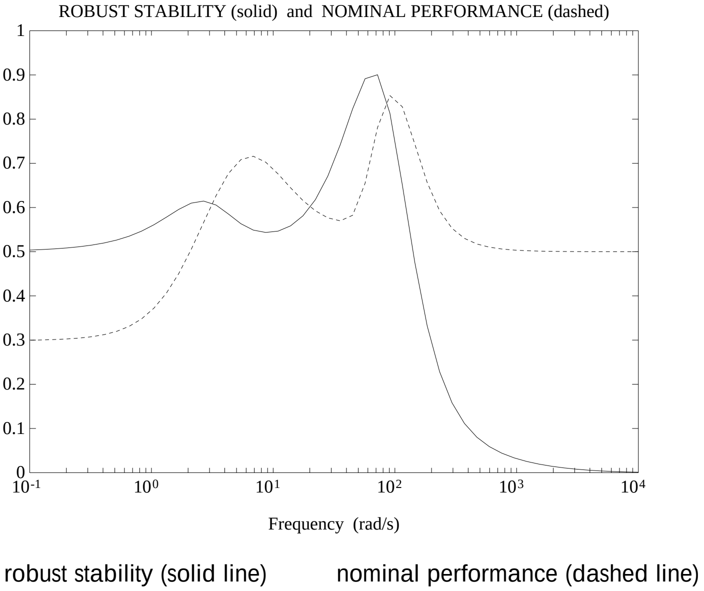


Figure 7-56: Robust Stability and Nominal Performance Plots Using Controller k1

The interpretation of the plots in Figure 7-56 is as follows:

- The controlled system achieves nominal performance. This conclusion follows from the singular value plot of the nominal weighted output sensitivity function, which has a peak value of 0.92.
- The controlled system achieves robust stability. This conclusion stems from the singular value plot of the nominal weighted input complementary sensitivity function, which has a peak value of 0.86.

Assessing Robust Performance with μ

The robust performance, HIMAT example properties of the two different closed-loop systems can be analyzed using μ-analysis. The closed-loop systems, `clp1` associated with the H_∞ controller, and clp, associated with the loop shaping controller, each have four inputs and four outputs. The first two inputs/outputs correspond to the two channels across which the perturbation Δ_G connects, while the third and fourth inputs/outputs correspond to the weighted output sensitivity transfer function. Therefore, for a frequency domain μ-analysis of robust performance properties, the block structure should consist of a 2 × 2 uncertainty block, and a 2 × 2 performance block.

$$\Delta := \left\{ \begin{bmatrix} \Delta_1 & 0 \\ 0 & \Delta_2 \end{bmatrix} : \Delta_1 \in \mathbf{C}^{2\times 2}, \Delta_2 \in \mathbf{C}^{2\times 2} \right\}$$

Referring back to the "Robust Performance" section in Chapter 4, robust performance (with respect to the uncertainty and performance weighting functions specified above) is achieved if and only if for every frequency, $\mu_\Delta(\cdot)$ of the closed-loop frequency response is less than 1.

The syntax of a general μ calculation is:

```
[bnds,dvec,sens,pvec] = mu(matin,deltaset)
```

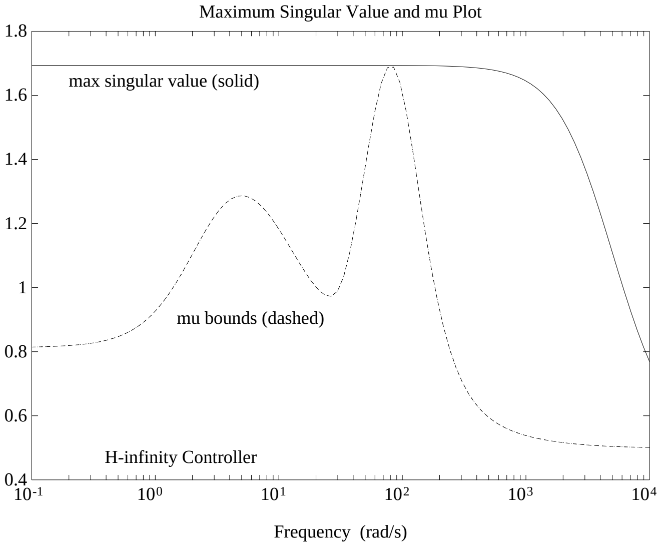
The μ-analysis program, `mu`, calculates upper and lower bounds for the structured singular value of the matrix `matin`, with respect to the block structure `deltaset`. The matrix `matin` can be a CONSTANT MATLAB matrix, or a VARYING matrix, such as a frequency response matrix of a closed-loop transfer function. In this example, the frequency response is `clp1g` and the block structure is two, 2 × 2 full blocks. mu returns the upper and lower bounds in 1× 2 VARYING matrix `bnds1`, the frequency-varying *D*-scaling matrices in `dvec1`, the frequency dependent perturbation associated with the lower bound in `rp1`, and the sensitivity of the upper bound to the *D*-scales in sens1.

The bounds can be calculated by specifying the block structure, and running `mu`.

μ Analysis of H_∞ Design

The H_∞ design is analyzed with respect to structured uncertainty using μ. First, the density of points in the frequency response is increased from 50 to 100 to yield smoother plots. Then the upper and lower bounds for μ are calculated on the 4 × 4 closed-loop response of the matrix `clp_g1`. The upperand lower bounds for μ are plotted (in this example they lie on top of one another) along with the maximum singular value in Figure 7-57.

```
deltaset=[2 2; 2 2];
omega1 = logspace(-1,4,100);
clp_g1 = frsp(clp1,omega1);
[bnds1,dvec1,sens1,pvec1] = mu(clp_g1,deltaset);
vplot('liv,m',vnorm(clp_g1),bnds1)
title('Maximum Singular Value and mu Plot')
xlabel('Frequency (rad/s)')
text(.15,.84,'max singular value (solid)','sc')
text(.3,.4,'mu bounds (dashed)','sc')
text(.2,.15,'H-infinity Controller','sc')
```



$F_L(P,K_1)(j\omega)$ (solid line) robust performance μ (dashed line)

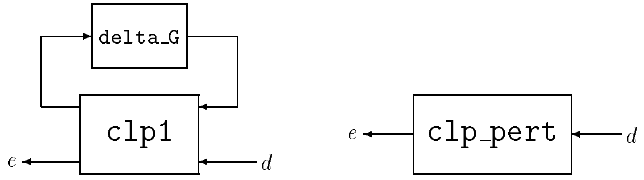
Figure 7-57: Maximum Singular Value of the 4 × 4 Closed-Loop Transfer Function $F_L(P,K_1)(j\omega)$ and Robust Performance μ

Hence, the controlled system (from H_∞) does not achieve robust performance. This conclusion follows from the μ plot in Figure 7-57, which peaks to a value of 1.69, at a frequency of 73.6 rad/sec. This means that there is a perturbation matrix Δ_G, with $\|\Delta_G\|_\infty = \frac{1}{1.69}$, for which the perturbed weighted sensitivity gets large

$$\|W_P(I + G_{\text{nom}}(I + W_{\text{del}}\Delta_G)K^{-1}\|_\infty = 1.69$$

This perturbation, Δ_G, can be constructed using dypert. The input variables to the command dypert consist of two outputs from μ, the perturbation matrix and the bounds, along with the block structure, and the numbers of the blocks for which the rational matrix construction should be carried out. Often times, some of the blocks correspond to performance blocks and therefore need not be constructed. Here, only the first block is an actual perturbation, so the construction is only done for this 2 × 2 perturbation (fourth argument of dypert).

```
delta_G = dypert(pvec1,deltaset,bnds1,1);
minfo(delta_G) % 2 by 2
rifd(spoles(delta_G)) % stable
hinfnorm(delta_G) % 1/1.69
clp_pert = starp(delta_G,clp1,2,2); % close top loop with delta
minfo(clp_pert)
rifd(spoles(clp_pert)) % stable, since RS passed
hinfnorm(clp_pert) % degradation of performance
```



μ-Analysis of Loop Shape Design

Robust performance for the system with the loop shape controller, K_{loop}, can also be analyzed using μ. You might think that the loop shaping controller would exhibit good robust performance properties, based on its excellent nominal performance and robust stability properties.

```
clpg = frsp(clp,omega1);
bnds_loop = mu(clpg,deltaset);
vplot('liv,m',bnds_loop)
title('mu Plot of Closed-loop System')
xlabel('Frequency (rad/s)')
text(.6,.85,'Loop Shape Controller','sc')
```

However, the closed-loop system with the loop shaping controller does not achieve robust performance. In fact, μ reaches a peak value of 11.7 at a frequency of 0.202 rad/sec, as seen in Figure 7-58. This means that there is a perturbation matrix Δ_G, with $\|\Delta_G\|_\infty = \frac{1}{11.7}$, for which the perturbed weighted sensitivity gets large

$$\|W_P(I + G_{\text{nom}}(I + W_{\text{del}}\Delta_G)K^{-1}\|_\infty = 11.7$$

Notice that this perturbation is 8.2 times smaller than the perturbation associated with the H_∞ control design, but that the subsequent degradation in closed-loop performance is 8.2 times worse. Therefore, the loop shaping controller will most likely perform poorly on the real system.

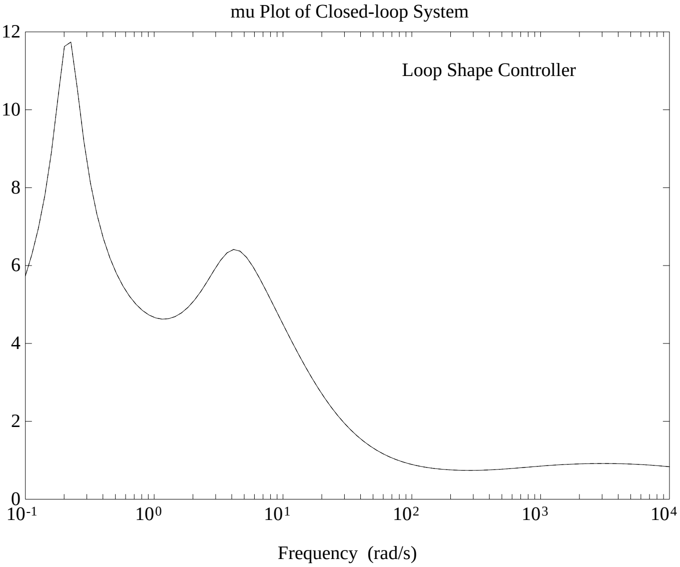


Figure 7-58: Robust Performance μ Plot of the Closed-Loop HIMAT System with the Loop Shaping Controller

The structured singular value μ is large in the low frequency range due to the off-diagonal elements of `clpg` being large. One can see this using the command `blknorm`, which outputs the individual norms of the respective blocks. The coupling between the off-diagonal terms associated with 0.202 rads/sec point to the problem — the upper right entry is 0.14, somewhat small, but not small enough to counteract the large (nearly 1000) lower left entry. As expected, $\mu \approx \sqrt{0.14 * 959} = 11.6$.

```
blkn_cl = blknorm(clpg,deltaset);
see(xtract(blkn_cl,.15,.3))

2 rows 2 columns

iv = 0.159986
   4.9995e-01   1.4127e-01
   5.6525e+02   1.6402e-01

iv = 0.202359
   4.9991e-01   1.4193e-01
   9.5950e+02   1.6520e-01

iv = 0.255955
   4.9985e-01   1.4294e-01
   7.5635e+02   1.6607e-01
```

Recapping Results

Let's summarize what has been done so far:

- The generalized plant, himat_ic, which includes the aircraft model, uncertainty and performance weighting functions, and the interconnection of all of these components was built using sysic.
- A controller was designed using hinfsyn.
- The robust performance characteristics of the closed-loop system were analyzed with a structured singular value frequency domain test using mu.

The structured singular value analysis involved computing μ at each frequency of this 4×4 closed loop response, with respect to a block structure Δ which is made up of two 2×2 full blocks. The blocks represent, respectively, uncertainty in the aircraft model, and the performance objectives.

At this stage, the controller which has been designed using H_∞ techniques (nearly) minimizes the H_∞ norm of the closed loop transfer function from the 4×1 vector of perturbation inputs and disturbance inputs to the 4×1 vector of perturbation outputs and error signals. The structured singular value analysis shows that the μ analysis improves on the $\bar{\sigma}(\cdot)$ bound at most frequencies, but there is no improvement at the frequency of 73.6 rads/sec.

Hence, the peak value of the μ-plot is as high as the peak value on the singular value plot, the μ analysis seems to have been of little use. However, at most of the frequencies, μ is smaller than $\bar{\sigma}$, and in the next iteration of synthesis, the controller can essentially focus its efforts at the problem frequency, and lower the peak of the μ-plot.

D – K Iteration for HIMAT Using dkit

The μ-Tools M-file `dkit` automates the μ-synthesis procedure via *D* – *K* iteration. This example is a modified version of the HIMAT problem considered earlier (see Figure 7-47) and is extended to include a frequency dependent sensor noise signal, as shown in the closed-loop interconnection diagram in Figure 7-59. This sensor noise signal is included to represent a more realistic performance objective.

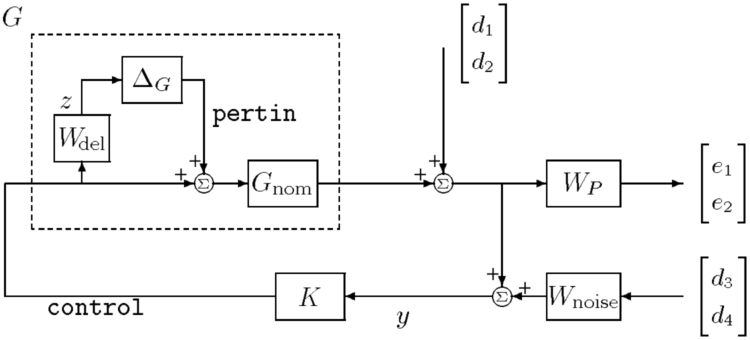
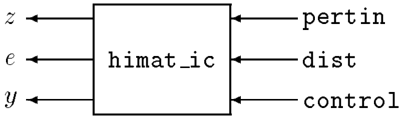

G
Δ_G
z
pertin
W_{del}
G_{nom}
$\begin{bmatrix} d_1 \\ d_2 \end{bmatrix}$
W_P
$\begin{bmatrix} e_1 \\ e_2 \end{bmatrix}$
K
W_{noise}
$\begin{bmatrix} d_3 \\ d_4 \end{bmatrix}$
control
y
+
+
+
+
+
+

Figure 7-59: HIMAT Closed-Loop Interconnection Structure

Now, the open-loop interconnection structure is the eight input, six output linear system, shown below



with internal structure, as in Figure 7-60.

The M-file mkhicn creates the plant model, weighting functions and the interconnection structure shown in Figure 7-60. This can be produced with nine MATLAB commands, listed below, and also in the M-file mkhicn (which creates the plant and weighting functions).

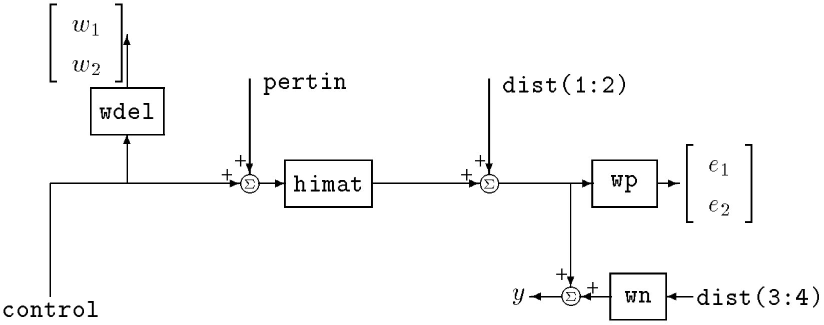


Figure 7-60: HIMAT Open-Loop Interconnection Structure

```
mkhicn
```

file: mkhicn.m

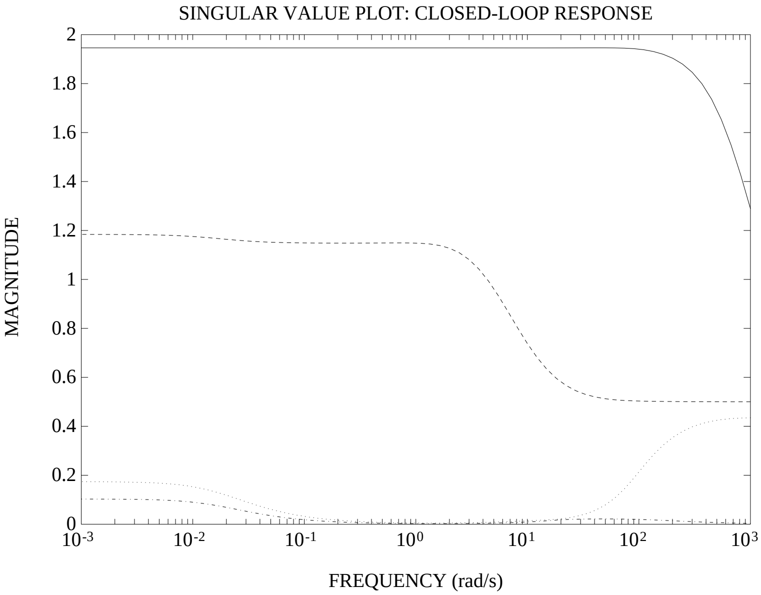
```
mkhimat;
wdel = nd2sys([50 5000],[1 10000]);
wp = nd2sys([0.5 0.9],[1 0.018]);
poleloc = 320;
Wn = nd2sys([2 0.008*poleloc],[1 poleloc]);
wdel = daug(wdel,wdel);
wp = daug(wp,wp);
Wn = daug(wn,wn);
   systemnames = ' himat wp wdel wn ';
   inputvar = '[ pertin{2} ; dist{4} ; control{2} ]';
   outputvar = '[ wdel ; wp ; himat + dist(1:2) + wn ]';
   input_to_himat = '[ control + pertin ]';
   input_to_wdel = '[ control ]';
   input_to_wp = '[ himat + dist(1:2) ]';
   input_to_wn = '[ dist(3:4) ]';
   sysoutname = 'himat_ic';
   cleanupsysic = 'yes';
sysic;
```

The dkit file himat_dk has been set up with the necessary variables to design robust controllers for HIMAT using $D - K$ iteration. A listing of the himat_dk file follows. You can copy this file into your directory from the μ-Tools subroutines directory, mutools/subs, and modify it for other problems, as appropriate.

```
% himat_dk
%
% This script file contains the USER DEFINED VARIABLES for the
%mutools DKIT script file. The user MUST define the 5
%variables below.
%------------------------------------------%
   REQUIRED USER DEFINED VARIABLES
%------------------------------------------%
% Nominal plant interconnection structure
NOMINAL_DK = himat_ic;

% Number of measurements
NMEAS_DK = 2;

% Number of control inputs
NCONT_DK = 2;

% Block structure for mu calculation
BLK_DK = [2 2;4 2];

% Frequency response range
OMEGA_DK = logspace(-3,3,60);

%----------------------end of himat_dk-------------------------%
```

After the `himat_dk.m` file has been set up, you need to let the `dkit` program know which setup file to use. This is done by setting the string variable `DK_DEF_NAME` in the MATLAB workspace equal to the setup filename. Typing `dkit` at the MATLAB prompt will then begin the $D - K$ iteration procedure.

```
  DK_DEF_NAME = 'himat_dk';
  dkit
  starting mu iteration #: 1

  Iteration Number: 1
  -------------------

  Information about the Interconnection Structure IC_DK:
  system10 states 6 outputs8 inputs
  Test bounds: 0.0000 < gamma <= 100.0000

   gamma  hamx_eig  xinf_eig  hamy_eig  yinf_eig   nrho_xy   p/f

 100.000   2.3e-02   0.0e+00   1.8e-02   0.0e+00    0.0003    p

  50.000   2.3e-02   0.0e+00   1.8e-02   0.0e+00    0.0011    p

  25.000   2.3e-02   0.0e+00   1.8e-02   0.0e+00    0.0046    p

  12.500   2.3e-02   0.0e+00   1.8e-02   0.0e+00    0.0183    p

   6.250   2.3e-02   0.0e+00   1.8e-02   0.0e+00    0.0742    p

   3.125   2.3e-02   0.0e+00   1.8e-02   0.0e+00    0.3117    p

   1.562   2.3e-02   0.0e+00   1.7e-02   0.0e+00    1.5583#   f

   2.152   2.3e-02   0.0e+00   1.8e-02   0.0e+00    0.7100    p

Gamma value achieved: 2.1516
```

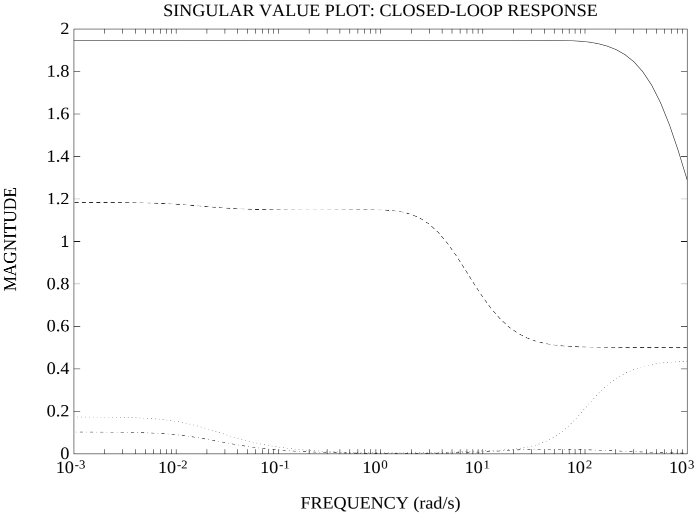


Singular value plot of closed-loop system in graphics window. Make sure that the chosen frequency range is appropriate.

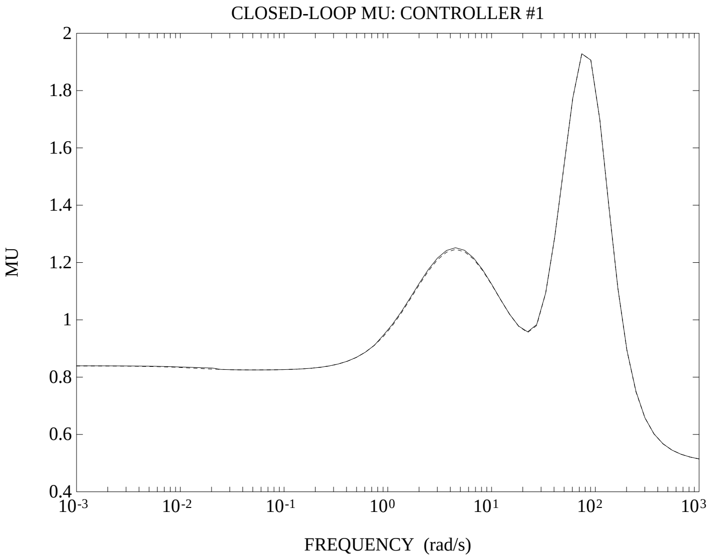
Next, we get to change the frequency range, if desired. For illustrative purposes, we will change the number of logarithmically spaced points from 60 to 70.

```
Do you want to modify OMEGA_DK? (y/n): y

Current Frequency Variable
---------------------------------------------------------------
(s)Frequency Spacinglog
(n) # Frequency Points60
(b)Frequency - bottom1.00e-03
(h)Frequency - high1.00e+03

Enter (s n b and/or h) to change OMEGA, (e) to exit unchanged: n

------------CHANGING # of Points--------------

Enter desired # of points: 70

Current Frequency Variable
---------------------------------------------------------------
(s)Frequency Spacinglog
(n) # Frequency Points70
(b)Frequency - bottom1.00e-03
(h)Frequency - high1000

Enter (s n b and/or h) to change, (e) to exit: e
```

By typing e, we exit the frequency range modification, and the closed-loop singular value frequency response is recalculated and plotted. In this case, the plot looks exactly the same.



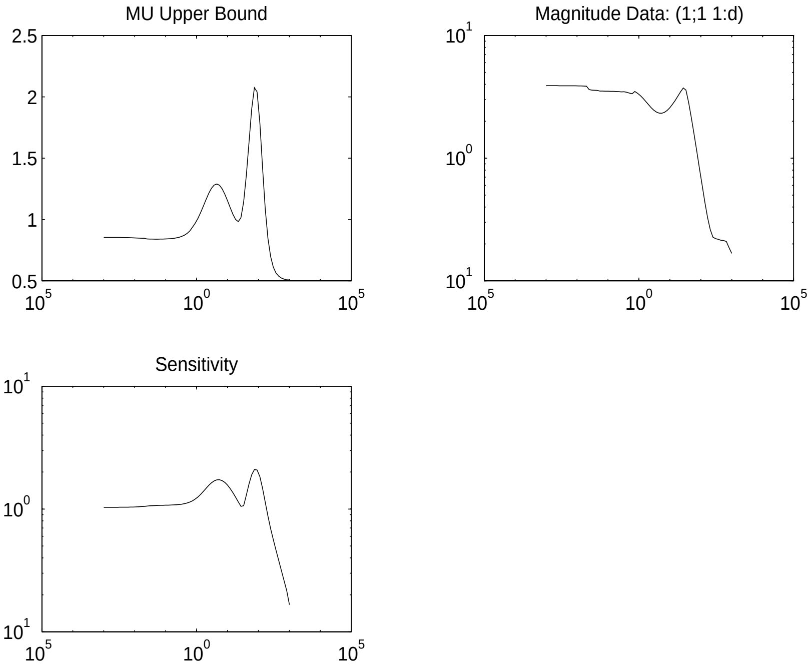
```
Singular value plot of closed-loop system in graphics window.
Make sure that chosen Frequency range is appropriate.

   Do you want to modify OMEGA_DK? (y/n): n

   RERUN H_inf with DIFFERENT bounds/tolerances? (y/n): n

Calculating MU of closed-loop system: g_dk
points completed....
1.2.3.4.5.6.7.8.9.10.11.12.13.14.15.16.17.
18.19.20.21.22.23.24.25.26.27.28.29.30.31.32.33.34.35.
36.37.38.39.40.41.42.43.44.45.46.47.48.49.50.51.52.53 .
54.55.56.57.58.59.60.61.62.63.64.65.66.67.68.69.70.
```



```
   MU plot for control design:Press any key to continue

Iteration Summary
------------------------------------
Iteration #1
Controller Order10
Total D-Scale Order0
Gamma Achieved2.152
Peak mu-Value2.075

   Another D-K itera;tion? (y/n): y
```

Proceeding with the $D - K$ iteration, we must fit the D-scaling variable that was calculated in the μ upper-bound computation. This rational D-scaling will then be absorbed into the open-loop interconnection.

A plot of the μ upper bound, the first frequency-dependent *D*-scaling data (this is the curve we want to fit), and the sensitivity of the μ upper bound. The sensitivity measure roughly shows (across frequency) the relative importance of the accuracy of the curve fit. It is used in the curve fit optimization to weight some frequency ranges differently than others.



You are prompted to enter your choice of options for fitting the *D*-scaling data. Press return to see your options.

```
Enter Choice (return for list):
Choices:

nd       Move to Next D-Scaling

nb       Move to Next D-Block

i        Increment Fit Order

d        Decrement Fit Order

apf      Auto-PreFit

mx 3     Change Max-Order to 3

at 1.01  Change Auto-PreFit Tol to 1.01

0        Fit with zeroth order

2        Fit with second order

n        Fit with n'th order

e        Exit with Current Fittings

s        See Status
```

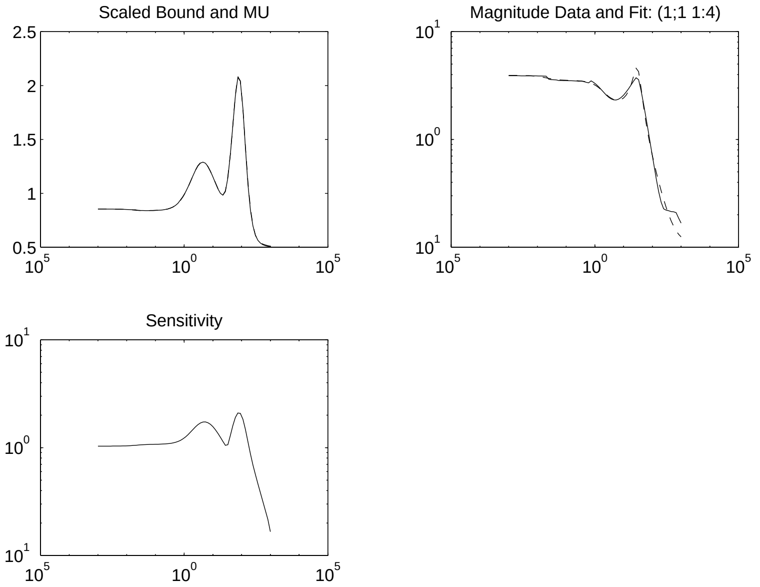
- `nd` and `nb` allow you to move from one *D*-scale data to another. `nd` moves to the next scaling, whereas `nb` moves to the next scaling block. For scalar *D*-scalings, these are identical operations, but for problems with full *D*-scalings, (perturbations of the form δI) they are different. In the (1,2) subplot window, the title displays the *D*-scaling Block number, the row/column of the scaling that is currently being fit, and the order of the current fit (with `d` for data, when no fit exists).
- The order of the current fit can be incremented or decremented (by 1) using `i` and `d`.
- `apf` automatically fits each *D*-scaling data. The default maximum state order of individual *D*-scaling is 5. The `mx` variable allows you to change the maximum *D*-scaling state order used in the automatic pre-fitting routine. `mx` must be a positive, nonzero integer. `at` allows you to define how close the rational, scaled μ upper bound is to approximate the actual μ upper bound in a norm sense. Setting `at 1` would require an exact fit of the *D*-scale data, and is not allowed. Allowable values are greater than 1, and the meaning is

explained in Chapter 5, “Control Design via m Synthesis”. This setting plays a role (mildly unpredictable, unfortunately) in determining where in the (*D*,*K*) space the *D* – *K* iteration converges.

- Entering a positive integer at the prompt will fit the current *D*-scale data with that state order rational transfer function.
- `e` exits the *D*-scale fitting to continue the *D* – *K* iteration.
- The variable `s` will display a status of the current and fits.

Select `apf` to automatically fit the *D*-scale data. After a few seconds of calculation, the first *D*-scale is fit with a fourth order rational curve as shown in the top-right plot, along with the frequency-dependent magnitude data that is being fit. Also shown in the top-left portion of the graphics window is a plot comparing the upper bound of μ (using the frequency dependent *D*-scalings) along with the maximum singular value of the closed-loop transfer function after being scaled by the just-computed rational fit. Note that the second *D*-scale data, which corresponds to the performance block, is fit with a constant. This is expected since one of the *D*-scalings can always be normalized to be 1. Enter `s` after the *D*-scale fitting is completed to see the status.

```
Enter Choice (return for list): apf
Starting Auto-PreFit...
Block 1 , Order = 0 1 2 3 4
Block 2 , Order = 0

Done

Enter Choice (return for list): s

Block 1: 4
Block 2: 0
Auto PreFit Fit Tolerance: 1.03
Auto PreFit Maximum Order: 5

Enter Choice (return for list):
```

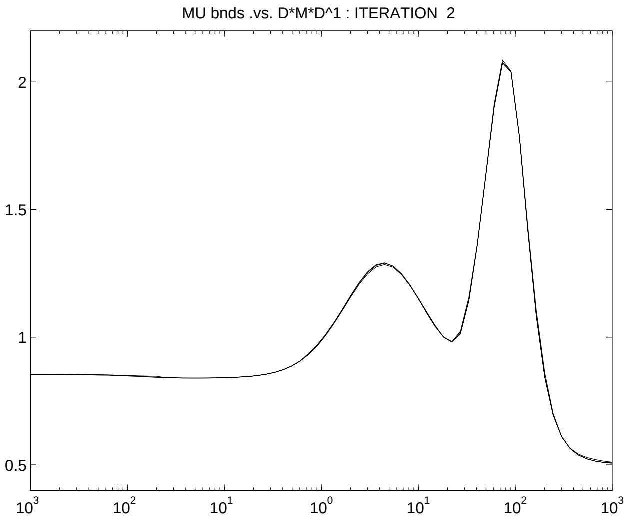


In this case, the μ upper bound with the *D*-scale data is very close to the μ upper bound with the rational *D*-scale fit. The fourth order fit is quite adequate in scaling the closed-loop transfer function. The curve fitting procedure for this scaling variable is concluded by entering `e` at the `Enter Choice` prompt.

```
Enter Choice (return for list): e
```

In this problem, the block structure consists of two complex full blocks: the 2 × 2 block associated with the multiplicative uncertainty model for the aircraft, and the 4 × 2 performance block. Since there are two blocks, there is only one *D*-scaling variable, and we are completely done with the curve fitting in this iteration.

At the conclusion of the curve fitting procedure, a frequency response plot is shown, which compares the norm of the rationally scaled, closed-loop system to the lower and upper bound for μ.

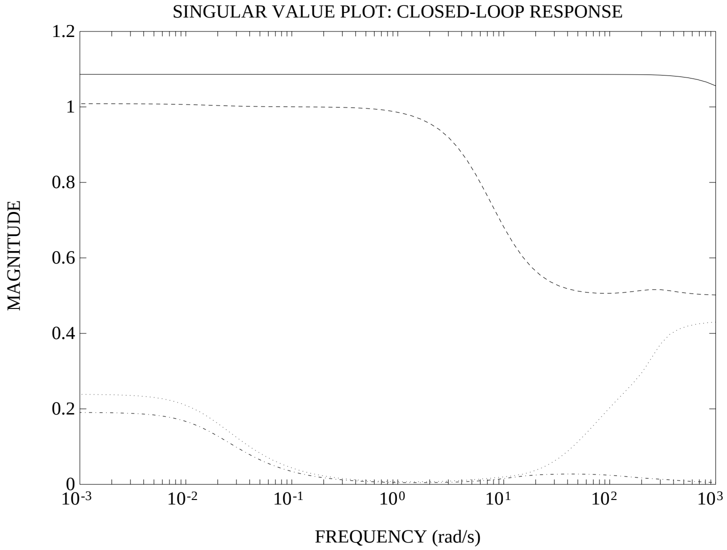


Finally, before the next H_∞ synthesis procedure, we get the option of changing the parameters used in the `hinfsyn` routine. This is useful to change the lower bound in the γ-iteration. In this example, we make no changes, and simply continue.

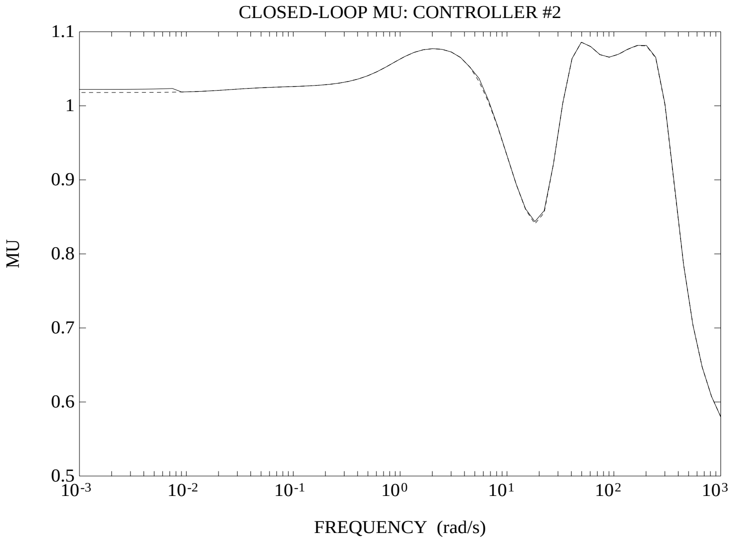
```
Altering the HINFSYN settings for next synthesis...

HINFSYN Settings Previously  Next
------------------------------------------------------------
(u)GAMMA Upper Bound100 2.146
(l)GAMMA Lower Bound  0.00e+000.00e+00
(t)Bisection Tolerance1.0004.29e-02
(p)Riccati PSD epsilonl.00e-06l.00e-06
(j)Riccati j-w epsilonl.00e-08l.00e-08

Enter (u l t p j) to change, (e) to exit: e
```

The iteration proceeds by computing the H_∞ optimal controller for the scaled (using the rational scalings from the curve fitting) open-loop system.

```
Iteration Number:  2
--------------------

Information about the Interconnection Structure IC_DK:
system:26 states6 outputs8 inputs
Test bounds: 0.0000 <  gamma  <= 2.1461


gamma  hamx_eig  xinf_eig  hamy_eig  yinf_eig nrho_xy   p/f

2.146   2.0e–02  –5.2e–14   1.8e–02  –1.5e–16  0.1557 p

1.073   1.9e–02  –4.6e–14   1.7e–02  –3.4e–17  0.9958    p

0.537   1.4e–14#   *******   1.2e–02  –1.7e–18  ******    f

0.805   1.8e–02  –7.7e–13   1.6e–02  –5.8e–18 4.9336#    f

1.019   1.9e–02  –6.0e–14   1.7e–02  –2.4e–17 1.2077#    f

1.055   1.9e–02  –2.3e–13   1.7e–02  –3.8e–17 1.0589#    f

Gamma value achieved:1.0730
```

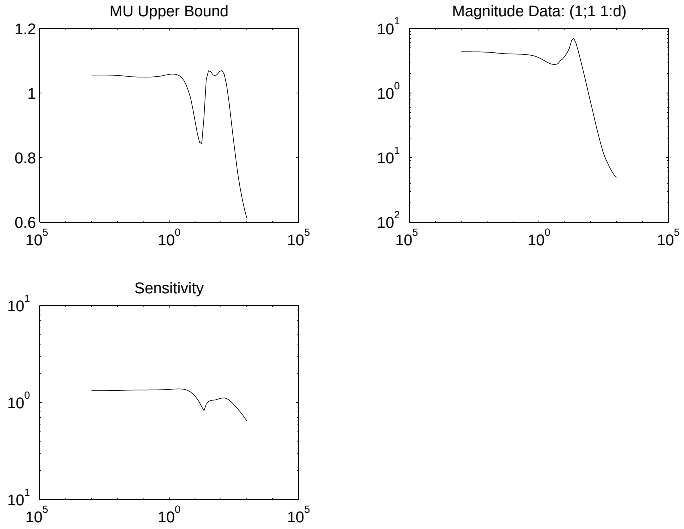


```
Singular value plot of closed-loop system in graphics window.
Make sure that chosen Frequency range is appropriate.

Do you want to modify OMEGA_DK? (y/n): n

RERUN H_inf with DIFFERENT bounds/tolerances? (y/n): n

Calculating MU of closed-loop system: g_dk
points completed....
1.2.3.4.5.6.7.8.9.10.11.12.13.14.15.16.17.
18.19.20.21.22.23.24.25.26.27.28.29.30.31.32.33.34.35.
36.37.38.39.40.41.42.43.44.45.46.47.48.49.50.51.52.53.
54.55.56.57.58.59.60.61.62.63.64.65.66.67.68.69.70.
```



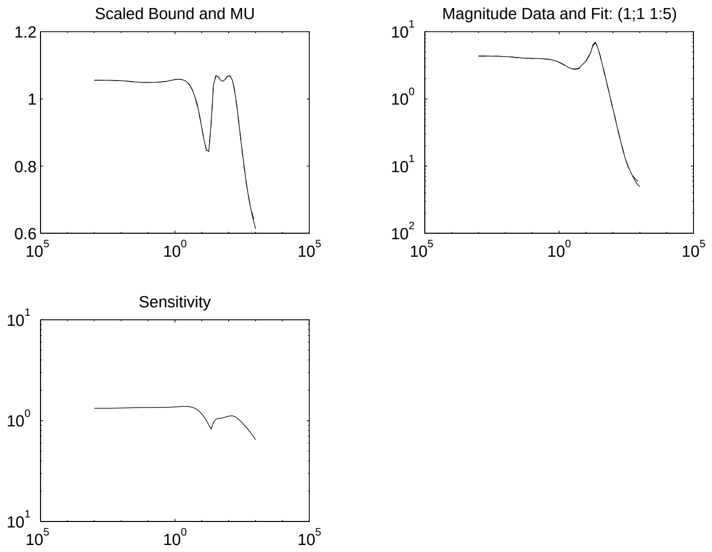
```
MU plot for control design:Press any key to continue

Iteration Summary
--------------------------------------------
Iteration #12
Controller Order1026
Total D-Scale Order016
Gamma Achieved2.1521.073
Peak mu-Value2.0751.073

Another D-K iteration? (y/n): y
```

The third iteration begins by fitting the new frequency-dependent *D*-scaling.



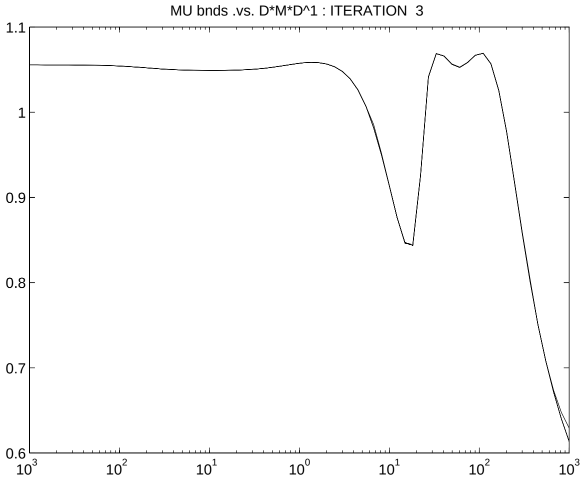
Again, enter the automatic pre-fitting option apf.

```
Enter Choice (return for list): apf
Starting Auto-PreFit...
Block 1 , Order = 0 1 2 3 4 5
Block 2 , Order = 0
Done
```

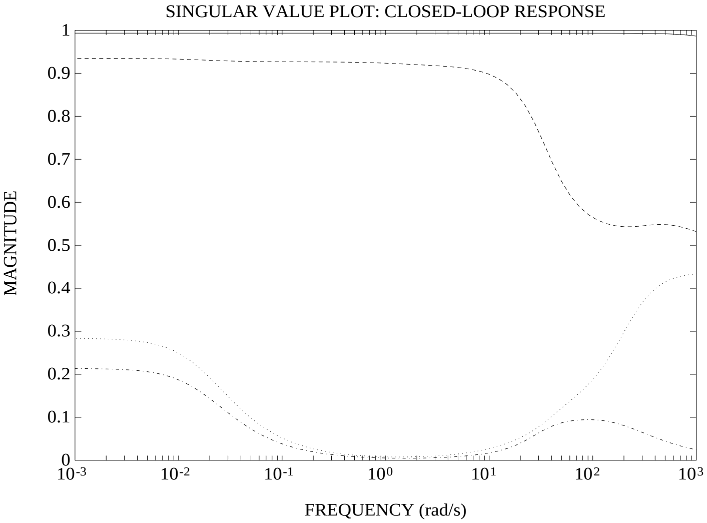


This fifth order fit works well in scaling the transfer function, so we exit the curve fitting routine.

```
Enter Choice (return for list):e
```

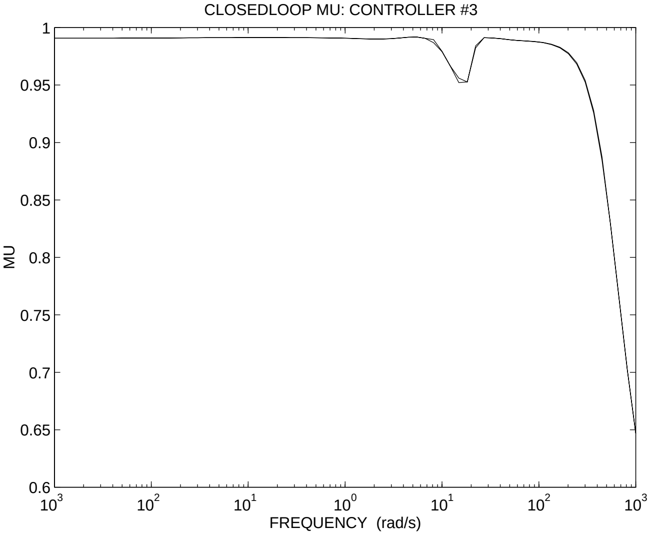


```
Altering the HINFSYN settings for next synthesis...

HINFSYN Settings       Previously          Next
----------------------------------------------------------------
(u)GAMMA Upper Bound   2.146               1.095
(1)GAMMA Lower Bound   0.00e+00            0.00e+00
(t)Bisection Tolerance 4.29e-02            2.19e-02
(p)Riccati PSD epsilon 1.00e-06            1.00e-06
(j)Riccati j-w epsilon 1.00e-08            1.00e-08

Enter (u l t p j) to change,  (e) to exit: e


Iteration Number:  3
--------------------

Information about the Interconnection Structure IC_DK:
system:30 states6 outputs8 inputs
Test bounds: 0.0000 <  gamma  <= 1.0947


gamma hamx_eig xinf_eig hamy_eig yinf_eig  nrho_xy   p/f

1.095  2.1e-02 -1.2e-13  1.7e-02 -3.1e-18   0.5970    p

0.547  9.1e-13#  *******  1.2e-02 -4.2e-17  ******     f

0.821  2.0e-02 -1.2e-11  1.6e-02 -5.5e-16  45.1126#    f

1.040  2.1e-02 -7.0e-13  1.7e-02 -2.4e-16   0.7263    p

0.996  2.1e-02 -8.4e-13  1.6e-02 -2.9e-17   0.8741    p

0.961  2.1e-02 -1.8e-13  1.6e-02 -2.5e-17   1.0433#    f

0.970  2.1e-02 -7.9e-13  1.6e-02 -2.2e-16   0.9922    p

Gamma value achieved:0.9704
```



Singular Value plot of closed-loop system in GRAPHICS window. Make sure that chosen Frequency range is appropriate

```
Do you want to modify OMEGA_DK? (y/n): n

RERUN H_inf with DIFFERENT bounds/tolerances? (y/n): n

Calculating MU of closed-loop system: g_dk
points completed....
1.2.3.4.5.6.7.8.9.10.11.12.13.14.15.16.17.
18.19.20.21.22.23.24.25.26.27.28.29.30.31.32.33.34.35.
36.37.38.39.40.41.42.43.44.45.46.47.48.49.50.51.52.53.
54.55.56.57.58.59.60.61.62.63.64.65.66.67.68.69.70.
```



```
MU plot for control design:Press any key to continue

Iteration Summary
-----------------------------------------------------------
Iteration #123
Controller Order102630
Total D-Scale Order01620
Gamma Achieved2.1521.0730.970
Peak mu-Value2.0751.0730.973

Another D-K itera;tion? (y/n): n
echo off
Next MU iteration number: 4
```

At this point, we have achieved the robust performance objective, and we end the *D* – *K* iteration. We have designed a 30 state controller using *D* – *K* iteration which achieves a μ value less than 1.

In this example, it is also possible to reduce the controller order to 12, using truncated balanced realizations, and still maintain closed-loop stability and robust performance.

```
max(real(spoles(kd_k3)))

ans =
   -1.6401e-02

[k_dk3bal,hsv] = sysbal(k_dk3);
[k_dk3red] = strunc(k_dk3bal,12);
clpred_12 = starp(himat_ic,k_dk3red);
max(real(spoles(clpred_12)))

ans =
   -6.9102e-03

clpred_12g = frsp(clpred_12,OMEGA_DK);
[bnds] = mu(clpred_12g,[2 2;4 2],'c');
pkvnorm(sel(bnds,1,1))

ans =
   9.9910e-01
```

H_∞ Loop Shaping Design for HIMAT

Now consider H_∞ loop shaping control design for the HIMAT example discussed in previous sections. Recall that the objective is to reject disturbances up to about 1 rad/sec in the presence of substantial plant uncertainty above 100 rad/sec. A loop-shaping design that gives a bandwidth of approximately 10 rad/sec and robustness which should be satisfactory.

Design Precompensator

First form the HIMAT system and plot its maximum singular values across frequency (see Figure 7-61).

```
mkhimat
[type,p,m,n] = minfo(himat);
om = logspace(-2,4,100);
himatg = frsp(himat,om);
vplot('liv,lm',vsvd(himatg),1);
title('SINGULAR VALUES OF HIMAT')
ylabel('SINGULAR VALUES'); xlabel('FREQUENCY (RAD/SEC)');
```

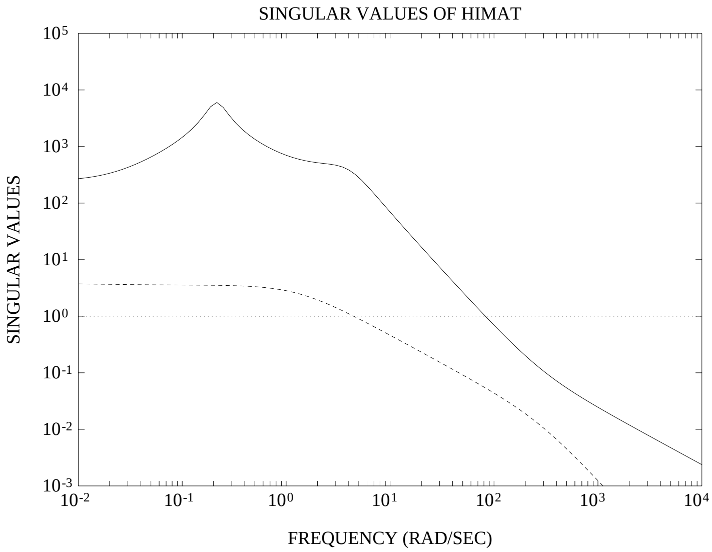


Figure 7-61: Singular Values of HIMAT

The singular values of `himat` are plotted in Figure 7-61, and although the unity gain cross over frequency is approximately correct, the low frequency gain is too low. We therefore introduce a proportional plus integral (P+I) precompensator with transfer function $(1 + s^{-1})I_{2\times2}$ to boost the low frequency gain and give zero steady state errors. The singular values of `himat` and `himat` augmented with the P+I compensator are shown in Figure 7-62.

```
sysW1 = daug(nd2sys([1 1],[1 0]),nd2sys([1 1],[1 0]));
sysGW = mmult(himat,sysW1);
sysGWg = frsp(sysGW,om);
vplot('liv,lm',vsvd(himatg),'-.',vsvd(sysGWg),'-',1,'--')
title('SINGULAR VALUES OF HIMAT AND AUGMENTED HIMAT')
ylabel('SINGULAR VALUES');
xlabel('FREQUENCY (RAD/SEC)');
```

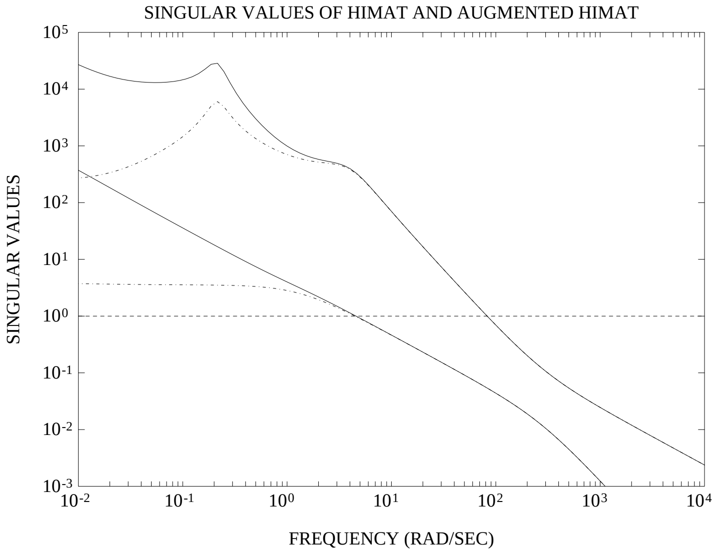


Figure 7-62: Singular Values of HIMAT (dashed-dotted) and Augmented Plant with the H_∞ Loop Shaping Controller (solid)

H_∞ Loop Shaping Feedback Compensator Design

The optimally robust controller can now be designed for the frequency shaped plant.

```
[sysK1,emax] = ncfsyn(sysGW,1);
disp(['emax = ' num2str(emax)]);
emax = 0.436
```

The value of `emax = 0.436` is a very satisfactory stability margin. The closed-loop norm can be checked by forming the open-loop interconnection of Figure 7-63, denoted by `p_ic`, and checking the reciprocal of the H_∞ gain. See the “Loop Shaping Using H• Synthesis” section in Chapter 3 for more details about this control design technique.

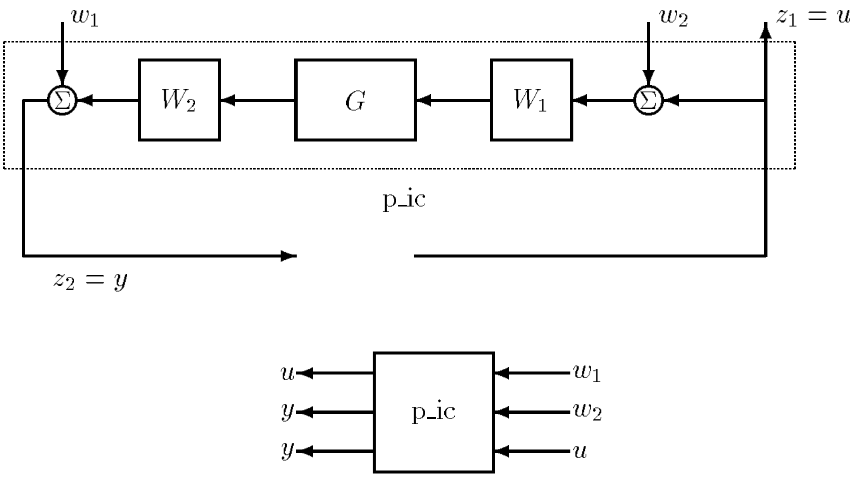


Figure 7-63: H_∞ Loop Shaping Standard Block Diagram

```
systemnames = 'sysGw';
inputvar = '[ w12; w22; u2 ]';
outputvar = '[ u; w1+sysGw; w1+sysGw ]';
input_to_sysGw = '[ w2+u ]';
sysoutname = 'p_ic';
cleanupsysic = 'yes';
sysic;
ncf_cl = starp(p_ic,sysK1);
ncf_cl_nm = hinfnorm(ncf_cl);
1/ncf_cl_nm(1)
ans =
   4.3598e-01
```

The implemented controller involves the pre- and postweighting functions W_1 and W_2, as shown in Figure 7-64.

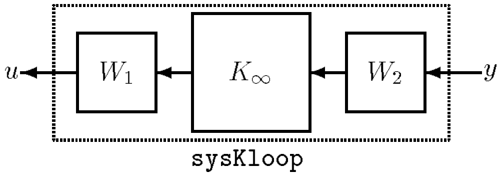


Figure 7-64: Actual Implemented H_∞ Loop Shaping Controller

In this example $W_2 = I_{2\times2}$, therefore, the implemented loop shaping controller is:

```
sysKloop = mmult(sysW1,sysK1);
```

Assessing Robust Performance with μ

We can now assess this controller by testing the original specification by using a μ-test as in previous designs. First the interconnection structure needs to be formed.

```
wdel = nd2sys([50,5000],[1,10000]);
wp = nd2sys([0.5,1.5],[1,0.03]);
wdel = daug(wdel,wdel);
wp = daug(wp,wp);
himatic
clear wp wdel
```

Now form the closed-loop and evaluate the robust performance μ with the H_∞ loop shaping compensator implemented (see Figure 7-65).

```
clp1 = starp(himat_ic,sysKloop,2,2);
clp_g1 = frsp(clp1,om);
deltaset = [2 2; 2 2];
[bnds1,dvec1,sens1,pvec1] = mu(clp_g1,deltaset);
vplot('liv,m',bnds1);
title('ROBUST PERFORMANCE MU WITH LOOPSHAPE CONTROLLER')
ylabel('MU');
xlabel('FREQUENCY (RAD/SEC)');
disp(['mu value is ' num2str(pkvnorm(sel(bnds1,1,1)))])
mu value is 1.323
```

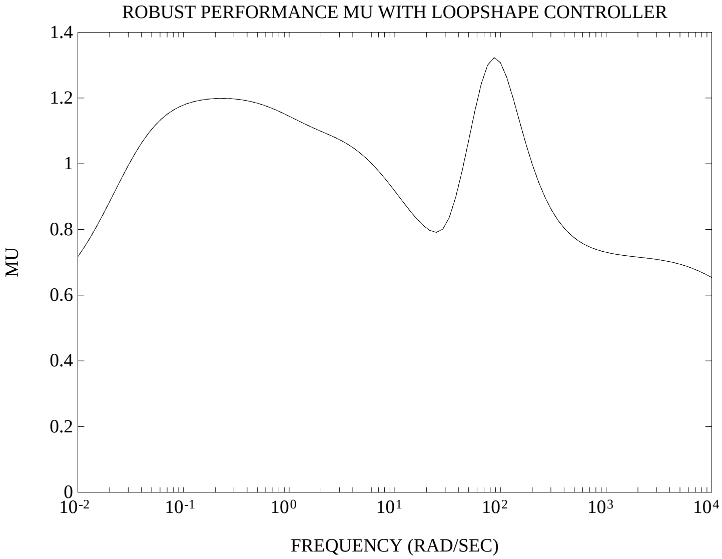


Figure 7-65: Robust Performance μ with sysK1

The plot of μ is shown in Figure 7-65 (solid line), the μ-value is close to that required, giving a satisfactory design without exploiting the details of the performance and uncertainty weights. This substantiates the claim that this design method can give a very robust initial design which does not require detailed trade-offs between weights to be studied.

Reduced Order Designs

The previously designed controller will typically have one less state than the precompensator plus the plant. It is therefore often desirable to reduce the number of states in the controller. There are systematic techniques for doing this based on model reduction in the ν gap metric, δ_ν, which is roughly equivalent to model reduction of normalized coprime factors of the plant and controller.

First reduce the weighted plant model order and measure the resulting gap.

```
[sysGW_cf,sigGW]=sncfbal(sysGW);
sigGW
sigGW =

   8.9996e-01
   7.1355e-01
   3.3542e-01
   7.9274e-02
   8.5314e-04
   2.1532e-04

sysGW_4 = cf2sys(hankmr(sysGW_cf,sigGW,4,'d'));
gapGW_4 = nugap(sysGW,sysGW_4)
gapGW_4 =

   8.6871e-04
```

It is seen that a fourth order model is essentially indistinguishable from the full order model due to the small value of the ν gap. Now design the controller for this reduced order system.

```
[sysK1_3,emax_3] = ncfsyn(sysGW_4,1);
emax_3
emax_3 =

   4.3597e-01
```

This three state controller can be reduced to two states using Hankel model reduction techniques (hankmr).

```
[sysK1_3_cf,sigK1_3] = sncfbal(sysK1_3);
sigK1_3
sigK1_3 =

   3.1674e-01
   2.7851e-01
   6.9959e-02

sysK1_2 = cf2sys(hankmr(sysK1_3_cf,sigK1_3,2,'d'));
gapK_2=nugap(sysK1_3,sysK1_2)
gapK_2 =

  6.9959e-02
```

The robustness bound in the “Loop Shaping Using H• Synthesis” section in Chapter 3, equation 3-23, can now be applied to give a lower bound on robustness,

```
e_bound=sin(asin(emax_3)-asin(gapGW_4)-asin(gapK_2))
e_bound =

   3.7114e-01
```

and this can be compared with the actual stability margin with the reduced order controller as follows.

```
cl_red = starp(p_ic,sysK1_2);
tmp = hinfnorm(cl_red);
e_act=1/tmp(1)
e_act =

   4.0786e-01
```

It is seen that the actual robustness is about half way between the optimal and this lower bound. The important use of the bounds is that they indicate what level of reduction is guaranteed not to degrade robustness significantly.

This gives a third-order controller together with the second-order P+I term. The μ-value for this controller, not shown here, turns out to have essentially the same μ -value as the closed-loop system with the full order controller.

Introducing a Reference Signal

A reference signal can be introduced into the loop shaping control design as follows.

```
[sysK3,emax] = ncfsyn(sysGW,1.1,'ref');
cl_ref = starp(p_ic,sysK3,2,2);
minfo(cl_ref)
system: 12 states  4 outputs  6 inputs
```

When the `ncfsyn` option ref is specified, the controller includes an extra set of reference inputs. The second input argument to `ncfsyn` is 1.1. This implies we are designing a suboptimal controller with 10% less performance than at the optimal. In practice, a 10% suboptimal design often performs better in terms of robust performance than the optimal controller on the actual system.

The last two inputs to cl_ref correspond to the reference signals, the first two outputs are the outputs of the controller and the last two outputs are the inputs to the controller (plant output plus observation noise). This design makes the closed-loop transfer function from reference to plant output the numerator of a normalized coprime factorization of sysGW. An external reference compensator could also be added to improve the command response and there are many possibilities. Here we first diagonalize the closed-loop reference to output transfer function and then insert some phase advance to increase the speed of response.

```
cl_ref_yr=sel(cl_ref,3:4,5:6);
P0 = transp(mmult([0 1; -1 0],cl_ref_yr,[0 1; -1 0]));
P1 = nd2sys([10 50],[1 50]);
P2 = daug(P1,P1);
sysQ = mmult(P0,P2);
```

Now reduce the order of sysQ to four states using the balanced realization technique (sysbal), and incorporate into the controller.

```
[sysQ_b,sig_Q] = sysbal(sysQ);
sig_Q
sig_Q =
   3.9665e+00
   2.9126e+00
   7.2360e-01
   4.5915e-01
   2.3600e-02
   1.0016e-02
   1.2526e-06
   5.2197e-07
sysQ4 = strunc(sysQ_b,4);
sysK_ref = mmult(sysK3,daug(eye(2),sysQ4));
```

Finally form the closed-loop and calculate the step response.

```
sys_cl_ref = starp(p_ic,sysK_ref,2,2);
y = trsp(sys_cl_ref,[0;0;0;0;1;0],0.5,.001);
vplot(sel(y,1,1),'-.',sel(y,2,1),'.',sel(y,3,1),'-',...
   sel(y,4,1),'--')
title('CLOSED LOOP TIME RESPONSE WITH SYSK1')
xlabel('TIME (SECONDS)')
```

The step responses are plotted in Figure 7-66. The first output (solid) tracks the command well with a rise time of less than 0.1 second and no overshoot. The output of the second channel (dashed) is zero, indicating that there is *no* cross coupling between the output channels in the nominal closed-loop system. The controller output commands (dotted and dashed-dotted lines) are also plotted. This is just the nominal step response and further tests are needed to check the sensitivity of the closed-loop to the plant uncertainty.

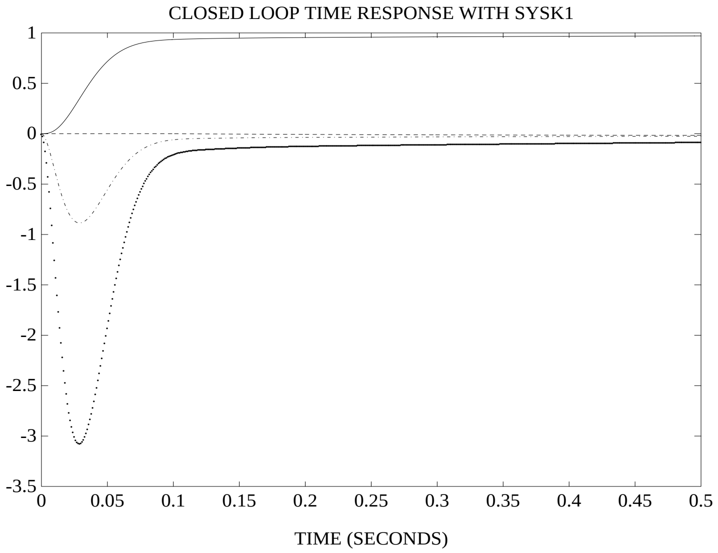


Figure 7-66: Time Response of Closed Loop System: sysK1

HIMAT References

Freudenberg, J., and D. Looze, “Frequency domain properties of scalar and multivariable feedback systems,” *Lecture Notes in Control and Information Sciences*, Springer-Verlag, 1988.

Hartman, G.L., M.F. Barrett and C.S. Greene, “Control designs for an unstable vehicle,” NASA Dryden Flight Research Center, *Contract Report* NAS 4–2578, December, 1979.

Merkel, P.A., and R.A. Whitmoyer, “Development and evaluation of precision control modes for fighter aircraft,” *Proceedings of the AIAA Guidance and Control Conference*, San Diego, CA, Paper 76–1950, 1976.

Safonov, M.G., A.J. Laub, and G.L. Hartman, “Feedback properties of multivariable systems: The role and use of the return difference matrix,” *IEEE Transactions on Automatic Control*, Vol. AC–26, No. 1, pp. 47–65, February, 1981.

F–14 Lateral-Directional Control Design

Consider the design of a lateral-directional axis controller for the F–14 aircraft during powered-approach to landing. The linearized F–14 model is found at an angle-of-attack (α) of 10.5 degrees and airspeed of 140 knots. The control problem is posed as a robust performance, model matching problem with multiplicative plant uncertainty at the plant input and minimization of weighted output transfer functions as the performance criterion. A diagram for the closed-loop system, which includes the feedback structure of the plant and controller, and elements associated with the uncertainty models and performance objectives, is shown in Figure 7-67.

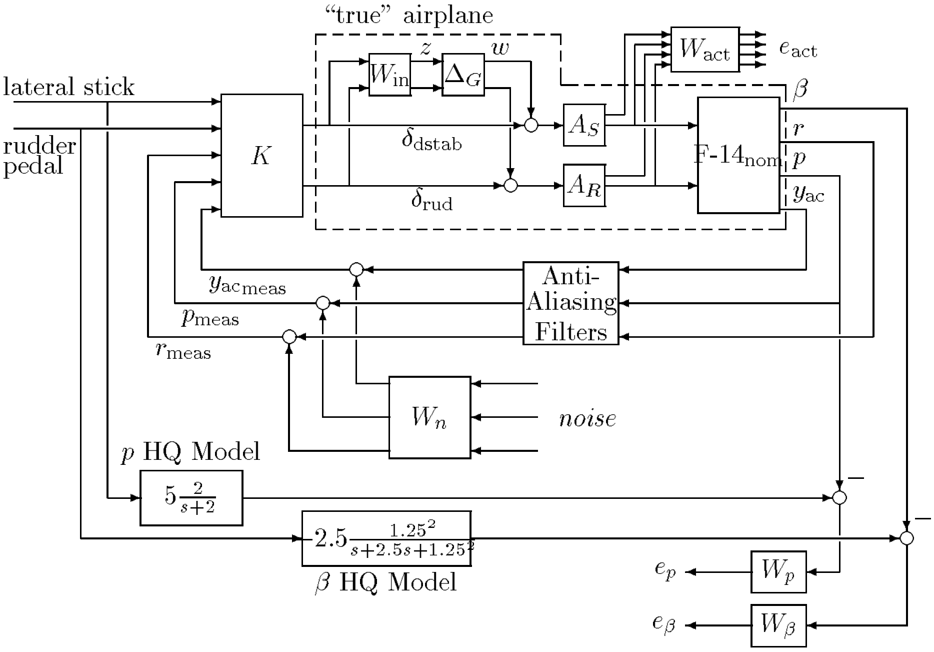


Figure 7-67: F-14 Control Block Diagram

The performance objective is to have the true airplane, represented by the dashed box in Figure 7-67, respond effectively to the pilot's lateral stick and rudder pedal inputs. These performance objectives include:

- Decoupled response of the lateral stick, δ_{lstk}, to roll rate, p, and rudder pedals, δ_{rudp}, to side-slip angle, β. The lateral stick and rudder pedals have a maximum deflection of ± 1 inch. Therefore, they are represented as unweighted signals in Figure 7-67.

The aircraft handling quality (HQ) response from the lateral stick to roll rate should be a first order system, $5\frac{2}{s+2}\frac{deg/sec}{inch}$. The aircraft handling quality response from the rudder pedals to side-slip angle should be

$-2.5\frac{1.25^2}{s^2+2.5s+1.25^2}\frac{deg/sec}{inch}$.

```
hqmod_p = nd2sys(5,[1 2]);
hqmod_beta = nd2sys(1.25^2,[1 2.5 1.25^2],-2.5);
```

The stabilizer actuators have ±20 degs and ±90 degs/sec deflection and deflection rate limits. The rudder actuators have ±30 degs and ±125 degs/sec deflection and deflection rate limits.

- The three measurement signals — roll rate, yaw rate, and lateral acceleration — are passed through second order, anti-aliasing filters prior to being fed to the controller.

$$\text{Anti-aliasing filter} = \frac{\omega^2}{s^2 + 2\zeta\omega + \omega^2}$$

- The natural frequency, ω, and damping, ζ, values for the yaw rate and lateral acceleration filters are 12.5Hz and 0.5, respectively, and 4.1 Hz and 0.7 for the roll rate filter. The anti-aliasing filters have unity gain at DC (see Figure 7-67). These signals are also corrupted by noise prior to entering the controller.

```
freq = 12.5;
fr = freq*2*pi;
zeta = 0.5;
antiaf_yaw = nd2sys(fr^2,[1 2*zeta*fr fr^2]);
antiaf_lata = nd2sys(fr^2,[1 2*zeta*fr fr^2]);
freq = 4.1; fr = freq*2*pi;
zeta = 0.7;
antiaf_roll = nd2sys(fr^2,[1 2*zeta*fr fr^2]);
antia_filt = daug(antiaf_lata,antiaf_roll,antiaf_yaw);
```

The performance objectives are accounted for in this framework via minimizing weight transfer function norms. Weighting functions serve two purposes in the H_∞ and μ framework: they allow the direct comparison of different performance objectives with the same norm and they allow frequency information to be incorporated into the analysis. The F–14 performance weighting functions include:

Limits on the actuator deflection magnitude and rates are included via the $W_{\_act}$ weight. W_{act} is a 4 × 4 constant, diagonal scaling matrix described by $W_{act} = \text{diag}(\frac{1}{90}, \frac{1}{20}, \frac{1}{125}, \frac{1}{30})$. These weights correspond to the stabilizer and rudder deflection rate and deflection limits.

W_{act} `= daug(1/90,1/20,1/125,1/30);`

- W_n is a 3 × 3 diagonal, frequency varying weight used to model the magnitude of the sensor noise $W_n = \text{diag}(0.025, 0.025, 0.0125\frac{s+1}{s+100})$, which corresponds to the noise levels in the roll rate, yaw rate and lateral acceleration channels.

  `W_n = daug(0.025,0.025,nd`$_2$`sys([1 1],[1 100],0.0125);`

- The desired δ_{lstk}-to-p and δ_{rudp}-to-β responses of the aircraft are formulated as a model matching problem in the μ-framework. The difference between the ideal response of the transfer functions, δ_{lstk} filtered through the roll rate HQ model and δ_{rudp} filtered through the side-slip angle HQ model, and the aircraft response, p and β, is used to generate an error that is to be minimized. The W_p transfer function (see Figure 7-67) weights the difference between the idealized roll rate response and the actual aircraft response, *p*.

$$W_p = 0.5 * W_\beta = \frac{0.05s^4 + 2.90s^3 + 105.93s^2 + 6.17s + 0.16}{s^4 + 9.2s^3 + 30.80s^2 + 18.33s + 3.95}$$

The magnitude of W_p emphasizes the frequency range between 0.06 and 30 rad/sec. The desired performance frequency range is limited due to a right-half plane zero in the model at 0.002 rad/sec, therefore accurate tracking of sinusoids below 0.002 rad/sec isn't required. Between 0.06 and 30 rad/sec a roll rate tracking error of less than 5% is desired. The performance weight on the β tracking error, W_β, is just 2 × W_p. This also corresponds to a 5% tracking error objective.

```
W_p = nd2sys([.05 2.9 105.93 6.17 .16],...
   [1 9.2 30.8 18.83 3.95]);
W_beta = mscl(W_p,2);
```

All the weighted performance objectives are scaled to have an H_∞ norm less than 1 when they are achieved. The performance of the closed-loop system is evaluated by calculating the maximum singular value of the weighted transfer functions from the disturbance and command inputs to the error outputs, as shown in Figure 7-68.

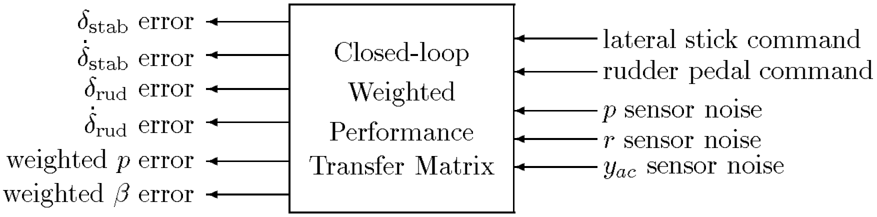

δ_{stab} error
$\dot{\delta}_{\text{stab}}$ error
δ_{rud} error
$\dot{\delta}_{\text{rud}}$ error
weighted p error
weighted β error
Closed-loop Weighted Performance Transfer Matrix
lateral stick command
rudder pedal command
p sensor noise
r sensor noise
y_{ac} sensor noise

Figure 7-68: F-14 Weighted Performance Objectives Transfer Matrix

Nominal Model and Uncertainty Models

The pilot has the ability to command the lateral-directional response of the aircraft with the lateral stick (δ_{lstk}) and rudder pedals (δ_{rped}). The aircraft has two control inputs: differential stabilizer deflection (δ_{dstab}, degrees) and rudder deflection (δ_{rud}, degrees); three measured outputs: roll rate (*p*, degs/sec), yaw rate (*r*, degs/sec) and lateral acceleration (y_{ac}, g's) and a calculated output, side-slip angle (β). Note that β is not a measured variable but is used as a performance measure. The lateral-directional F–14 model, $F14_{nom}$, has four states: lateral velocity (v), yaw rate (r), roll rate (p) and roll angle (ϕ). These variables are related by the state-space equations

$$\begin{bmatrix} \dot{v} \\ \dot{r} \\ \dot{p} \\ \dot{\phi} \\ \beta \\ p \\ r \\ y_{ac} \end{bmatrix} = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] \begin{bmatrix} v \\ r \\ p \\ \phi \\ \delta_{\mathrm{dstab}} \\ \delta_{\mathrm{drud}} \end{bmatrix}$$

$$A = \begin{bmatrix} -1.16e-1 & -2.27e+2 & 4.30e+1 & 3.16e+1 \\ 2.65e-3 & -2.59e-1 & -1.45e-1 & 0.00e+0 \\ -2.11e-2 & 6.70e-1 & -1.36e+0 & 0.00e+0 \\ 0.00e+0 & 1.85e-1 & 1.00e+0 & 0.00e+0 \end{bmatrix},$$

$$B = \begin{bmatrix} 6.22e-02 & 1.01e-1 \\ -5.25e-03 & -1.12e-2 \\ -4.67e-02 & 3.64e-3 \\ 0.00e+00 & 0.00e+0 \end{bmatrix},$$

$$C = \begin{bmatrix} 2.47e-1 & 0.00e+0 & 0.00e+0 & 0.00e+0 \\ 0.00e+0 & 0.00e+0 & 5.73e+1 & 0.00e+0 \\ 0.00e+0 & 5.73e+1 & 0.00e+0 & 0.00e+0 \\ -2.83e-3 & -7.88e-3 & 5.11e-2 & 0.00e+0 \end{bmatrix},$$

$$D = \begin{bmatrix} 0.00e+00 & 0.00e+0 \\ 0.00e+00 & 0.00e+0 \\ 0.00e+00 & 0.00e+0 \\ 2.89e-03 & 2.27e-3 \end{bmatrix},$$

Typing

```
load F14_nom
```

will load the nominal, F–14 plant model into the workspace. The dashed box in Figure 7-67 represents the true airplane, which corresponds to a set of F–14 plant models defined by $\mathcal{G}$. Inside the box is the nominal model of the airplane dynamics, $F14_{nom}$, models of the actuators, A_S and A_R, and two elements, W_{in} and Δ_G, which parameterize the uncertainty in the model. This type of uncertainty is called multiplicative plant input uncertainty. The transfer function W_{in} is assumed known, and reflects the amount of uncertainty in the model. The transfer function Δ_G is assumed to be stable and unknown, except for the norm condition, $\|\Delta_G\|_\infty \le 1$. The aircraft uncertainty is modeled as a complex full-block, multiplicative uncertainty at the input of the rigid body aircraft nominal model. This is the same type of uncertainty description that was used in the previous section entitled “HIMAT Robust Performance Design Example”.

The stabilizer and rudder actuators, A_S and A_R, are modeled as first order transfer functions, $\frac{25}{s+25}$. The actuator outputs are their respective rates and deflections.

```
A_S = pck(-25,25,[-25;1],[25;0]);
A_R = pck(-25,25,[-25;1],[25;0]);
```

Given the actuator and aircraft nominal models (denoted by $G_{\text{nom}}(s)$) we also specify a stable, 2×2 transfer function matrix $W_{\text{in}}(s)$, called the uncertainty weight. These transfer matrices parameterize an entire set of plants, $\mathcal{G}$, which must be suitably controlled by the robust controller K.

$$\mathcal{G} := \{G_{\text{nom}}(I + \Delta_G W_{in}) : \Delta_G \text{ stable}, \|\Delta_G\|_\infty \le 1\}.$$

All of the uncertainty in modeling the airplane is captured in the normalized, unknown transfer function Δ_G. The unknown transfer function $\Delta_G(s)$ is used to parameterize the potential differences between the nominal model, $G_{\text{nom}}(s)$, and the actual behavior of the real airplane, denoted by $\mathcal{G}$.

In this example, the uncertainty weight W_{in} is of the form

$$W_{\text{in}}(s) := \begin{bmatrix} w_1(s) & 0 \\ 0 & w_2(s) \end{bmatrix}$$

for particular scalar valued functions $w_1(s)$ and $w_2(s)$. The $w_1(s)$ weight associated with the differential stabilizer input is selected to be $w_1(s) = \frac{2(s+4)}{s+160}$. The $w_2(s)$ weight associated with the differential rudder input is selected to be $w_1(s) = \frac{1.5(s+20)}{s+200}$.

```
w_1 = nd2sys([1 4],[1 160],2);
w_2 = nd2sys([1 20],[1 200],1.5);
W_in = daug(w_1,w_2);
```

Hence the set of plants that are represented by this uncertainty weight

$$G := \left\{ F14_{\text{nom}} \begin{bmatrix} \frac{25}{s+25} & 0 \\ 0 & \frac{25}{s+25} \end{bmatrix} \left(I_2 + \begin{bmatrix} \frac{2(s+4)}{s+100} & 0 \\ 0 & \frac{1.5(s+20)}{s+200} \end{bmatrix} \Delta_G(s) \right) \right\}$$

with $\Delta_G(s)$ stable and $\|\Delta_G\|_\infty \le 1$. Note that the weighting functions are used to normalize the size of the unknown perturbation Δ_G. At any frequency ω, the value of $|w_1(j\omega)|$ and $|w_2(j\omega)|$ can be interpreted as the percentage of uncertainty in the model at that frequency. The dependence on frequency of the uncertainty weight indicates that the level of uncertainty in the airplane's behavior depends on frequency. Frequency response plots of weights w_1 and w_2 are shown in Figure 7-69.

```
om = logspace(-1,3,120);
W_ing=frsp(W_in,om);
vplot('liv,lm',sel(W_ing,1,1),'-',sel(W_ing,2,2),'--')
xlabel('Frequency (rad/sec)')
ylabel('Magnitude')
```

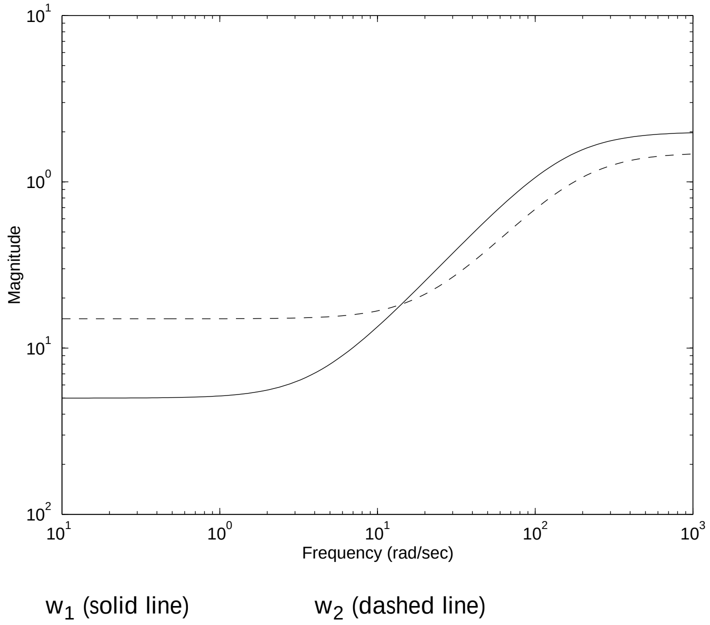


Figure 7-69: F-14 Uncertainty Weights w1, and w_2

The particular uncertainty weights chosen imply that in the differential stabilizer channel at low frequency, there is potentially a 5% modeling error, and at a frequency of 93 rad/sec, the uncertainty in channel 1 can be up to 100%, and could get larger at higher frequencies. The rudder channel has more uncertainty at low frequency, up to 15% modeling error, and at a frequency of 177 rad/sec, the uncertainty is at 100%. To illustrate the variety of plants represented by the set $\mathcal{G}$, type `ex_f14tp` at the command line to generate the step responses of different systems from $\mathcal{G}$shown in Figure 7-70.

```
ex_f14tp
```

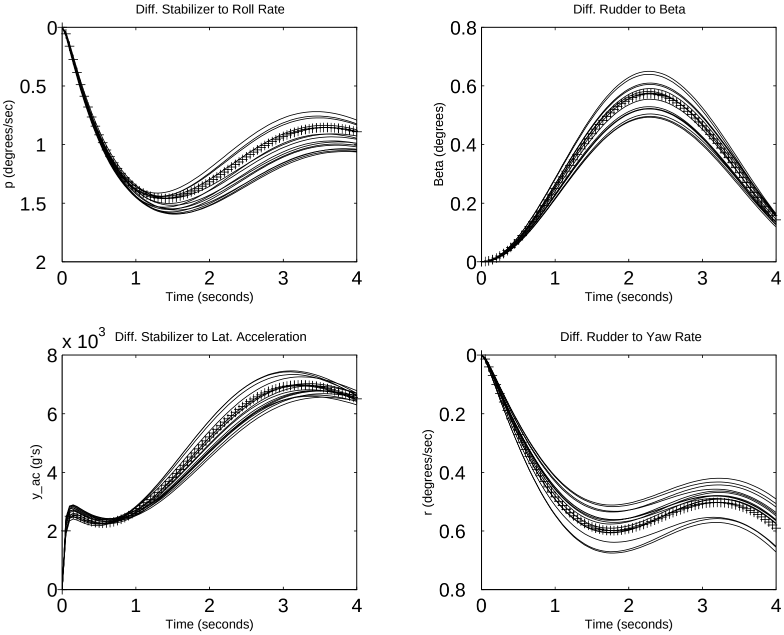


Figure 7-70: Unit Step Responses of the Nominal Model (+) and 15 Perturbed Models from $\mathcal{G}$

The M-file `ex_f14tp` generates the family of perturbed time responses shown in Figure 7-70.

file: ex_f14tp.m

```
Gnom = mmult(F14_nom,daug(sel(A_S,2,1),sel(A_R,2,1)));
u = step_tr(0,1,.02,2);
ydstab = trsp(Gnom,abv(u,0),4,.05);
ydrud = trsp(Gnom,abv(0,u),4,.05);
   for i=1:15
   delta = randn(2,2);
   delta = delta/norm(delta);
   p = mmult(Gnom,madd(eye(2),mmult(W_in,delta)));
   y1 = trsp(p,abv(u,0),4,.05);
   y2 = trsp(p,abv(0,u),4,.05);
   ydstab = sbs(ydstab,y1);
   ydrud = sbs(ydrud,y2);
end
cold = ynum(ydrud);
index = 2:cold;
subplot(221)
   vplot(sel(ydstab,2,1),'+',sel(ydstab,2,[index]))
   title('Diff. Stabilizer to Roll Rate')
   xlabel('Time (seconds)'), ylabel('p (degrees/sec)')
subplot(222)
   vplot(sel(ydrud,1,1),'+',sel(ydrud,1,[index]))
   title('Diff. Rudder to Beta')
   xlabel('Time (seconds)'), ylabel('Beta (degrees)')
subplot(223)
   vplot(sel(ydstab,4,1),'+',sel(ydstab,4,[index]))
   title('Diff. Stabilizer to Lat. Acceleration')
   xlabel('Time (seconds)'), ylabel('ac_y (g''s)')
subplot(224)
   vplot(sel(ydrud,3,1),'+',sel(ydrud,3,[index]))
   title('Diff. Rudder to Yaw Rate')
   xlabel('Time (seconds)'), ylabel('r (degrees/sec)')
```

The control design objective is to design a stabilizing controller K such that, for all stable perturbations $\Delta_G(s)$, with $\|\Delta_G\|_\infty \leq 1$, the perturbed closed-loop system remains stable, and the perturbed weighted performance transfer function has an H_∞ norm less than 1 for all such perturbations. These mathematical objectives exactly fit in the structured singular value framework.

Controller Design

The control design block diagram shown in Figure 7-67 is redrawn as F14IC shown in Figure 7-71. F14IC is the 25-state, six-input, six-output open-loop transfer matrix used for control design. The M-file ex_f14ic contains the sysic commands to generate the F14IC interconnection structure. The M-file ex_f14wt, called from ex_f14ic, creates the weighting functions (W_{act}, W_{in}, W_n, W_p, and W_β), the handling qualities models (hqmod_beta, hqmod_p), anti-aliasing filters (anti_filt), the actuator models (A_S and A_R) and loads the nominal F–14 plant model.

```
ex_f14ic
```

file: ex_f14ic.m

```
ex_f14wt
systemnames = 'W_in A_S A_R antia_filt hqmod_p hqmod_beta';
systemnames = [systemnames 'F14_nom W_act W_n W_P W_beta'];
inputvar = '[in_unc{2}; sn_nois{3}; roll_cmd; beta_cmd; ';
inputvar = [inputvar ' delta_dstab; delta_rud]' ];
outputvar = '[ W_in; W_P; W_beta; W_act; roll_cmd; ';
outputvar = [outputvar 'beta_cmd; antia_filt + W_n ]' ];
input_to_W_in = '[ delta_dstab; delta_rud ]';
input_to_A_S  = '[ delta_dstab + in_unc(1) ]';
input_to_A_R  = '[ delta_rud + in_unc(2) ]';
input_to_W_act = '[ A_S; A_R ]';
input_to_F14_nom = '[ A_S(1); A_R(1) ]';
input_to_antia_filt = '[ F14_nom(4) F14_nom(3) F14_nom(2)]';
input_to_hqmod_beta = '[ beta_cmd ]';
input_to_hqmod_p = '[ roll_cmd ]';
input_to_W_beta = '[ hqmod_beta - F14_nom(1) ]';
input_to_W_P  = '[ hqmod_p - F14_nom(3) ]';
input_to_W_n  = '[ sn_nois ]';
sysoutname = 'F14IC';
cleanupsysic = 'yes';
sysic
```

The first step in the $D - K$ iteration control design procedure is to design an H_∞ (sub)optimal controller for the open-loop interconnection, F14IC. In terms of the $D - K$ iteration, this amounts to holding the d variable fixed (at 1), and minimizing the $\|\cdot\|_\infty$ norm of F_L(F14IC, K), over the controller variable K. The resulting controller is labeled K_1.

The second step in $D - K$ iteration involves solving a μ analysis corresponding to the closed-loop system, F_L(F14IC, K_1). This calculation produces a frequency dependent scaling variable d_ω, the (1,1) entry in the scaling matrix. In a general problem (with more than two blocks), there would be several d variables, and the overall matrix is referred to as the D-scales. The varying variables in the D-scales are fit (in magnitude) with proper, stable, minimum phase rational functions and absorbed into the generalized plant for additional iterations. These scalings are used to trick the H_∞ minimization to concentrate more on minimizing μ rather than $\bar{\sigma}$ across frequency. For the first iteration in this example, the d scale data is fit with a first order transfer function.

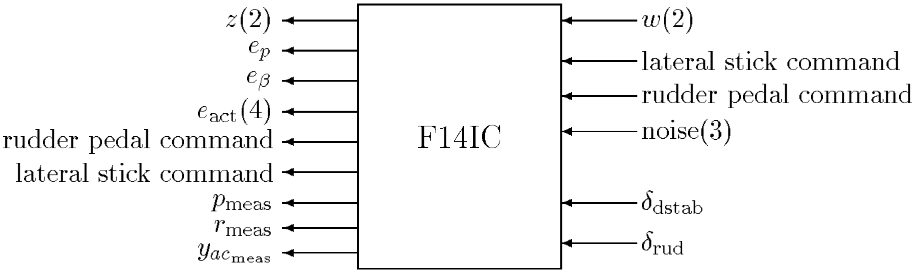


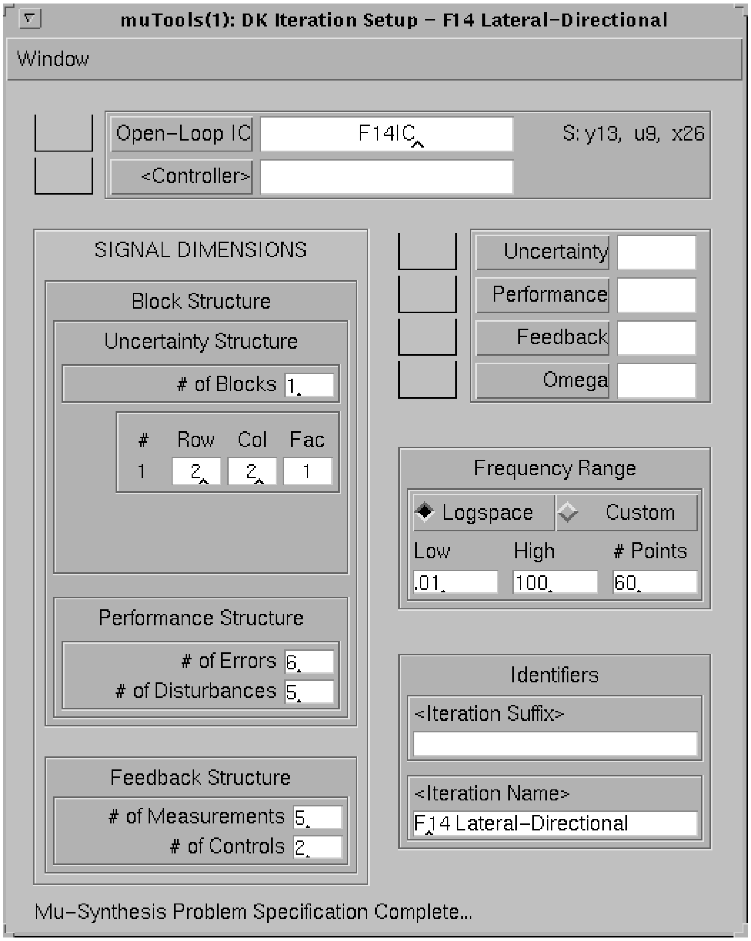
Figure 7-71: F-14 Generalized Plant

The new generalized plant used in the second iteration has 29 states, 4 more states than the original 25-state generalized plant, F14IC. These extra states are due to the D-scale data being fit with a rational function, and absorbed into the generalized plant for the next iteration. Four $D - K$ iterations are performed until μ reaches a value of 1.02. Information about the $D - K$ iterations is shown in Table 7-1.

Table 7-1 F-14 D – K Iteration Information

Iteration Number	1	2	3	4
Total D-Scale Order	0	4	4	4
Controller Order	25	29	29	29
H_∞ Norm Achieved	1.562	1.079	1.025	1.017
Peak μ Value	1.443	1.079	1.025	1.017

To replicate these results using $D-K$ iteration, start up `dkitgui` and press the **SETUP** button in the main window. The data required in the **DK Iteration Setup** window should be filled in to duplicate the **Setup** window shown in Figure 7-72. The message "`Mu-Synthesis Problem Specification Complete...`" will appear in the message bar upon correctly entering the required data. Return to the main `dkitgui` window and press the **Control Design** button. This will synthesize controller K_1. To run 4 automated $D-K$ iterations, pull down the `Iteration` menu and select the number 4 from the `Auto Iterate` menu.



Open-Loop IC: F14IC   S: y13, u9, x26

<Controller>

# of Blocks: 1

#	Row	Col	Fac
1	2	2	1

# of Errors: 6
# of Disturbances: 5

# of Measurements: 5
# of Controls: 2

● Logspace ◇ Custom

Low	High	# Points
.01	100	60

<Iteration Suffix>

<Iteration Name> F14 Lateral–Directional

Mu–Synthesis Problem Specification Complete...

Figure 7-72: dkitgui F-14 D – K Iteration Setup Window

Analysis of the Controllers

The robust performance properties of the controllers can be analyzed using μ-analysis methods. Recall that robust performance is achieved if and only if for every frequency, $\mu_\Delta(F_L(\text{F14IC},K)(j\omega))$ is less than 1. Plots of robust performance μ of the closed-loop system with K_1 and K_4 implemented are shown in Figure 7-73. The M-file `ex_f14mu` generates Figure 7-73 given that the original generalized plant, F14IC, and the controller SYSTEM matrices, K_1 and K_4, are in your MATLAB workspace.

```
ex_f14mu
```

file: ex_f14mu.m

```
om = logspace(-2,2,60);
clp1 = starp(F14IC,K1);
clp4 = starp(F14IC,K4);
clp1g = frsp(clp1,om);
clp4g = frsp(clp4,om);
deltaset = [2 2; 5 6];
mubnds1 = mu(clp1g,deltaset);
mubnds4 = mu(clp4g,deltaset);
vplot('liv,lm',mubnds1,'-',mubnds4,'--')
xlabel('Frequency (rad/sec)')
```

The controlled system with K_1 implemented does not achieve robust performance. This conclusion follows from the μ plot, which peaks to a value of 1.44, at a frequency of 7 rad/sec. Since μ is 1.44, there is a perturbation matrix Δ_G, such that $\|\Delta_G\|_\infty = \frac{1}{1.44}$, and the perturbed weighted performance transfer functions gets large. After four $D - K$ iterations the peak robust performance μ value is reduced to 1.02 (Figure 7-73), thereby nearly achieving all of our robust performance objectives.

To illustrate the robustness of the closed-loop system in the time-domain, time responses of the ideal model, the nominal closed-loop system and the worst-case closed-loop system from $\mathscr{G}$(using perturbations of size 1) are shown in Figure 7-74. Controller K_4 is implemented in the closed-loop simulations. A 1-inch lateral stick command is given at nine seconds, held at 1-inch till 12 seconds, and then returns to zero. The rudder is commanded at one second with a positive 1-inch pedal deflection and held at 1-inch till four seconds. At four seconds a –1-inch pedal deflection is commanded and held to seven seconds and then returned to zero. One can see from the time responses that the closed-loop response is nearly identical for the nominal closed-loop system and the worst-case closed-loop system.

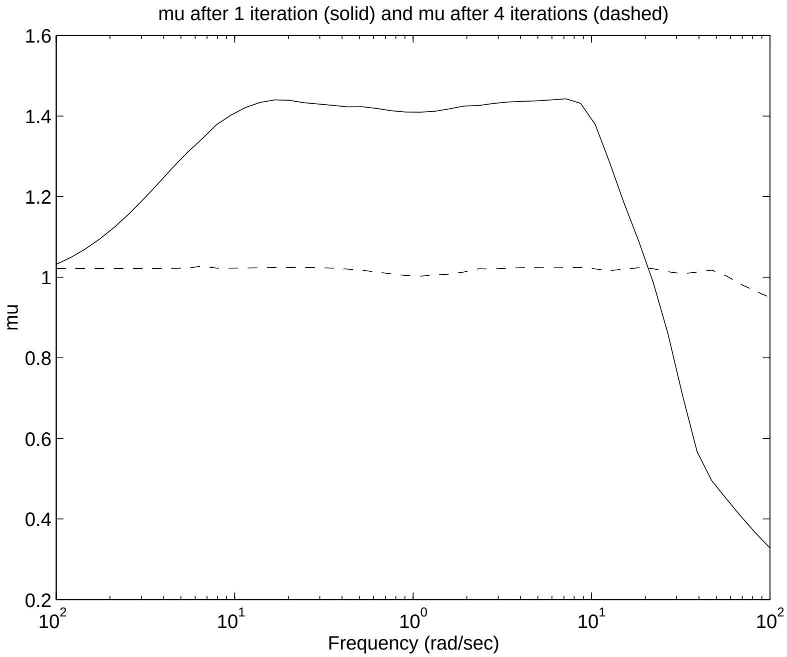


Figure 7-73: F-14 Robust Performance μ Plots with *K*1 and *K*4 Implemented

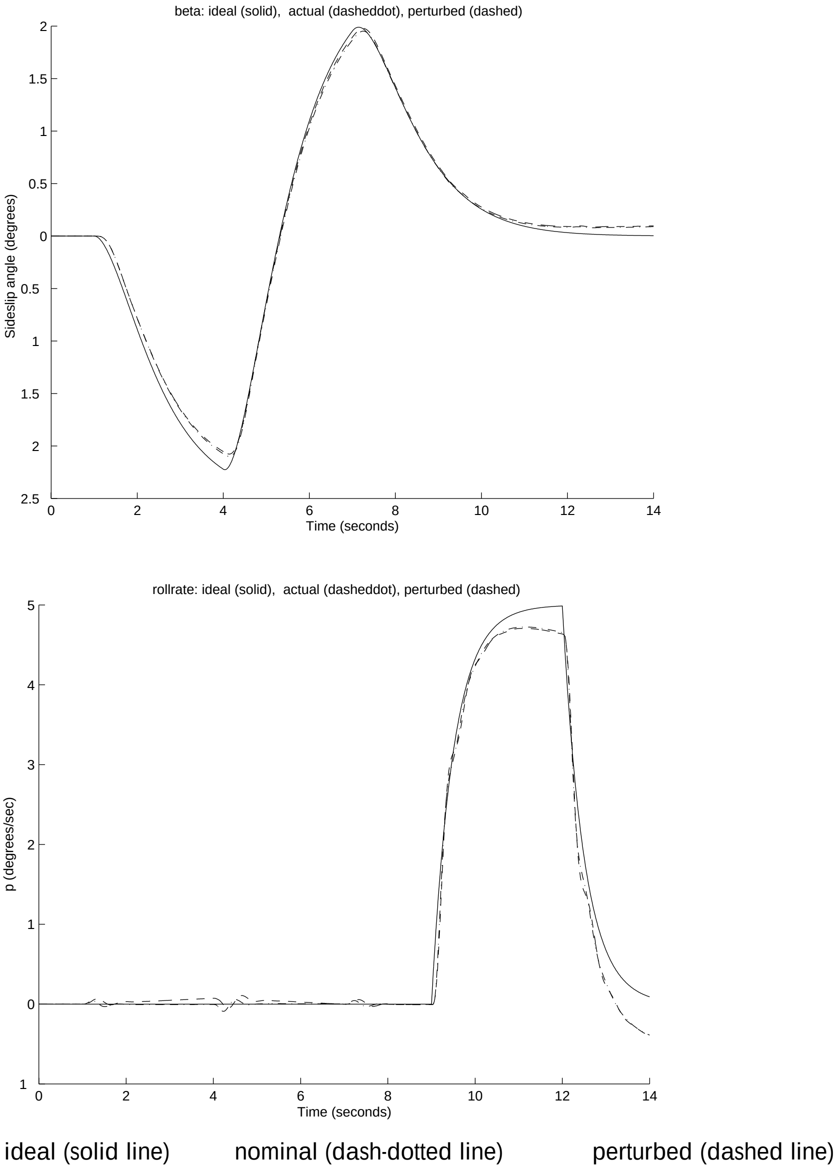


Figure 7-74: Time Response Plots of the F-14 Lateral-Directional Control System: Ideal, Nominal, and Perturbed

The M-file ex_f14s1 contains the sysic commands to generate the F14IC simulation interconnection structure. ex_f14s2 contains the commands to calculate the worst-case perturbation of size 1 for K_4, the closed-loop time response of the nominal and perturbed systems, and plots the results.

```
ex_f14s1
ex_f14s2
```

Figure 7-74 validates the frequency domain results showing that the controller synthesized via $D - K$ iteration, K_4, is insensitive to changes in the model. You will notice that the roll-rate response of the F–14 tracks the roll-rate command well initially and then departs from the command. This is due to a right-half plane zero in the aircraft model at 0.024 rad/sec.

file: ex_f14s1.m

```
systemnames = 'Win A_S A_R F14_nom antia_filt hqmod_p ';
systemnames = [systemnames ' hqmod_beta '];
inputvar = '[in_unc{2}; roll_cmd; beta_cmd; ';
inputvar = [inputvar ' delta_dstab; delta_rud]' ];
outputvar = '[ W_in; hqmod_p; F14_nom(2); hqmod_beta; '
outputvar = [outputvar ' F14_nom(1); roll_cmd; beta_cmd; '];
outputvar = [outputvar 'antia_filt ]' ];
input_to_W_in = '[ delta_dstab ; delta_rud ]';
input_to_A_S  = '[ delta_dstab + in_unc(1) ]';
input_to_A_R  = '[ delta_rud + in_unc(2) ]';
input_to_F14_nom = '[ AS(1); AR(1) ]';
input_to_antia_filt = '[F14_nom(4); F14_nom(3); F14_nom(2)]';
input_to_hqmod_beta = '[ beta_cmd ]';
input_to_hqmod_p = '[ roll_cmd ]';
sysoutname = 'F14SIM';
cleanupsysic = 'yes';
sysic
```

file: ex_f14s2.m

```
om = logspace(-2,2,40);
delta = wcperf(frsp(clp4,om),deltaset,1,1);
sclp4_nom = starp(zeros(2,2),starp(F14SIM,K4));
sclp4_pert = starp(delta,starp(F14SIM,K4));
ustk = step_tr([0 1 4],[0 1 0],.02,10);
upedal = step_tr([0 1 4 7 ],[0 1 -1 0],.02,10);
input = abv(ustk,upedal);
y4nom = trsp(sclp4_nom,input,14,0.02);
y4pert = trsp(sclp4_pert,input,14,0.02);
subplot(211), vplot(sel(y4nom,3,1),'-',sel(y4nom,4,1),...
     '-.',sel(y4pert,4,1),'--')
   xlabel('Time (seconds)')
   ylabel('Side-slip angle (degrees)')
   title('beta: ideal (solid), actual (dashed-dot),...
     perturbed (dashed)')
subplot(212), vplot(sel(y4nom,1,1),'-',sel(y4nom,2,1),...
     '-.',sel(y4pert,2,1),'--')
   xlabel('Time (seconds)')
   ylabel('Roll rate (degrees/sec)')
   title('roll-rate: ideal (solid), actual (dashed-dot),...
     perturbed (dashed)')
```

A Process Control Example: Two Tank System

In this example we provide a relatively complete description of the entire modeling, identification, and design process. The experiment is a simple two tank system at Caltech. Other experimental work relating to this system is described by Smith *et al*. [SmDMS, SmD1, SmD2].

Experimental Description

The system consists of two water tanks in cascade and is shown schematically in Figure 7-75. The upper tank (tank 1) is fed by hot and cold water via computer controllable valves. The lower tank (tank 2) is fed by water from an exit at the bottom of tank 1. A constant level is maintained in tank 2 by means of an overflow. A cold water bias stream also feeds tank 2 and enables the tanks to have different steady-state temperatures.

Tank 1 is $5\frac{3}{8}$ inches in diameter and 30 inches in height. Tank 2 is $7\frac{1}{2}$ inches in diameter and the overflow maintains the water level at $7\frac{1}{4}$ inches. This configuration maintains the water level in tank 2 at $4\frac{3}{4}$ inches below the base of tank 1. Flow control is obtained via linear electropneumatic actuators with a $\mathbf{C}_V$ of 1.0. One hundred inches of $\frac{1}{2}$-inch piping runs from each valve to the top of tank 1. Approximately 36 inches of pipe connect the tanks, from the base of tank 1 to the base of tank 2. The tank 2 cold water bias stream is manually adjustable between 0.015 and 0.3 gpm. Thermocouples are mounted $\frac{1}{4}$ inch above the base of each tank. A pressure sensor (0 to 5 psig) provides a measurement of the water level in tank 1.

All measured signals are filtered with fourth order Butterworth filters, each with a cutoff frequency of 2.25 Hz. Twelve bit resolution is used for the A/D and D/A conversions. In digital implementations of the controllers a sample period of 0.1 seconds has been used.

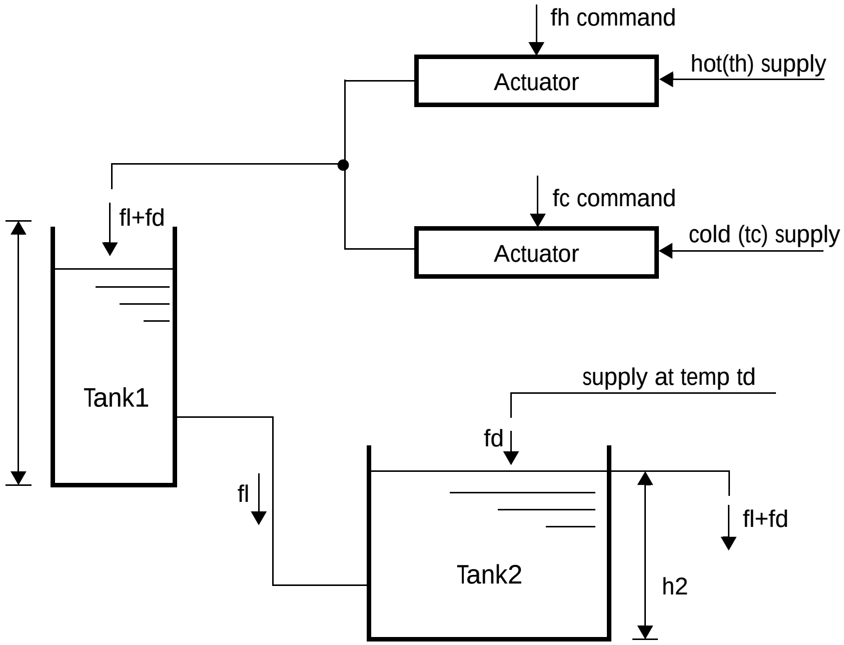


Figure 7-75: Schematic Diagram of the Two Tank System

An Idealized Nonlinear Model

The Two tank example, nonlinear model system is first considered without the actuators or sensors and an ideal nonlinear model is derived. Suitable units are added to give a basis for the subsequent discussion of the experimental data. The model of the actuators and sensors (including noise) is based on experimental data and is included later.

We make the following unrealistic assumptions:

- There are no thermal losses in the system.
- Perfect mixing occurs in both tanks.
- The flow out of tank 1 is related only to the height of tank 1.
- There are no thermal or flow delays.

The system variables are given the following designations.

Variable	Physical Quantity
f_{hc}	Command to hot flow actuator
f_h	Hot water flow into tank 1
f_{cc}	Command to cold flow actuator
f_c	Cold water flow into tank 1
f_1	Total flow out of tank 1
A_1	Cross-sectional area of tank 1
h_1	Tank 1 water level
t_1	Temperature of tank 1
t_2	Temperature of tank 2
A_2	Cross-sectional area of tank 2
h_2	Tank 2 water level
f_b	Flow rate of tank 2 bias stream
t_b	Temperature of tank 2 bias stream
t_h	Hot water supply temperature
t_c	Cold water supply temperature

Tank 1 is considered first. Conservation of mass gives,

$$\frac{\mathrm{d}}{\mathrm{dt}}(A_1 h_1) = f_h + f_c - f_1.$$

(7-2)

It is assumed that the flow out of tank 1 (f_1) is a memoryless function of the height (h_1). As the exit from tank 1 is a pipe with a large length to diameter ratio, the flow is proportional to the pressure drop across the pipe and thus to the height in the tank. With a constant correction term for the flow behavior at low tank levels the height and flow can be reasonably approximated by an affine function,

$$h_1 = \alpha f_1 - \beta \quad \text{where} \quad \alpha, \beta > 0 \quad \text{and} \quad f_1 \geq \beta / \alpha.$$

(7-3)

Defining f_1 as a state variable leads to a linear state equation and an affine output equation for h_1 (in the allowable range of f_1).

$$\dot{f}_1 = \frac{-1}{A_1\alpha} f_1 + \frac{1}{A_1\alpha} f_h + \frac{1}{A_1\alpha} f_c.$$

(7-4)

$$h_1 = \alpha f_1 - \beta.$$

(7-5)

Conservation of energy leads to a model for the temperature of tank 1 (t_1). It is useful to define a variable,

$$E_1 = h_1 t_1,$$

(7-6)

which can loosely be thought of as the energy in tank 1. Defining E_1 as a state variable gives a nonlinear state equation and a nonlinear output equation for t_1.

$$\dot{E}_1 = \frac{-1}{A_1\alpha}\left(1 + \frac{\beta}{h_1}\right) E_1 + \frac{t_h}{A_1} f_h + \frac{t_c}{A_1} f_c.$$

(7-7)

$$t_1 = \frac{1}{h_1} E_1.$$

(7-8)

Note that for a fixed h_1 the above equations are linear. This will aid in the identification.

In tank 2 the height (h_2) is fixed and the input flow from tank 1 (f_1) is of temperature t_1. This gives only one equation.

$$\frac{\mathrm{d}}{\mathrm{dt}}(A_2 h_2 t_2) = f_1 t_1 + f_b t_b - (f_1 + f_b) t_2. \tag{7-9}$$

We will develop a model which has t_1 and h_1 as inputs and t_2 as an output. This will allow us to concatenate the tank 1 and tank 2 models to give a model for the full system. A more physically motivated model might have t_1 and f_1 as inputs. The linearizations will differ only by the factor α, so the difference is not significant.

We can rearrange equation 7-9 to give,

$$\dot{t}_2 = -\left(\frac{h_1 + \beta + \alpha f_b}{\alpha A_2 h_2}\right) t_2 + \left(\frac{h_1 + \beta}{\alpha A_2 h_2}\right) t_1 + \frac{f_b t_b}{A_2 h_2} \tag{7-10}$$

The output equation for tank 2 is trivial.

Equations 7-4, 7-5, 7-7, 7-8, and 7-10 provide a simple nonlinear model for the two tanks. To complete the description of the system a model must be obtained for the actuators and sensors. The uncertainty associated with elements of the model must also be identified. First, we define a suitable set of units for our model.

Normalization Units

To quantify the model we define a system of normalized units as follows.

Physical Variable	Unit Name	0 Unit Definition	1 Unit Definition
temperature	tunit	t_c = 1.0 tunit	t_h = 0.0 tunit
height	hunit	tank 1 empty	tank 1 full
flow	funit	zero input flow	maximum (2.0 gpm) input flow

The first two definitions are sufficient to define all of the other units in the problem. The input flows range from 0 to 2.0 gallons/minute and it is convenient to define a flow unit at the input by 2.0 gpm = 1.0 funit. Using the above units the system dimensions are now,

Variable	Value	Units
A_1	0.0256	hunits^2
A_2	0.0477	hunits2
h_2	0.241	hunits
f_b	7.4×10^{-5}	hunits^3/sec
t_b	0.0	tunits
f_s	0.00028	hunits^3/sec/funit

The variable f_s is a flow scaling factor which converts the input (0 to 1 funits) to flow in hunits^3/second. This is used in the `tankdemo` script.

Operating Range

The physical system imposes constraints on the operating region. The most obvious of these is that as the bias stream is cold, the temperature of tank 2 (t_2) must be less than that of tank 1 (t_1). Saturation in the actuators prevents tank 1 from being completely full of either hot or cold water. The relationship between f_1 and h_1 can only be modeled by equation 7-3 for h_1 in the range:

$$0.15 \leq h_1 \leq 0.75.$$

The numerical model given below applies only to this range.

In the linear design example the operating range of tank 1 is further reduced to:

$$0.25 \leq h_1 \leq 0.75.$$
$$0.25 \leq t_1 \leq 0.75.$$

Both sets of constraints severely limit the operating regions.

Actuator Model

There are significant dynamics and saturations associated with the actuators and a model of these is included. In the frequency range of interest the actuators can be modeled as a single pole system with rate and magnitude saturations. The rate saturation has been estimated from observing the effect of triangle waves of different frequencies and magnitudes. The following model will be used for the actuators.

$$fh = \left[\frac{1}{(1 + 0.05s)}\right] fhc \tag{7-11}$$

with a magnitude limit of 1.0 funits and a rate limit of 3.5 funits/sec. It is the rate limit, rather than the pole location, that limits the actuator performance for most signals. For a linear model some of the effects of rate limiting can be included in a perturbation model.

Experimental Assessment of the Model

Where possible, the system has been broken into subsystems, which can be identified independently. For example, the actuators are considered independently and tank 1 is identified separately from tank 2. This approach is not necessarily general.

Open-Loop Experiments

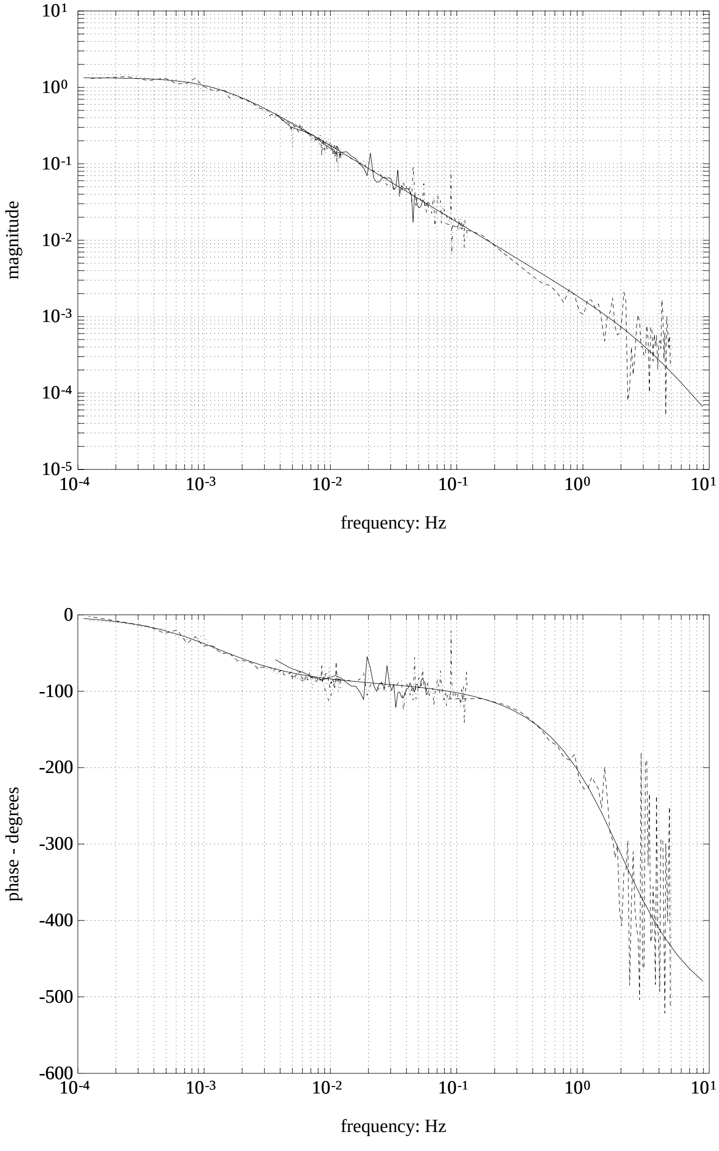
Static measurements of h_1 and f_1 can be used to obtain estimates of α and β from the theoretical relationship: $h_1 = \alpha f_1 - \beta$. Note that α appears as a gain in the nominal model and can also be easily estimated by dynamic measurements. Using such experiments gives the estimates, $\alpha = 4876$ and $\beta = 0.59$.

Open loop experiments have been conducted to test the applicability of the model and obtain some feel for the level of uncertainty that will be appropriate. Band limited white noise, in several frequency bands, was used for the input signals. Data records were 8192 samples in length with sample rates of 1.0 Hz and 10.0 Hz. The transfer function estimates, presented below, have been obtained by the Welch method, using Hanning windows on sections of the data. The data plotted in the figures comes from several window lengths, typically 1024 and 4096. Only the points with good coherency are plotted.

Equation 7-76 shows the estimated transfer function between $f_{hc} + f_{cc}$ and h_1, and the transfer function predicted from the model (equations 7-4 and 7-5), including the nominal actuator and Butterworth filter. The experimental data comes from five experiments at three different levels. For frequencies greater than 0.2 Hz, the plotted data comes from an experiment at $h_1 = 0.47$.

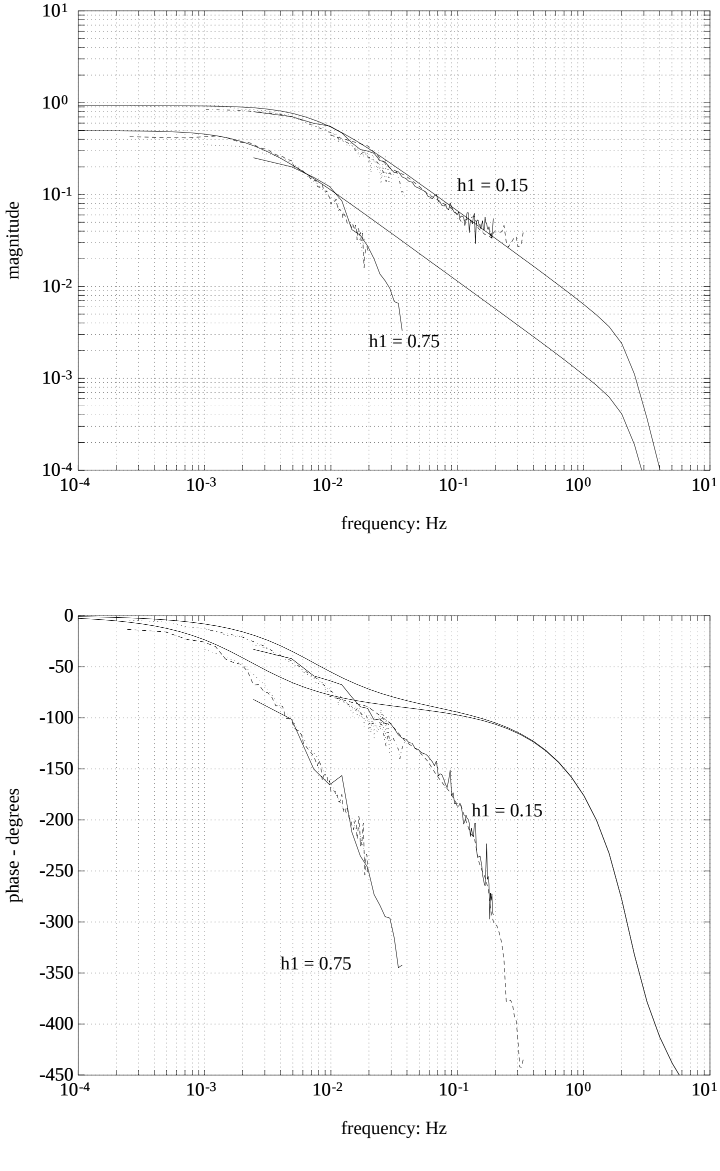
Note that the theoretical model is a very close match to the experimental data over a wide range of frequencies. This suggests that a small perturbation weight is suitable for modeling the discrepancies between the nominal model and physical system behavior. Note also that the h_1 model is independent of both h_1 and t_1. In other words it applies to the entire range of operating points.

For h_1 fixed, the E_1 state variable equation 7-7 and the t_1 output equation 7-8 are linear. Experiments have been performed at $h_1 = 0.15$, 0.25, 0.47, and 0.75. The input waveforms were generated such that $f_{hc} = -f_{cc}$ which maintains a constant h_1. Figure 7-77 shows the transfer function between $f_{hc} - f_{cc}(= 2f_{hc})$ and t_1 calculated from the experimental data and estimated from the model (equations 7-7 and 7-8. For the data shown $h_1 = 0.15$ and $h_1 = 0.75$. The other cases lie between the two curves shown.



experimental data
and theoretical
model (solid line)

Figure 7-76: Transfer Function Between $f_{hc} + f_{cc}$ and h_1. Experimental Data and Theoretical Model



experimental data
and models
(solid lines)

Figure 7-77: Transfer Function Between $f_{hc} - f_{cc}$ and t_1. Experimental Data and Models. h_1 = 0.15 and h_1 = 0.75.

The nominal model/physical system discrepancies are far more significant for the t_1 case. A larger t_1 perturbation weight is required to cover these discrepancies. A more complete discussion on this issue is given in the “Perturbation Model” section.

Similar analyses have been performed on tank 2. The open loop data is less conclusive due to observing the system through tank 1 and a higher noise level. The available data does confirm the theoretical model but to a lower frequency than that for tank 1. More definitive results have been obtained for tank 2 with the closed-loop experiments.

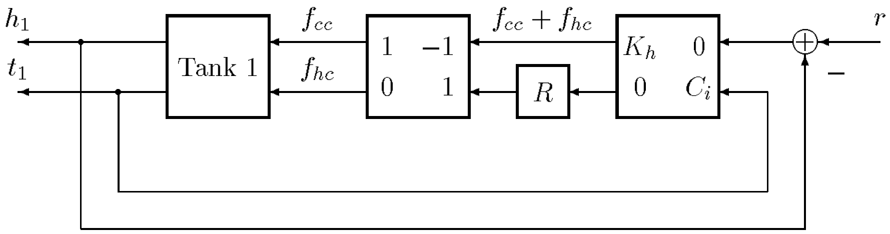
Closed-Loop Experiments

We will look at a closed-loop method of estimating a suitable uncertainty level for the model. This involves using a relay to induce limit cycling and is based on an auto-tuning method proposed by Åström and Hägglund [AstH]. More detail on using such approaches for estimating uncertainty levels is described by Smith and Doyle [SmD2].

Applying a relay in a feedback loop may drive the closed-loop system into stable limit cycles. This technique works for a large class of systems including the two tank system. In the experiments performed here a decoupled controller (into height and temperature loops) was used with a relay in the temperature control loop. This allowed stable control of the tank height (h_1) and produced limit cycles in t_1. These experiments were performed at fixed heights ($h_1 = 0.15$, 0.25, 0.47, and 0.75).

With a simple relay the closed-loop system will limit cycle at the frequency where the response has a phase of 180 degrees. The gain at this frequency can also be estimated from the input/output data. This experiment will identify the system at a single point. Using this information, a new controller is designed to introduce some lead into the closed-loop system. This new closed-loop system has limit cycles at a higher frequency giving an additional point at which the plant can be identified. In practice this technique can be repeated until the nonlinear and/or inconsistent effects dominate and the closed-loop system no longer limit cycles consistently. This also provides information on the frequency at which uncertainty should dominate in the model.

Details of the application of this approach to tank 1 are given in [SmD2]. To illustrate the concept, the configuration used to induce limit cycles in t_1 is shown in Figure 7-78.



Height, h, is controlled by the proportional controller, Kh. R denotes the relay and Ci denotes one of a series of lead controllers used to adjust the limit cycle frequency.

Figure 7-78: Closed-Loop Configuration for Relay Limit Cycle Experiments of t_1.

A series of controllers, C_i in Figure 7-78, was applied. These were used to introduce differing amounts of lead into the closed-loop system, giving differing limit cycle frequencies and amplitudes. A typical time response for such an experiment is shown in Figure 7-79. In this manner limit cycles were induced in the t_1 loop. The highest frequency at which a limit cycle could be induced was 0.023 Hz. We will subsequently see that this can be used as a heuristic for determining an appropriate perturbation weight for t_1.

The tank 2 model was similarly studied. Figure 7-80 presents the limit cycle data obtained for tank 2. The E_1 to t_2 theoretical transfer function (for fixed h_1) is shown for h_1 = 0.15, 0.25, 0.47, and 0.75. Also shown are the points identified from each experiment. It was not possible to induce consistent limit cycles at frequencies above 0.03 Hz, indicating that this is a frequency at which the uncertainty should dominate in the model.

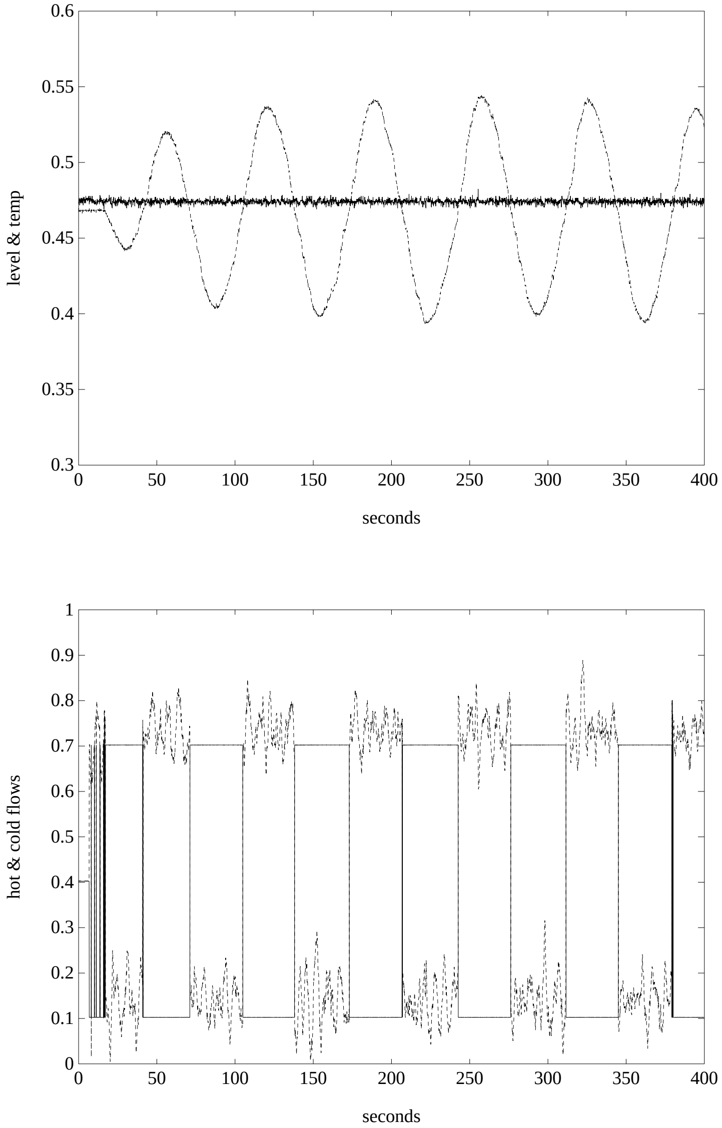


Figure 7-79: Closed-Loop Relay Limit Cycle Experiment Controller: C_1

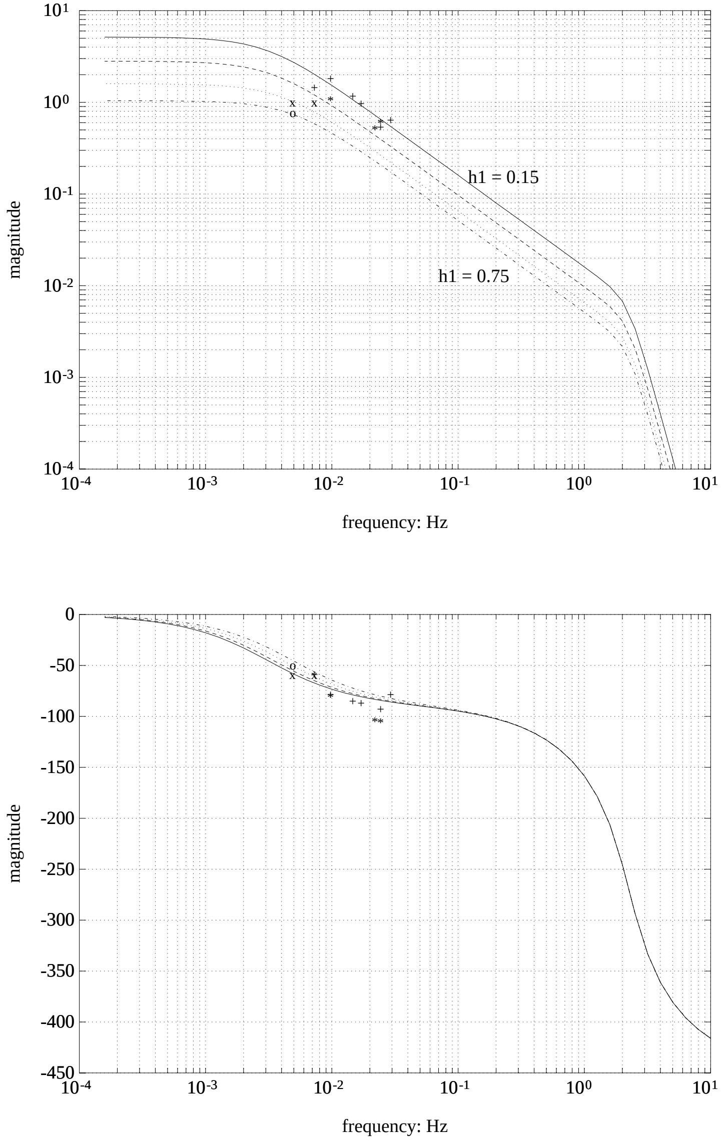
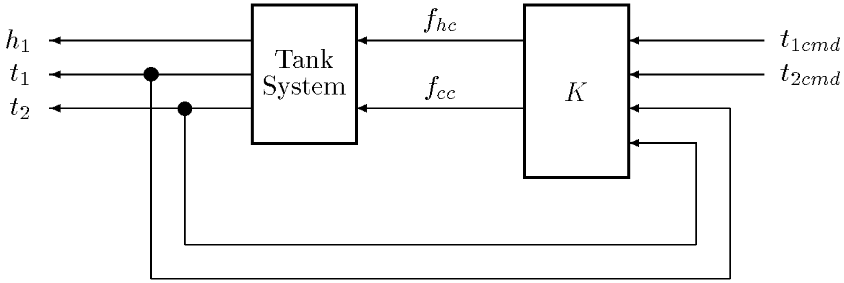


Figure 7-80: Limit Cycle Identification Experiments for Tank 2. Tank 1 Height Held Fixed at h_1 = 0.15, 0.25, 0.47, and 0.75

Developing the Interconnection Model for Design

The problem considered is the control of t_1 and t_2 for command response. The desired closed-loop configuration is illustrated in Figure 7-81. The controller has access to both the reference inputs and the temperature measurements. It would be possible (although more restrictive) to design a controller which used only the error between the temperature measurements and the setpoints.



The t1 and t2 reference signals are denoted by t1cmd and t2cmd, respectively. Noise is assumed to enter within the tank system block.

Figure 7-81: Closed-Loop Design Problem

The layout of the interconnection structure is illustrated in Figure 7-82. The subsequent sections will develop a perturbed model and the necessary design weights for the design problem.

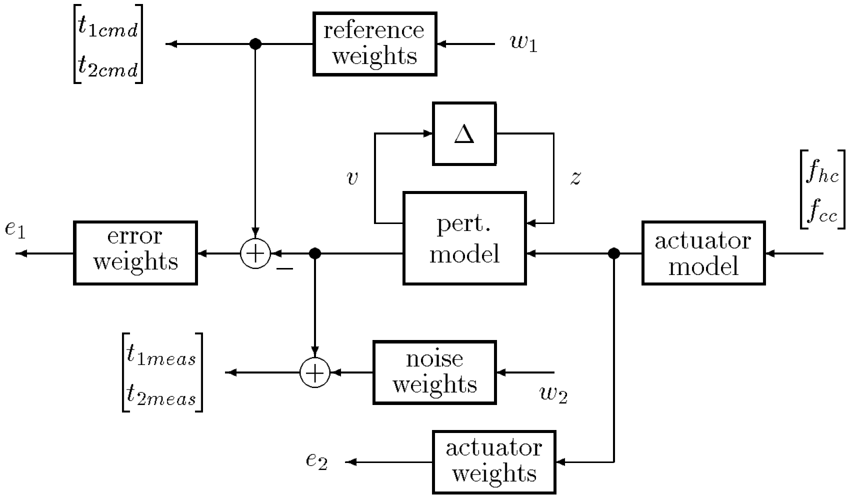

$\begin{bmatrix} t_{1cmd} \\ t_{2cmd} \end{bmatrix}$
reference weights
w_1
Δ
v
z
$\begin{bmatrix} f_{hc} \\ f_{cc} \end{bmatrix}$
e_1
error weights
$-$
pert. model
actuator model
$\begin{bmatrix} t_{1meas} \\ t_{2meas} \end{bmatrix}$
noise weights
w_2
e_2
actuator weights

Figure 7-82: Interconnection Structure for the Design Problem

Linearized Nominal Model

The top tank model, given in equations 7-4, 7-5, 7-7, and 7-8, can be linearized to give the following model.

$$\begin{bmatrix} \dot{f}_1 \\ \dot{E}_1 \end{bmatrix} = \begin{bmatrix} \frac{-1}{\alpha A_1} & 0 \\ \frac{\beta \hat{t}_1}{A_1 \hat{h}_1} & \frac{-(\beta + \hat{h}_1)}{\alpha A_1 \hat{h}_1} \end{bmatrix} \begin{bmatrix} f_1 \\ E_1 \end{bmatrix} + \begin{bmatrix} \frac{1}{\alpha A_1} & \frac{1}{\alpha A_1} \\ \frac{t_h}{\alpha A_1} & \frac{t_c}{\alpha A_1} \end{bmatrix} \begin{bmatrix} f_h \\ f_c \end{bmatrix} \tag{7-12}$$

$$\begin{bmatrix} h_1 \\ t_1 \end{bmatrix} = \begin{bmatrix} \alpha & 0 \\ \dfrac{-\alpha\hat{t}_1}{\hat{h}_1} & \dfrac{1}{\hat{h}_1} \end{bmatrix} \begin{bmatrix} f_1 \\ E_1 \end{bmatrix} \tag{7-13}$$

The steady state values of h_1 and t_1 are given by $\hat{h}_1$ and $\hat{t}_1$, respectively.

A linearized model for tank 2 is given by,

$$\dot{t}_2 = \frac{-(\hat{h}_1 + \beta + \alpha f_b)}{\alpha A_2 h_2} t_2 + \begin{bmatrix} \dfrac{\hat{t}_1 + \hat{t}_2}{\alpha A_2 h_2} & \dfrac{\hat{h}_1 + \beta}{\alpha A_2 h_2} \end{bmatrix} \begin{bmatrix} h_1 \\ t_1 \end{bmatrix}. \tag{7-14}$$

As above, $\hat{t}_2$ denotes the steady state value of t_2, and can be calculated by,

$$\hat{t}_2 = \frac{\hat{f}_1 \hat{t}_1 + f_b t_b}{\hat{f}_1 + f_b}$$

where the steady state flow out of tank 1 is,

$$\hat{f}_1 = \alpha\hat{h}_1 - \beta\,.$$

To complete the linearized nominal model we must select a nominal operating point. The demo script, `tankdemo`, develops the equations in the form given above and uses the values $\hat{h}_1 = 0.75$ and $\hat{t}_1 = 0.75$ for the actual design.

Perturbation Model

We must now select a perturbation model structure for our robust control model. This involves determining the manner in which the perturbations enter the model, and selecting appropriate frequency dependent weights to normalize the perturbations. The guiding approach is to make the model perturbations match our estimate of uncertainty in the physical system as closely as possible.

For example, it is reasonable to attribute some dynamic uncertainty to the actuators, which suggests that uncertainty at the inputs, f_h and f_c, is appropriate. It is also reasonable to consider this uncertainty as being relative with respect to the input. A potential model may have uncertainty modeled by multiplicative perturbations on f_h and f_c. This is particularly appropriate for high condition number plants as it prevents the controller from inverting the plant from the input in a robust design.

We already know from the open-loop experiments that there is a significant amount of dynamic uncertainty in the t_1 response. This is due primarily to mixing and heat loss. This suggests that a multiplicative perturbation on the t_1 output is also an appropriate model.

There are many choices available and we have chosen to capture the uncertainty with multiplicative perturbations on h_1, t_1, and t_2. One could also add perturbations on f_h and f_c, although it is doubtful that the additional model complexity (it would now have five perturbations), would be worthwhile. Arguably, we do not have such a detailed model of the uncertainty in this case.

Figure 7-83 illustrates the perturbed linear model of the two tank system. The three perturbations, Δ_1, Δ_2, and Δ_3, model the uncertainty in h_1, t_1, and t_2, respectively. Note that they are multiplicative perturbations and that t_2 is influenced by all three perturbations. The nominal tank 1 model is described by equations 7-12 and 7-13, and the nominal tank 2 model is given by equation 7-14.

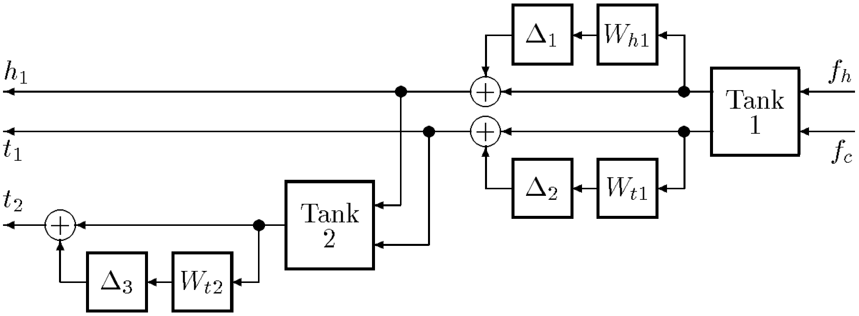


Figure 7-83: Schematic Representation of the Perturbed, Linear, Two Tank Model

To complete this model we must specify the perturbation weights, W_{h1}, W_{t1}, and W_{t2}. As we might expect, it is not possible to precisely determine the amount of uncertainty in the system. At best, we look for a rough frequency dependent bound. We discuss some of the ad-hoc means for getting suitable weights below:

- *Accuracy limits on actuators and sensors.* Equipment manufacturers will often specify accuracy limits on such equipment and this serves as a useful starting point for at least the steady-state uncertainty of some system components.

  Known rate limits can also be viewed as limiting the high frequency behavior of an actuator. Rate limits are nonlinear and it will not be possible to completely capture their effects with even a linear perturbation model. However, these limits may give an approximate estimate of the bandwidth of an actuator.

- *Linearization constants.* If a nonlinear model is available, then examining the variation in the linearization constants across the expected range of operation will give an estimate of appropriate levels of uncertainty for elements of the model. For example, we could examine how the time constant of the tank 1 t_1 response varied across the h_1 operating range.

  Again, this is only a crude approximation. The full nonlinear behavior of the plant is not captured by considering a series of frozen linear systems at various operating points.

- *Open-loop identification experiments.* Comparing experimentally estimated transfer functions to those predicted by a nominal model gives a good estimate of the model/system discrepancy at each frequency.

- *Closed-loop identification experiments.* These are preferred over open-loop identification experiments for a number of reasons:

  Closing the loop automatically biases the plant to the correct operating point. This provides appropriate signals for identification as they are close to those signals that will occur in final closed-loop operation;

  The achievable performance (at least in terms of closed-loop bandwidth) is limited by the uncertainty at frequencies close to the loop cross-over frequency. Open-loop experiments tend to emphasize the frequencies where the plant gain is high – typically steady-state in many process control systems. This is simply due to the significantly larger response on the plant at those frequencies. On the other hand, using closed-loop identification experiments places the emphasis on the loop cross-over frequencies, which is more important for the ultimate closed-loop design.

The analysis of closed-loop identification experiments needs to be performed carefully. The measurement noise is no longer uncorrelated with the input and output signals and its effect cannot be removed simply by averaging. Van den Hof *et al*. [VdHSB] give a suitable means of dealing with this situation.

- *Closed-loop relay experiments*. The “Closed-Loop Experiments” section briefly discussed using relays to induce closed-loop limit cycle operation. A large class of systems (including the two tank system) will limit cycle under these circumstances. The frequency at which the system limit cycles is simply the loop cross-over frequency and we can investigate the achievable closed-loop bandwidth by using lead controllers to make the system limit cycle at successively higher frequencies.

  Beyond some frequency the system no longer has a limit cycle; the dynamics are no longer dominated by linear time-invariant behavior. The heuristic is that at such frequencies the effect of the perturbations should dominate the model. This is easy to apply when a multiplicative perturbation is used; when the magnitude of the weight is greater than one, the perturbation dominates the model.

  This approach was used for determining the frequencies at which $|W_{t1}|$ and $|W_{t2}|$ exceeded one.

- *Systematic design, implementation, and evaluation*. In other words, try it and see. Clearly, closed-loop implementation of the controller design should be the final arbiter of a suitable model. One approach is to design a series of controllers, each with a smaller perturbation weight (i.e., assuming less uncertainty), and implement these controllers on the physical system. Setting the weight too high will give a conservative result as performance is unnecessarily traded off against robust stability. A weight that is too small will lead to a controller design that exhibits too little robustness, or is even unstable, in practice. Balas and Doyle [BalDoy1, BalDoy2] give details of a similar approach to a flexible structure problem.

At first it seems that this approach does not save any effort as we must design and implement all potential controllers. However, it does provide information about the appropriate level of uncertainty, and we may be able to conduct simpler experiments (for example designing SISO controllers to evaluate W_{h1}) to gain information about components within the system. This involves less effort than attempting to tune all perturbation weights at once.

These issues have been considered in developing the perturbation weights. The most important aspects of the two tank problem were considered to be:

- The nominal model for h_1 is very accurate up to at least 0.3 Hz.
- Limit cycle experiments in the t_1 loop suggest that uncertainty should dominate above 0.02 Hz.
- There is about 180 degrees of additional phase lag in the t_1 model (at $h_1 = 0.75$) at about 0.02 Hz. There is also a significant gain loss at this frequency. These effects are the result of the unmodeled mixing dynamics.
- Limit cycle experiments in the t_2 loop suggest that uncertainty should dominate above 0.03 Hz.

It is not surprising that there is less uncertainty associated with tank 2; most of the temperature uncertainty arises from the mixing dynamics and tank 2 is somewhat smaller than tank 1. The perturbation weights are illustrated graphically in Figure 7-84.

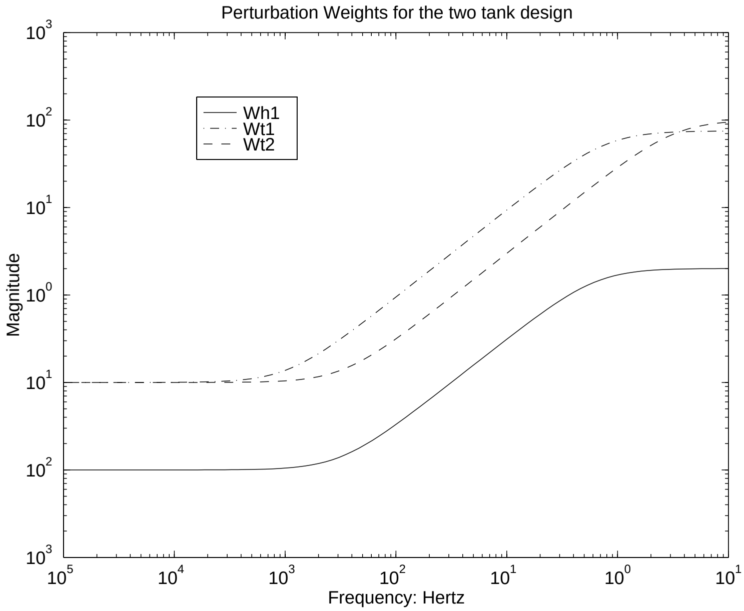


Figure 7-84: Perturbation Weights for the Robust Control Design Problem

The output uncertainties W_{h1} and W_{t1} are given by (note that here we express the t_1 perturbation weight as a function of the steady state height),

$$W_{h1} = 0.01 + \frac{0.5s}{0.25s + 1}$$

$$W_{t1} = 0.1 + \frac{20\hat{h}_1 s}{0.2s + 1}.$$

The t_2 perturbation weight is chosen as,

$$W_{t2} = 0.1 + \frac{100s}{s + 21}.$$

Sensor Noise Weights

The sensor dynamics are insignificant relative to the dynamics of the rest of the system. This will not be true of the sensor noise. The potential sources of noise include electronic noise in thermocouple compensators, amplifiers, and filters, radiated noise from the stirrers, and poor grounding (this is, after all, an inexpensive laboratory experiment).

We will model the noise by adding a weighted unknown input to h_1, t_1, and t_2. Smoothed FFT analysis has been used to estimate the noise on a quiescent system. This gives the following weights.

$W_{h1noise} = 0.01$

$W_{t1noise} = 0.03$

$W_{t2noise} = 0.03$

The design considered in `tankdemo.m` uses measurements of only t_1 and t_2. The weight $W_{h1noise}$ is unnecessary for this case but is included so that you can investigate different control configurations.

There are additional disturbances or noises associated with the measurements that are not, strictly speaking, sensor noise. For example, in tank 2 the imperfect mixing of the bias stream causes variations in the temperature measurement. The inclusion of such noises here does not affect the performance requirements of the controller.

Specifying the Design Requirements

It now remains to include in the interconnection structure (refer to Figure 7-82) the weighting functions that specify performance. These are the reference weights, error weights, and actuator weights. Arguably, noise weights and disturbance weights also fall into those categories although we have chosen to present the noise weights as part of the system model above. In a more general setting weighted disturbances or more complex error specifications might also be considered.

The weights should be considered as frequency dependent, relative weightings. For example, it is the relative size of the reference weights and noise weights that determines the extent to which the final controller design emphasizes reference tracking error over sensor noise rejection.

The error weights are first order low pass filters, with W_{t1perf} weighting the t_1 tracking error and W_{t2perf} weighting the t_2 tracking error. These are given by,

$$W_{t1perf} = \frac{100}{400s+1}$$

$$W_{t2perf} = \frac{50}{800s+1}.$$

We have selected a higher weight (better tracking performance) for t_1 because physical considerations lead us to believe that t_1 will be easier to control than t_2.

The majority of the water flowing into tank 2 comes from tank 1, which means that changes in t_2 are dominated by changes in t_1. It makes more sense to express the reference weighting in terms of t_1 and $t_2 - t_1$. This allows us to express the fact that t_2 is normally commanded to a value close to t_1. This is done by using the following weighting approach.

$$\begin{bmatrix} t_{1cmd} \\ t_{2cmd} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \end{bmatrix} \begin{bmatrix} W_{t1cmd} & 0 \\ 0 & W_{tdiffcmd} \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \end{bmatrix}.$$

In this case,

$$W_{t1cmd} = 0.1$$

$$W_{tdiffcmd} = 0.01$$

Figure 7-85 graphically illustrates the error and command weighting functions.

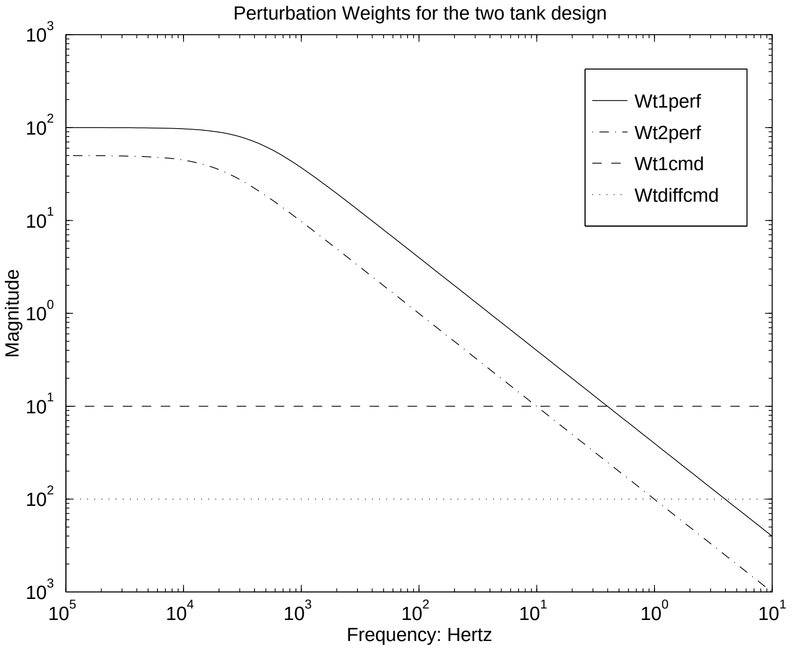


Figure 7-85: Performance Weights for the Robust Control Design Problem

In the case of the actuation weights we would like to weight both the amplitude and the rate of the actuator. This can be done by weighting f_{hc} (and f_{cc}) with a function that rolls up at high frequencies. An alternative approach can be used when we have a first order actuator model.

Using f_{hc} as an example; the approach is to create an actuator model with f_h and df_h/dt as outputs. These can then be separately weighted with constant weights. Note that this approach has the advantage of reducing the number of states in the interconnection structure, and hence in the final design. Figure 7-86 illustrates the form of such a weighted actuator model.

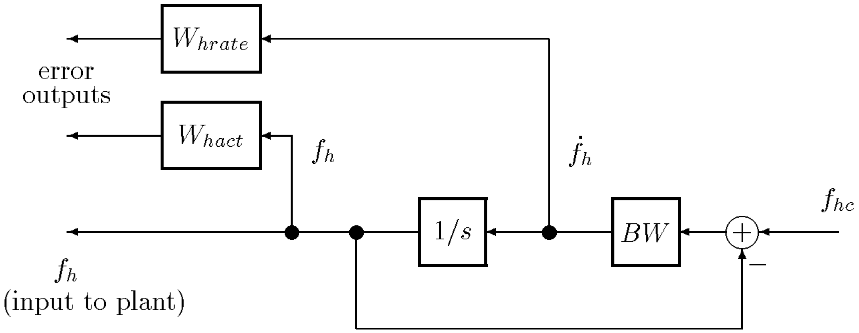


Figure 7-86: Model of the Flow Valve Actuator Including Magnitude and Rate Weightings

The actuator bandwidth, denoted by BW in Figure 7-86, is,

BW = 20 radians/sec.

The weights selected for the `tankdemo.m` design were,

$$\begin{aligned} W_{hact} &= 0.01 \\ W_{cact} &= 0.01 \\ W_{hrate} &= 50 \\ W_{crate} &= 50 \end{aligned}$$

Note that each weighted actuator model contributes only one state to the interconnection structure, and allows independent weighting of f_h and $\dot{f}_h$ (and f_c and $\dot{f}_c$).

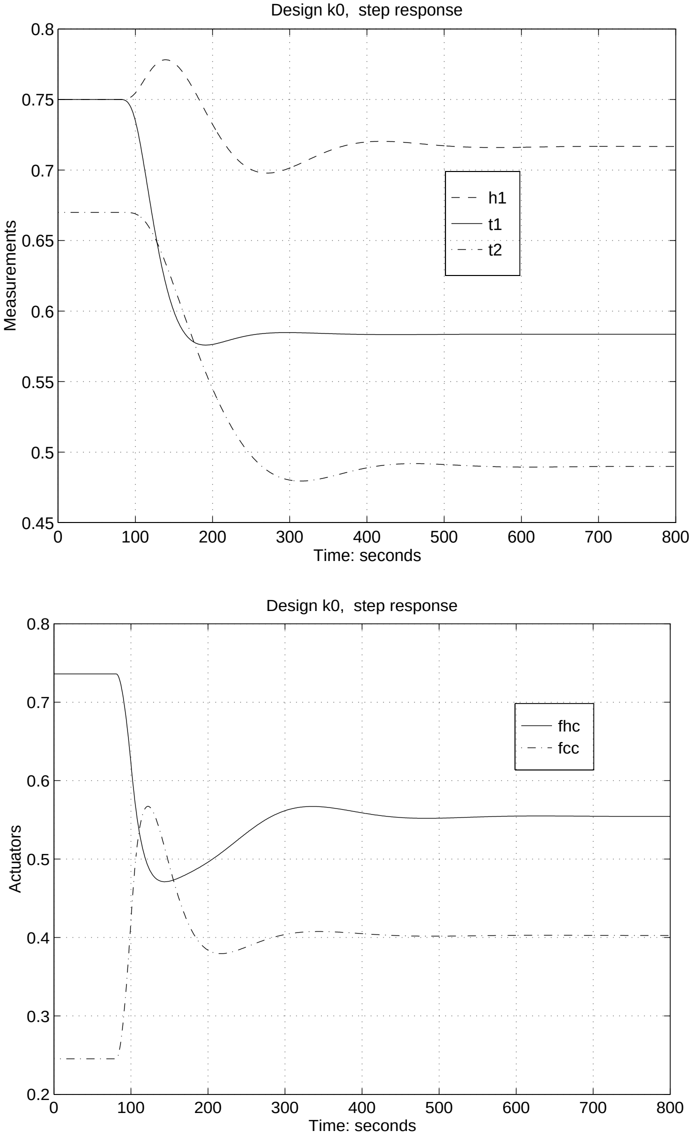
Controller Design and Analysis

The `tankdemo.m` script runs through the design for the problem outlined above. The script file is not discussed in detail as it roughly follows the layout of the above material.

Design Issues

One issue regarding the design approach is worth noting. We begin by designing for the nominal case. This is done by selecting out of the interconnection structure the perturbation inputs and outputs (v and z). This design achieved a closed-loop H_∞ norm of $\gamma = 0.9082$. This is a lower bound on the achievable value of μ for robust performance.

The response of the nominal controller (`k0` in the script) is simulated, and is shown in Figure 7-87. This is a check on the applicability of our performance weights. In general, the inclusion of the robustness in the design problem will have the effect of trading off this nominal performance in order to improve the stability robustness. Simulating the nominal design gives a rough idea of what the time domain performance implied by the weights.



The reference signal ramps (from 80 to 100 seconds) *t*1 from 0.75 to 0.57, and *t*2 from 0.67 to 0.47.

Figure 7-87: Time Response for the Nominal H• Controller

The demo file, `tankdemo`, uses this interconnection structure for the robust performance design. In the demo example three $D-K$ iterations are performed. In each of the D-scale fitting calls (performed with the function `musynfit`), second order D-scale fits were selected. The final value of μ achieved is 1.73. Further iterations may give further improvements; for this example it was decided to stop at this point.

Closed-Loop Analysis

We will run through a typical series of analyses for our design and briefly discuss the aspects of interest. The procedure guarantees closed-loop stability (although it is still worth checking to make sure numerical problems have not invalidated the design) but the controller need not be stable. It often is stable and this is preferable for implementation purposes.

The frequency response of the controller is given in Figure 7-88. We note that the controller rolls off around the frequency range where the uncertainties start to become larger. This is what we would expect from a classical point of view.

Note that there is not a great deal of low frequency gain. We will subsequently see that this is because the noise level is high relative to the error weighting.

An identical simulation is run to give an idea of the loss of nominal performance in the time domain. This is not intended as a comparison between k_0 and k_{mu}. A more appropriate comparison would also include a simulation of a perturbed system. The function `dypert` can be used to select the worst case perturbation. The nominal response of k_{mu} is shown in Figure 7-89. There is little deterioration in the nominal response for controller k_{mu}.

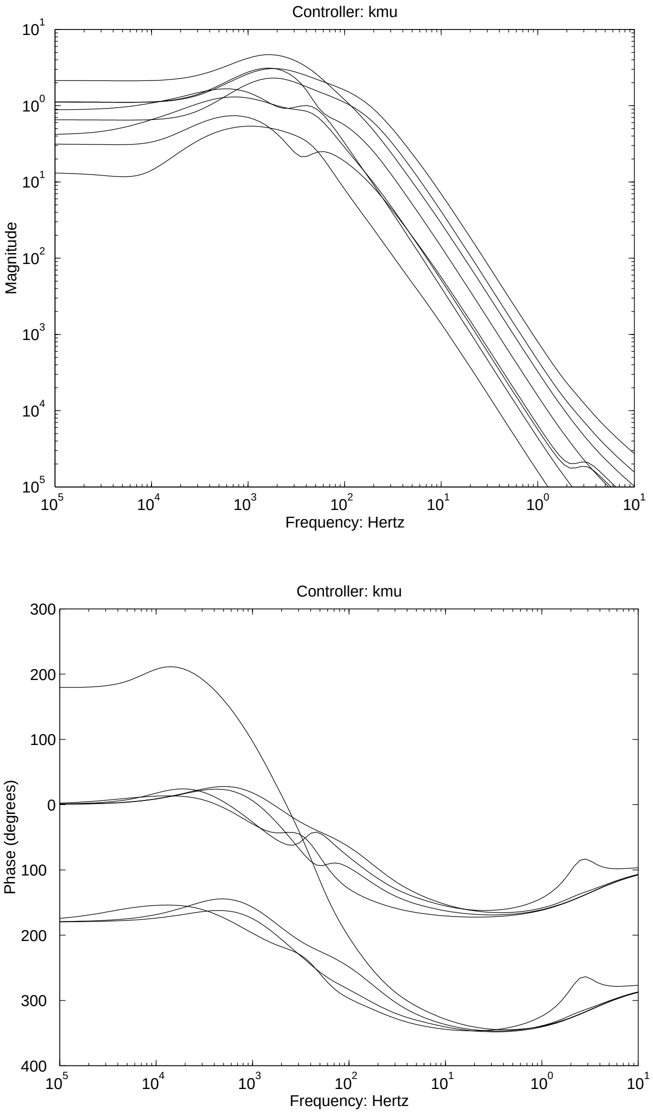
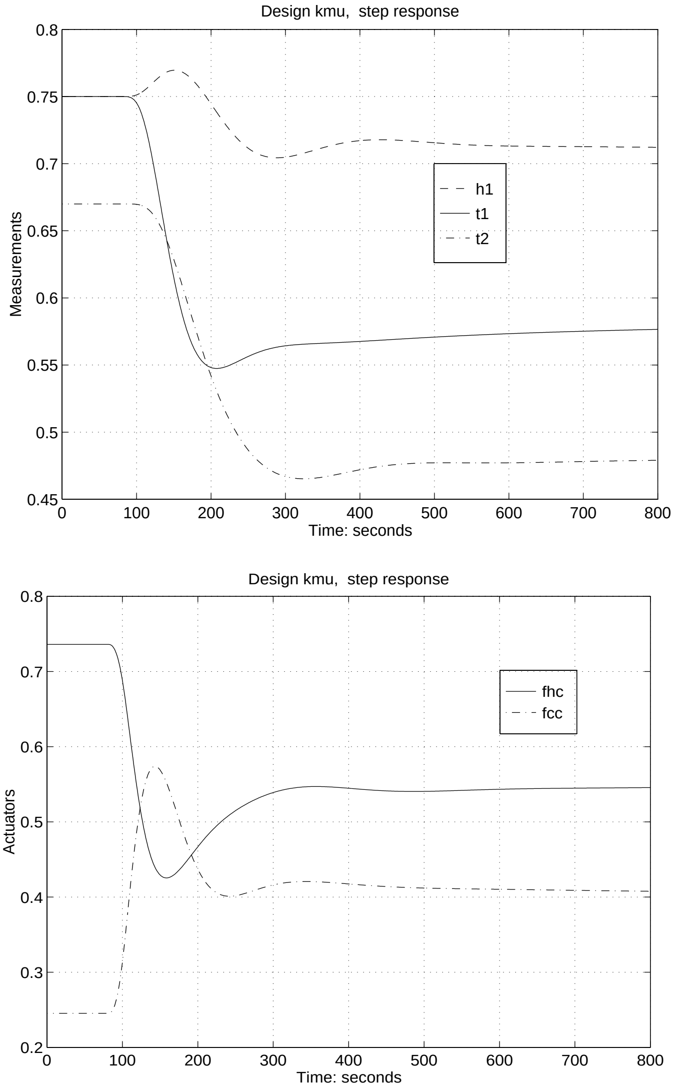


Figure 7-88: Frequency Response of the μ Synthesis Controller



The reference signal ramps (from 80 to 100 seconds) $t1$ from 0.75 to 0.57, and $t2$ from 0.67 to 0.47

Figure 7-89: Nominal Time Response for the k_{mu} Controller

The tradeoff between robustness and performance can be further studied by independently calculating μ for robust stability, and the closed-loop nominal performance. Note that robust stability is a μ test on the *v* to *z* block of the closed-loop transfer function in Figure 7-82. Three 1 × 1 blocks are specified for the robust stability block structure. The robust performance is given by the maximum singular value of the *w* to *e* block of the closed-loop transfer function. Figure 7-90 compares these with robust performance over the frequency range.

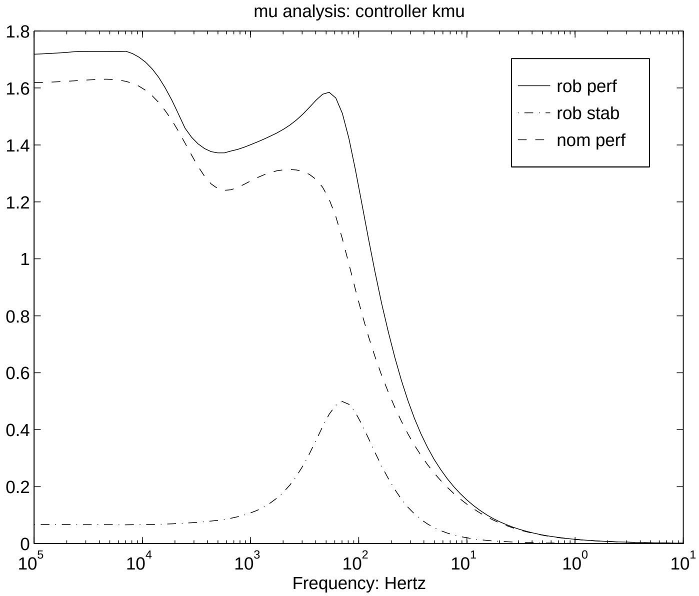


Figure 7-90: Robust Performance, Robust Stability, and Nominal Performance for the μ Synthesis Controller

Figure 7-90 illustrates that the nominal performance limits the low frequency robust performance. Robust stability does not become an issue except around 0.01 Hz. This is not surprising as the perturbation weights begin to increase around this frequency, yet the controller has only begun to roll off.

These issues can be further investigated by examining smaller subblocks of the *D*-scaled closed-loop system. This form of analysis has been referred to as *M*-analysis in the aerospace robust control community, simply because the closed-loop system was habitually designated by *M*. It allows us to determine which inputs and outputs are limiting the achievable robust performance. This analysis is performed by calculating the μ upper bound, DMD^{-1}, at each frequency. We then look at the maximum singular values of the appropriate subblocks. The example will serve as clarification.

The performance inputs, w, are divided into two groups:

- Reference commands
- Noise

There are three groups of performance outputs:

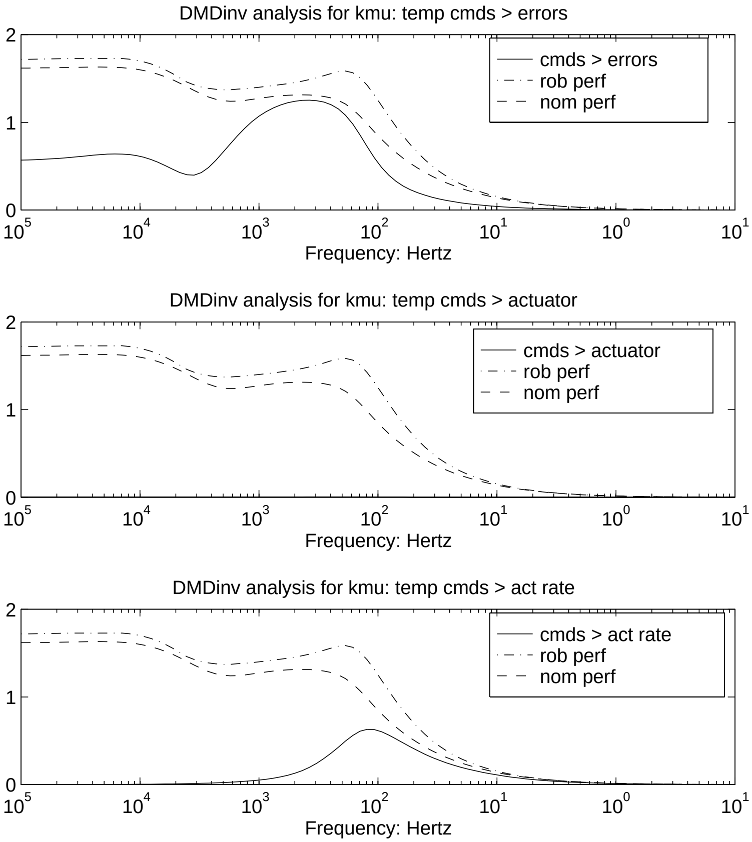
- Tracking errors
- Actuator penalties
- Actuator rate penalties

This give six 2 × 2 transfer function blocks, and for each of these we calculate the maximum singular value as a function of frequency.

The results for the temperature inputs are shown in Figure 7-91. The calculated nominal performance (maximum singular value of the w to e transfer function) is an upper bound for each of the subblock calculations and serves as a comparison point.

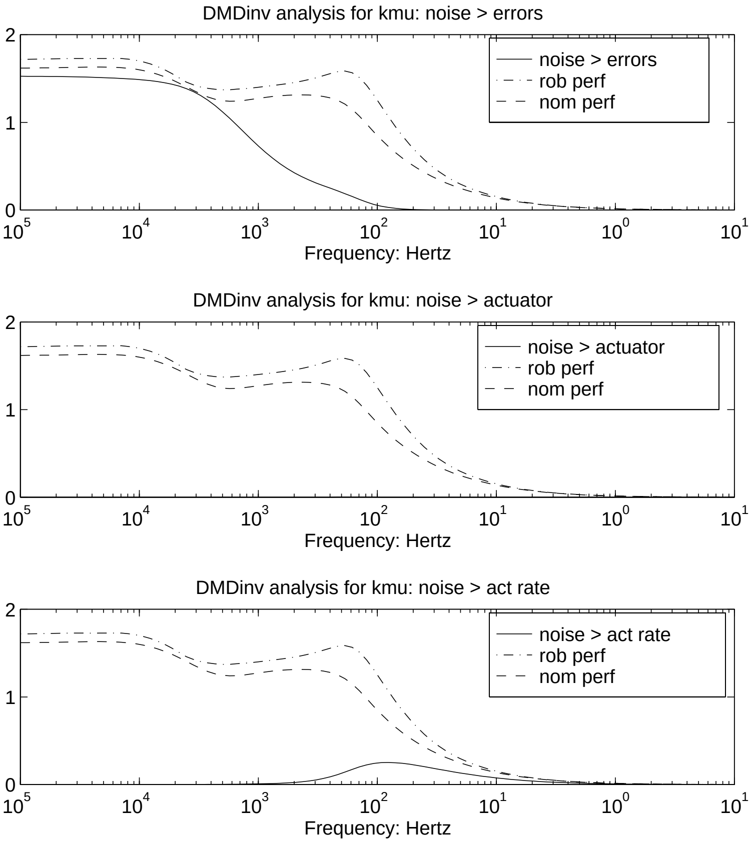
The temperature reference command to tracking error transfer function dominates those shown in Figure 7-91. However it is only really significant, with respect to the nominal performance, in the frequency range 0.001 to 0.01 Hz. It also has some contribution at lower frequencies. By contrast, the actuator penalty has almost no effect on the design. We could change this weight significantly without affecting the design. The actuator rate penalty starts to influence the design at frequencies above 0.01 Hz.

Figure 7-92 illustrates the subblocks corresponding to the noise inputs. It is immediately clear that the noise to tracking error transfer function dominates the design at low frequencies. To improve the low frequency nominal performance (and the low frequency robust performance) we must reduce the noise weights. In practical terms this means buying higher quality sensors.



*robust performance and nominal performance are included for comparison

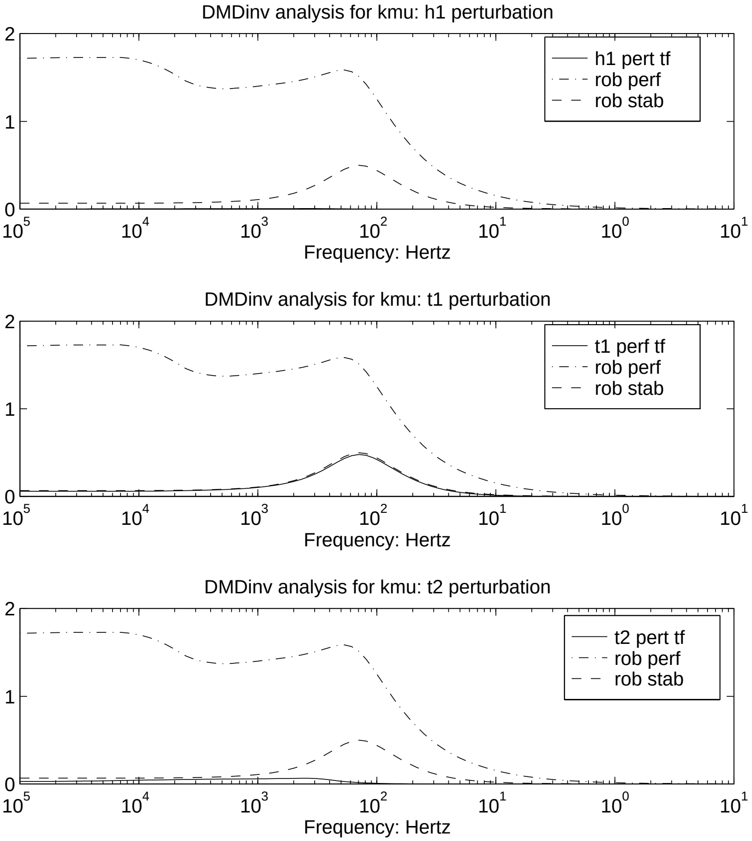
Figure 7-91: *DMD*$^{-1}$ Analysis for Temperature Reference Command Inputs



*robust performance and nominal performance are included for comparison

Figure 7-92: DMD^{-1} Analysis for Noise Inputs

A similar analysis can be used to determine which aspects of the design are limiting robust stability. In this case, there are three transfer functions to examine; v_1 to z_1, etc. These are illustrated in Figure 7-93 and compared to both robust performance and robust stability.



*robust performance and nominal performance are included for comparison

Figure 7-93: *DMD*$^{-1}$ Analysis for the Perturbations

As we would expect, the h_1 perturbation has little influence on the design. Recall that the uncertainty associated with the h_1 output is small compared to the temperature outputs. We can also see that the t_1 perturbation weight dominates the robust stability. This is still not large compared with robust performance, but if we wish to improve the robust stability, we must reduce W_{t1}. In practical terms this means that we must invest more effort in modeling and identifying some of the tank 1 mixing dynamics.

We can also apply this type of analysis at a single frequency. To do this we would simply examine the mu upper bound for the closed-loop system at a particular frequency. A column with large numbers would indicate that the corresponding input was dominating the design. Similarly, a row of large numbers would indicate that the corresponding output was dominating the design.

This form of analysis is very useful in giving engineering information about the limiting factors in the design. As we have illustrated above, it can be used to indicate where additional effort (in terms of further modeling/identification, or higher quality sensing) is required to improve robust performance.

Experimental Evaluation

We present an experimental comparison between a μ synthesis design and a more standard loopshaping controller on the two tank system. The design shown here is not identical to that given above as it was calculated and implemented several years before μ-Tools was written. However, the method and relative weightings were similar.

A simple SISO style loopshaping technique has been used to design controllers (K_{loop}) for comparison purposes. One of the worst case plant conditions, (h_1 = 0.75, t_1 = 0.25), was selected as a nominal design point. The controller simply inverts the plant to a diagonal loopshape. It is now well known that this technique will not work well for high condition number plants, particularly those with uncertainty at the input. Refer to the work of Skogestad *et al.* [SkoMD] for further details on this point.

The loopshape chosen was

$$t_1 \text{ loop } = \frac{100}{(1 + 3000s)}$$

Figure 7-94 shows the magnitude of the frequency response of K_{loop}.

$$t_2 \text{ loop } = \frac{100}{(1 + 5000s)}$$

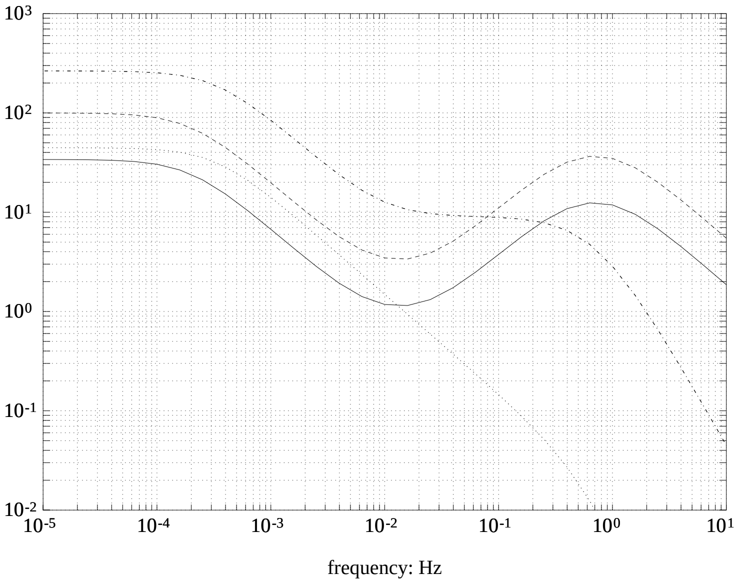


Figure 7-94: Magnitude Response of the Loopshaping Controller: K_{loop}

Both controllers have been tested over a wide range of commands. Figure 7-95 shows a typical command tracking response for the K_{loop} and Figure 7-96 shows the same response for K_{mu}. The oscillatory behavior of the loopshaping controller cannot be alleviated by selecting loopshapes that have a lower crossover frequency. These merely produce lower frequency and higher amplitude oscillations. It is the method of inverting the plant that leads to this problem.

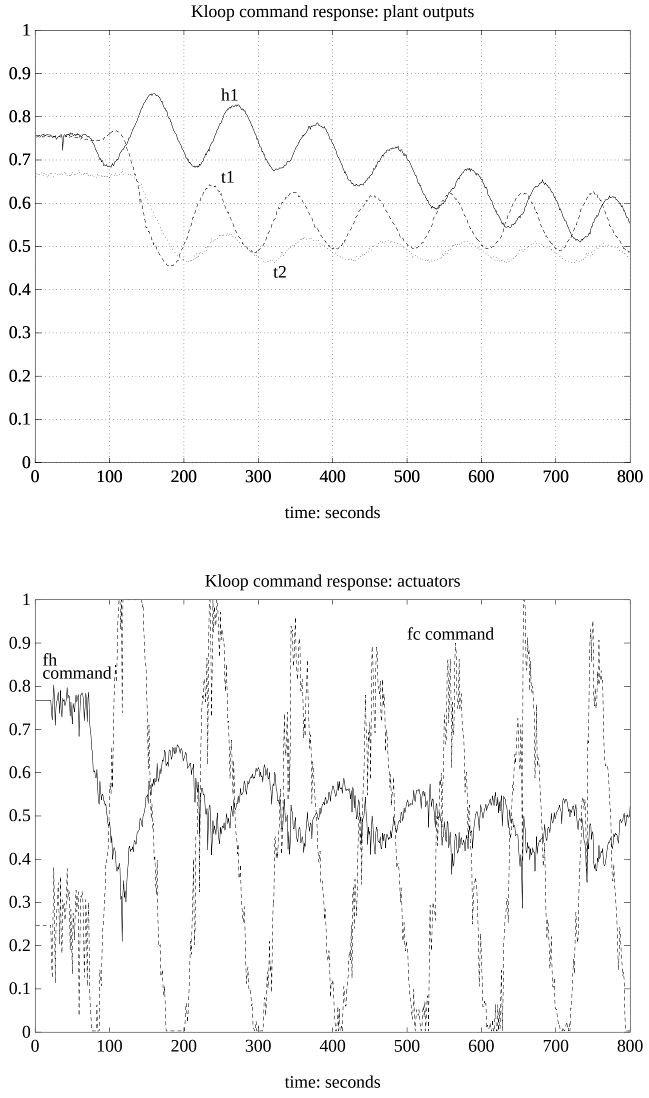
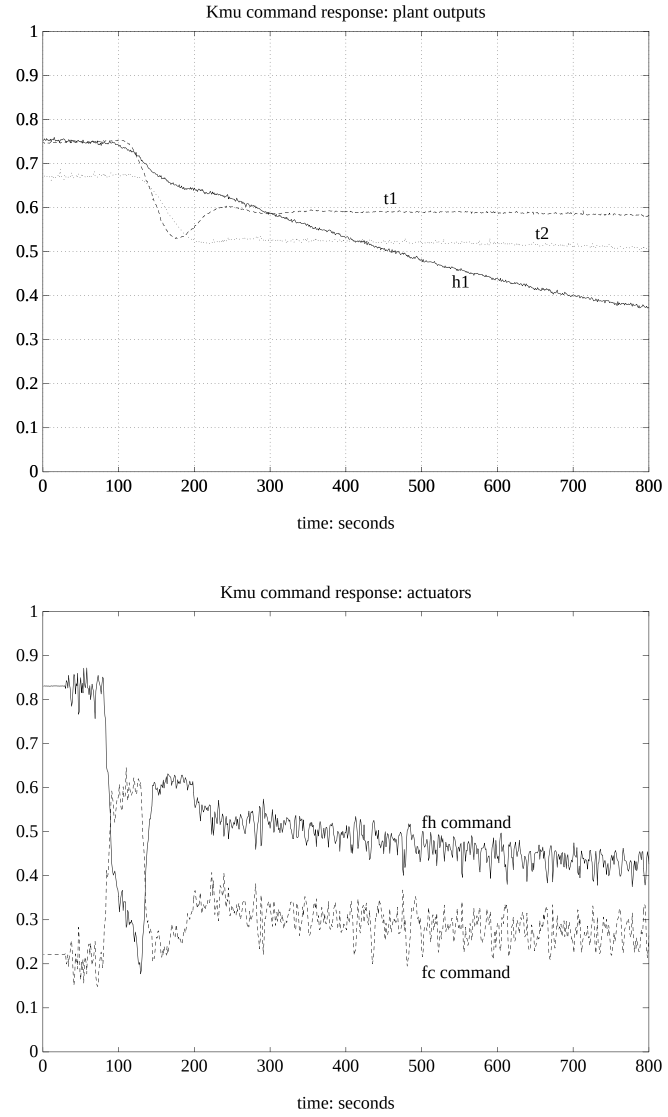


Figure 7-95: Experimental Command Response: Controller K_{loop} with Input Temperature Ramps Going from t_1 = 0.75, t_2 = 0.67 at 80 Seconds to t_1 = 0.55, t_2 = 0.47 at 100 Seconds



Kmu command response: plant outputs

Kmu command response: actuators

Figure 7-96: Experimental Command Response: Controller K_{mu} with Input Temperature Ramps Going from t_1 = 0.75, t_2 = 0.67 at 80 Seconds to t_1 = 0.55, t_2 = 0.47 at 100 Seconds

Two Tank System References

[SmDMS:] Smith, R.S., J. Doyle, M. Morari, and A. Skjellum, “A case study using μ: Laboratory process control problem,” in *Proc. Int. Fed. Auto. Control*, vol. 8, pp. 403–415, 1987.

[SmD1:] Smith, R.S, and J. Doyle, “The two tank experiment: A benchmark control problem,” in *Proc. Amer. Control Conf.*, vol. 3, pp. 403–415, 1988.

[Smith:] Smith, R.S., “Model validation for robust control: an experimental process control application,” *Automatica,* Nov. 1995.

[AstH:] Åström, K.J., and T. Hägglund, “Automatic tuning of simple regulators with specifications on phase and amplitude margins,” *Automatica,* vol. 20, no. 5, pp. 645–651, 1984.

[SmD2:] Smith, R.S., and J. C. Doyle, “Closed loop relay estimation of uncertainty bounds for robust control models,” in *Proc. of the 12th IFAC World Congress*, vol. 9, pp. 57–60, July 1993.

[VdHSB:] Van den Hof, P.M., R. J. Schrama, and O. H. Bosgra, “An indirect method for transfer function estimation from closed loop data,” in *Proc. IEEE Control Decision Conf.*, pp. 1702–1706, 1992.

[BalD1:] Balas, G.J., and J. C. Doyle, “Identification of flexible structures for robust control,” *IEEE Control Sys. Magazine,* vol. 10, pp. 51–58, June 1990.

[BalD2:] Balas, G.J., and J. Doyle, “Robustness and performance trade-offs in control design for flexible structures,” *IEEE Trans. Control Syst. Tech.*,” vol. 2, no. 4, pp. 352–361, 1994.

[SkoMD:] Skogestad, S., M. Morari, and J. C. Doyle, “Robust control of ill-conditioned plants: High-purity distillation,” *IEEE Trans. Auto. Control*, vol. 33, pp. 1092–1105, December 1988.

8

Reference

This chapter contains a detailed description of all μ-Analysis and Synthesis Toolbox (μ-Tools) functions. It begins with a list of functions in alphabetical order, followed by a list of functions grouped by subject area, and continues with a detailed description of each command. Information on each function is also available through the MATLAB on-line help facility.

In the summary of commands index and tables, the following abbreviations are used:

- CONSTANT matrices are denoted by **C**
- SYSTEM matrices are denoted by **S**
- VARYING matrices are denoted by **V**

Therefore, if a command can be used with either a CONSTANT, SYSTEM or VARYING matrix, it may be denoted by **CSV**.

Summary of Commands

Command	Description
`abv`	Stack **CSV** matrices above one another
`blknorm`	Block norm of a **CV** matrix
`cf2sys`	Create a **S** from a normalized coprime factorization
`cjt`	Conjugate transpose of a **CSV** matrix
`cmmusyn`	Constant matrix μ synthesis
`cos_tr`	Generate a cosine signal as a **V** matrix
`crand`	Generate a complex random **C** matrix (uniform distribution)
`crandn`	Generate a complex random **C** matrix (normal distribution)
`csord`	Ordered complex Schur form
`daug`	Diagonal augmentation of **CSV** matrices
`dhfnorm`	Calculate discrete-time ∞-norm of a stable **S** matrix
`dhfsyn`	Discrete-time H_∞ control design
`dkit`	Automated $D - K$ iteration for μ synthesis
`dkitgui`	Automated $D - K$ iteration GUI for μ synthesis
`drawmag`	Interactive moused-based sketch and fitting tool
`dtrsp`	Discrete-time response of a linear **S** matrix
`dypert`	Create a rational perturbation from frequency `mu` output data
`emargin`	Calculate normalized coprime factor robust stability margin
`fitmag`	Fit magnitude data with a real, rational, transfer function

Command	Description
`fitsys`	Fit frequency response data with a transfer function
`frsp`	Frequency response of a **S** matrix
`gap`	Calculate the gap metric for **S** matrices
`genmu`	Generalized μ-analysis of **CV** matrices
`genphase`	Generate a minimum phase function from magnitude data
`getiv`	Get the independent variable of a **V** matrix
`h2norm`	Calculate 2-norm of a stable, strictly proper **S** matrix
`h2syn`	H_2 control design
`hankmr`	Optimal Hankel norm approximation of a **S** matrix
`hinffi`	H_∞ full information control design
`hinfnorm`	Calculate ∞-norm of a stable, proper **S** matrix
`hinfsyn`	H_∞ control design
`hinfsyne`	H_∞ control design, minimum entropy
`indvcmp`	Compare independent variable data of two **V** matrices
`madd`	Addition of **CSV** matrices
`magfit`	Fit frequency response data with a transfer function (batch)
`massign`	Assign a portion of a matrix
`mfilter`	Construct a Bessel, Butterworth, Chebychev, or RC filter
`minfo`	Information on a **CSV** matrix
`minv`	Inverse of **CSV** matrices
`mmult`	Multiplication of **CSV** matrices

Command	Description
`mprintf`	Formatted printing of a matrix
`mscl`	Scale (by a scalar) a **S** or **V** matrix
`msf`	An interactive *D*-scaling rational fit routine
`msub`	Subtraction of **CSV** matrices
`mu`	μ-analysis of **CV** matrices
`muftbtch`	A batch *D*-scaling rational fit routine
`musynfit`	An interactive *D*-scaling rational fit routine
`musynflp`	An interactive *D*-scaling rational fit routine (linear programming)
`muunwrap`	Construct *D*-scaling and perturbation from `mu`
`ncfsyn`	H_∞ loopshaping control design
`nd2sys`	Convert a SISO transfer function into a **S** matrix
`negangle`	Calculate angle of **CV** matrices elements between 0 and –2π
`nugap`	Calculate the ν (nu) gap for **S** matrices
`pck`	Create a **S** matrix from state-space data (`A,B,C,D`)
`pkvnorm`	Peak norm of a **V** matrix
`pss2sys`	Convert an `[A B;C D]` matrix into a μ-Tools **S** matrix
`randel`	Generate a random perturbation for `mu` block structure
`reordsys`	Reorder states of a **S** matrix
`ric_eig`	Solve a Riccati equation via eigenvalue decomposition
`ric_schr`	Solve a Riccati equation via real Schur decomposition
`rifd`	Display real, imaginary, frequency, and damping data

Command	Description
`samhld`	Sample-hold approximation of a continuous **S** matrix
`sbs`	Stack **CSV** matrices next to one another
`sclin`	Scale **S** matrix input
`scliv`	Scale the independent variable of **V** matrix
`sclout`	Scale **S** matrix output
`sdecomp`	Decompose a **S** matrix into two **S** matrices
`sdhfnorm`	Calculate sample-data ∞-norm of a stable **S** matrix
`sdhfsyn`	Sample-data H_∞ control design
`sdtrsp`	Time response of a sample-data **S** matrix
`see`	Display **SV** matrices
`seeiv`	Print independent variables of a **V** matrix
`seesys`	Formatted **SV** display
`sel`	Select rows/columns or outputs/inputs of **CSV** matrices
`sfrwtbal`	Weighted balanced realization of a **S** matrix
`sfrwtbld`	Stable weighted balanced realization of a **S** matrix
`siggen`	Generate a signal as a **V** matrix
`simgui`	A GUI for time simulations of LFTs
`sin_tr`	Generate a sine signal as a **V** matrix
`sisorat`	Fit a frequency point with a first order, all-pass, stable function
`sncfbal`	Balanced realization of coprime factors of a **S** matrix
`sortiv`	Sort independent variable of a **V** matrix
`spoles`	Poles of a **S** matrix

Command	Description
`srelbal`	Stochastic balanced realization of a **S** matrix
`sresid`	Residualize states of a **S** matrix
`starp`	Redheffer star product
`statecc`	Apply a coordinate transformation to **S** matrices
`step_tr`	Generate a step signal as a **V** matrix
`strans`	Bidiagonal coordinate transformation of **S** matrices
`strunc`	Truncate states of a **S** matrix
`sys2pss`	Extract the state-space matrix `[A B; C D]` from a **S** matrix
`sysbal`	Balanced realization of a **S** matrix
`sysic`	System interconnection program
`sysrand`	Generate a random **S** matrix
`szeros`	Transmission zeros of a **S** matrix
`tackon`	String together **V** matrices
`transp`	Transpose of **SV** matrices
`trsp`	Time response of a linear **S** matrix
`tustin`	Prewarped continuous-time to discrete-time **S** transformation
`unum`	Input or column dimension of **CSV** matrix
`unpck`	Extract state-space data (`A,B,C,D`) from a **S** matrix
`unwrapd`	Construct *D*-scaling from `mu`
`unwrapp`	Construct Δ perturbation from `mu`
`vabs`	Absolute value of a **CV** matrix

Command	Description
var2con	Convert a **V** matrix to a **C** matrix
varyrand	Generate a random **V** matrix
vceil	Round elements of **CV** matrices towards ∞
vcjt	Conjugate transpace of **CV** matrices
vdcmate	Decimate a **V** matrix
vdet	Determinant of **CV** matrices
vdiag	Diagonal of **CV** matrices
vebe	Element-by-element operations on **V** matrices
veig	Eigenvalue decomposition of **CV** matrices
veval	Evaluate general functions of **V** matrices
vexpm	Exponential of **CV** matrices
vfind	Find individual elements of a **V** matrix
vfft	FFT for **V** matrices
vfloor	Round elements of **CV** matrices towards –∞
vifft	Inverse FFT for **V** matrices
vimag	Imaginary part of a **CV** matrix
vinterp	Interpolate **V** matrices
vinv	Inverse of a **CV** matrix
vldiv	Left division of **CV** matrices
vnorm	Norm of **CV** matrices
vpck	Pack a **V** matrix
vpinv	Pseudoinverse of a **CV** matrix
vplot	Plot **CV** matrices

Command	Description
vpoly	Characteristic polynomial of **CV** matrices
vrcond	Condition number of a **CV** matrix
vrdiv	Right division of **CV** matrices
vreal	Real part of a **CV** matrix
vrho	Spectral radius of a **CV** matrix
vroots	Polynomial roots of **CV** matrices
vschur	Schur form of a **CV** matrix
vspect	Signal processing spectrum command for **V** matrices
vsvd	Singular value decomposition of a **CV** matrix
vtp	Transpose of **CV** matrices
vunpck	Unpack a **V** matrix
vzoom	Mouse-driven axis selection of plot window
wcperf	Worst-case performance for given uncertainty level
wsgui	A MATLAB workspace GUI
xnum	State dimension of a **S** matrix
xtract	Extract portions of a **V** matrix using independent variables
xtracti	Extract portions of a **V** matrix
ynum	Output or row dimension of a **CSV** matrix
zp2sys	Convert transfer function poles and zeros into a **S** matrix

Commands Grouped by Function

Standard Operations/Basic Functions	
abv	Stack **CSV** matrices above one another
cjt	Conjugate transpose of **SV** matrices
daug	Diagonal augmentation of **CSV** matrices
madd	Addition of **CSV** matrices
minv	Inverse of **CSV** matrices
mmult	Multiplication of **CSV** matrices
mscl	Scale (by a scalar) a **SV** matrix
msub	Subtraction of **CSV** matrices
sbs	Stack **CSV** matrices next to one another
sclin	Scale **S** matrix input
sclout	Scale **S** matrix output
sel	Select **CSV** matrix rows/columns or outputs/inputs
starp	Redheffer star product
transp	Transpose of **SV** matrices

Matrix Information, Display and Plotting	
drawmag	Interactive mouse-based sketch and fitting tool
minfo	Information on a matrix
mprintf	Formatted printing of a matrix
rifd	Display real, imaginary, frequency, and damping data
see	Display **SV** matrices
seeiv	Display independent variables of a **V** matrix
seesys	Formatted **SV** display
unum	Input or column dimension of a **CSV** matrix
vplot	Plotting **CV** matrices
vzoom	Mouse-driven axis selection of plot window
wsgui	A MATLAB workspace GUI
xnum	State dimension of a **S** matrix
ynum	Output or row dimension of a **CSV** matrix

Modeling Functions	
`mfilter`	Construct a Bessel, Butterworth, Chebychev, or RC filter
`nd2sys`	Convert a SISO transfer function into a μ-Tools **S** matrix
`pck`	Create a **S** matrix from state-space data (`A, B, C, D`)
`pss2sys`	Convert an `[A B;C D]` matrix into a μ-Tools **S** matrix
`sys2pss`	Extract state-space matrix [A B; C D] from a **S** matrix
`sysic`	System interconnection program
`unpck`	Extract state-space data (`A,B,C,D`) from a **S** matrix
`zp2sys`	Convert transfer function poles and zeros to a **S** matrix

SYSTEM Matrix Functions	
`reordsys`	Reorder states in a **S** matrix
`samhld`	Sample-hold approximation of a continuous **S** matrix
`spoles`	Poles of a **S** matrix
`statecc`	Apply a coordinate transformation to **S** matrices
`strans`	Bidiagonal coordinate transformation of **S** matrices
`sysrand`	Generate a random **S** matrix
`szeros`	Transmission zeros of a **S** matrix
`tustin`	Prewarped continuous to discrete **S** transformation

Model Reduction Functions	
cf2sys	Create a **S** from a normalized coprime factorization
hankmr	Optimal Hankel norm approximation of a **S** matrix
sdecomp	Decompose a **S** matrix into two **S** matrices
sfrwtbal	Frequency weighted balanced realization of a **S** matrix
sfrwtbld	Stable frequency weighted realization of a **S** matrix
sncfbal	Balanced realization of coprime factors of a **S** matrix
srelbal	Stochastic balanced realization of a **S** matrix
sresid	Residualize states of a **S** matrix
strunc	Truncate states of a **S** matrix
sysbal	Balanced realization of a **S** matrix

SYSTEM Response Functions	
cos_tr	Generate a cosine signal as a **V** matrix
dtrsp	Discrete-time response of a linear **S** matrix
frsp	Frequency response of a **S** matrix
sdtrsp	Sample data time response of a linear **S** matrix
siggen	Generate a signal as a **V** matrix
simgui	A GUI for time simulations of LFTs
sin_tr	Generate a sine signal as a **V** matrix
step_tr	Generate a step signal as a **V** matrix
trsp	Time response of a linear **S** matrix

H2 and H_∞ Analysis and Synthesis Functions	
`dhfnorm`	Calculate discrete-time ∞-norm of a stable **S** matrix
`dhfsyn`	Discrete-time H_∞ control design
`emargin`	Normalized coprime factor robust stability margin
`gap`	Calculate the gap metric between **S** matrices
`h2norm`	Calculate 2-norm of a stable, strictly proper **S** matrix
`h2syn`	H_2 control design
`hinffi`	H_∞ full information control design
`hinfnorm`	Calculate ∞-norm of a stable, proper **S** matrix
`hinfsyn`	H_∞ control design
`hinfsyne`	H_∞ minimum entropy control design
`ncfsyn`	H_∞ loopshaping control design
`nugap`	Calculate the ν (nu) gap between **S** matrices
`pkvnorm`	Peak norm of a **V** matrix
`sdhfnorm`	Sample-data ∞-norm of a stable **S** matrix
`sdhfsyn`	Sample-data H_∞ control design

Structured Singular Value (μ) Analysis and Synthesis	
`blknorm`	Block norm of **CV** matrices
`cmmusyn`	Constant matrix μ synthesis
`dkit`	Automated *D* – *K* iteration for μ synthesis
`dkitgui`	Automated *D* – *K* iteration GUI for μ synthesis
`dypert`	Create a rational perturbation from frequency `mu` data
`fitmag`	Fit magnitude data with real, rational, transfer function
`fitmaglp`	Fit magnitude data with real, rational, transfer function
`fitsys`	Fit frequency response data with transfer function
`genphase`	Generate a minimum phase frequency response to magnitude data
`genmu`	Real and complex generalized μ-analysis of **CV** matrices
`magfit`	Fit magnitude data with real, rational, transfer function (a batch process)
`mu`	Real and complex μ-analysis of **CV** matrices
`msf`	Interactive *D*-scaling rational fit routine
`muftbtch`	Batch *D*-scaling rational fit routine
`musynfit`	Interactive *D*-scaling rational fit routine
`musynflp`	Interactive *D*-scaling rational fit routine (linear program)
`muunwrap`	Construct *D*-scaling and Δ perturbation from `mu`
`randel`	Generate a random perturbation
`sisorat`	Fit a frequency point with first order, all-pass, stable transfer function
`unwrapd`	Construct *D*-scaling from `mu`

Structured Singular Value (μ) Analysis and Synthesis	
`unwrapp`	Construct Δ perturbation from `mu`
`wcperf`	Worst-case performance for a given Δ

VARYING Matrix Manipulation	
`getiv`	Get the independent variable of a **V** matrix
`indvcmp`	Compare the independent variable data of two **V** matrices
`scliv`	Scale the independent variable of a **V** matrix
`sortiv`	Sort the independent variable of a **V** matrix
`tackon`	String together **V** matrices
`var2con`	Convert a **V** matrix to a **C** matrix
`varyrand`	Generate a random **V** matrix
`vfind`	Find individual elements of a **V** matrix
`vpck`	Pack a **V** matrix
`vunpck`	Unpack a **V** matrix
`xtract`	Extract portions of a **V** matrix
`xtracti`	Extract portions of a **V** matrix using independent variable

Standard MATLAB Commands for VARYING Matrices	
`vabs`	Absolute value of a **CV** matrix
`vceil`	Round elements of **CV** matrices towards ∞
`vdet`	Determinant of **CV** matrices
`vdiag`	Diagonal of **CV** matrices
`veig`	Eigenvalue decomposition of **CV** matrices
`vexpm`	Exponential of **CV** matrices
`vfft`	FFT for **V** matrices
`vfloor`	Round elements of **CV** matrices towards $-\infty$
`vifft`	Inverse FFT for **V** matrices
`vimag`	Imaginary part of a **CV** matrix
`vinv`	Inverse of a **CV** matrix
`vnorm`	Norm of **CV** matrices
`vpinv`	Pseudoinverse of a **CV** matrix
`vpoly`	Characteristic polynomial of **CV** matrices
`vrcond`	Condition number of a **CV** matrix
`vreal`	Real part of a **CV** matrix
`vroots`	Polynomial roots of **CV** matrices
`vschur`	Schur form of a **CV** matrix
`vspect`	Signal processing `spectrum` command for **V** matrices
`vsvd`	Singular value decomposition of a **CV** matrix

Additional VARYING Matrix Functions	
vcjt	Conjugate transpose of **CV** matrices
vdcmate	Decimate **V** matrices
vebe	Element-by-element operations on **V** matrices
veval	Evaluate general functions of **V** matrices
vinterp	Interpolate **V** matrices
vldiv	Left division of **CV** matrices
vrdiv	Right division of **CV** matrices
vrho	Spectral radius of a **CV** matrix
vtp	Transpose of **CV** matrices

Utilities and Miscellaneous Functions	
crand	Complex random matrix generator (uniform distribution)
crandn	Complex random matrix generator (normal distribution)
csord	Order complex Schur form matrices
massign	Assign a portion of a matrix to another
negangle	Calculate angle of matrix elements between 0 and -2π
ric_eig	Solve a Riccati equation via eigenvalue decomposition
ric_schr	Solve a Riccati equation via real Schur decomposition

Purpose

Augment CONSTANT, SYSTEM and VARYING matrices

Syntax

```
out = abv(mat1,mat2,...,matN)
out = daug(mat1,mat2,...,matN)
out = sbs(mat1,mat2,...,matN)
```

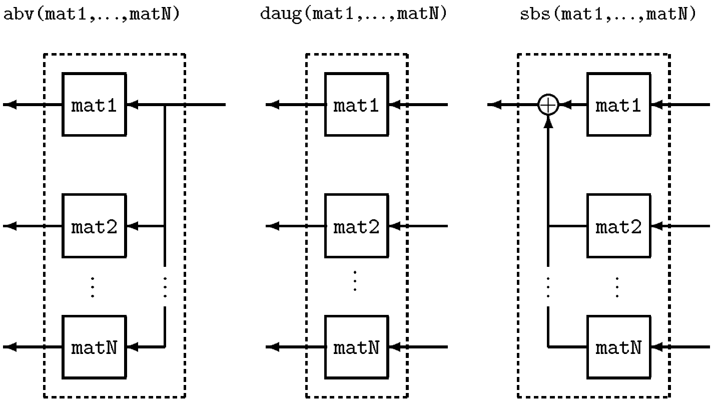
Description

abv places the matrix mat1 above the matrix mat2. daug places the input matrices on the diagonal of the output matrix. sbs places the input matrices next to one another. All these commands, abv, daug, and sbs, allow the use of multiple input arguments inputs (up to nine). CONSTANT, SYSTEM, and VARYING matrices can be placed by one another based on the following table.

		mat2		
		CONSTANT	**SYSTEM**	**VARYING**
mat1	**CONSTANT**	yes	yes	yes
	SYSTEM	yes	yes	no
	VARYING	yes	no	yes

The input matrices must be compatible in the respective dimension in order for the function to be performed. abv requires the same number of columns (inputs for SYSTEM matrices) and sbs requires the input matrices to have the same number of rows (outputs for SYSTEM matrices).

Pictorial Representation of Functions



Examples

Create two CONSTANT matrices a and b along with two SYSTEM matrices p1 and p2. Examples of manipulation using abv, daug, and sbs are shown in the following examples.

```
a = [1 2 3; 4 5 6];
b = [7 7 7; 8 8 8 ];
pl = pck(-10,1,10,0);
p2 = pck(-3,2,4,.1),
seesys(abv(p1,p2))

-1.0e+01    0.0e+00   |  1.0e+00
 0.0e+00   -3.0e+00   |  2.0e+00
---------------------------------
 0.0e+01    0.0e+00   |  0.0e+00
 0.0e+01    4.0e+00   |  1.0e-00

out = abv(a,b,b)
out =

   1     2     3
   4     5     6
   7     7     7
   8     8     8
   7     7     7
   8     8     8

out = daug(a,b)
out =

   1     2     3     0     0     0
   4     5     6     0     0     0
   0     0     0     7     7     7
   0     0     0     8     8     8

out = sbs(p1,p2);
seesys = out

-1.0e+01    0.0e+00   |  1.0e+00    0.0e+00
 0.0e+00   -3.0e+00   |  0.0e+00    2.0e+00
--------------------------------------------
 1.0e+01    4.0e+00   |  0.0e+00    1.0e-01
 minfo(out)
system:2states1 outputs2 inputs
out = sbs(a,b,a)
out =
1     2     3     7     7     7     1     2     3
4     5     6     8     8     8     4     5     6
```

See Also madd, mmult, sel, vdiag

Purpose

Create a matrix which is made up of norms of subblocks. Used in conjunction with mu (μ)

Syntax

```
matout = blknorm(matin,blk)
```

Description

blknorm computes the maximum singular value of the subblocks of matin, using the information in the perturbation block structure, blk. The output of blknorm is matout whose entries are the maximum singular value of the subblocks of matin with these norms as elements. This helps to show which parts of the matrix are contributing to making μ large. A more complete description of the perturbation block structure, blk, can be found with the command mu and in Chapter 4. blknorm is best used on scaled matrices from the upper bound for μ. Repeated δI blocks are treated the same way as full blocks. The function blknorm can be applied to both CONSTANT and VARYING matrices.

Examples

Create a 4 × 3 random matrix and determine its subblock norms for two different block structures. The first block structure consists of a two element repeated block and a 1 × 2 full block. The second block structure consists of a 1 × 1 block, a 1 × 2 full block, and a 1 × 1 block.

```
m = crand(4,3);
disp(m)

0.7012 + 0.9826i    0.0475 + 0.0727i    0.7564 + 0.4364i
0.9103 + 0.7227i    0.7361 + 0.6316i    0.9910 + 0.7665i
0.7622 + 0.7534i    0.3282 + 0.8847i    0.3653 + 0.4777i
0.2625 + 0.6515i    0.6326 + 0.2727i    0.2470 + 0.2378i

disp(blknorm(m,[2 0; 1 2]))
   1.8498    1.5272
   1.6656    0.6923

mprintf(blknorm(m,[1 1; 1 2; 1 1]), '%6.2f')
   1.21    0.09    0.87
   1.58    1.35    1.39
   0.70    0.69    0.34
```

Algorithm

The maximum singular value of each block associated with the `blk` structure is calculated via the MATLAB `norm`.

See Also

`mu`, `norm`, `vnorm`

cjt, transp, vcjt, vtp

Purpose

Transpose and conjugate transpose of CONSTANT, SYSTEM and VARYING matrices

Syntax

```
out = cjt(mat)
out = transp(mat)
out = vcjt(mat)
out = vtp(mat)
```

Description

`cjt` forms the complex conjugate transpose of the input matrix `mat` and `transp` forms the transpose of `mat`. `transp` outputs similar results to the MATLAB command .′. These commands also work on SYSTEM and VARYING matrices. For consistency in our naming convention, `vcjt` and `vtp` are the same commands as `cjt` and `transp`, but work on just CONSTANT and VARYING matrices.

For a SYSTEM matrix `mat`, `transp`, and `cjt` are defined as

$$\texttt{mat} = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right]$$

$$\text{transp(mat)} = \left[\begin{array}{c|c} A.' & C.' \\ \hline B.' & D.' \end{array}\right], \quad \text{cjt(mat)} = \left[\begin{array}{c|c} -A' & -C' \\ \hline -B' & -D' \end{array}\right]$$

Examples

Create a SYSTEM matrix and calculate its transpose and conjugate transpose using cjt and transp.

```
A = [-10 0; 0 3];
B = [1 0 3; 0 2 -9];
C = [10 0; 0 4];
D = [0 -.2 -45; .82 0 .1];
out = pck(A,B,C,D);
seesys (out, '%5 .2g')

-10      0    |     1       0       3
 0       3    |     0       2      -9
--------------------------------------
 10      0    |     0     -.2     -45
 0       4    |      .82    0       .1


x = transp (out);
seesys (x, '%5 .2g')


-10      0    |     10      0
 0       3    |     0       4
---------------------------
 1       0    |     0      .82
 0       2    |    -.2      0
 3      -9    |    -45     .1

x = cjt(out);
seesys (x, '%5 .2g')


10      0    |    -10      0
 0      3    |     0      -4
---------------------------
 1      0    |     0      .82
 0      2    |   -.2       0
 3     -9    |   -45      .1
```

Algorithm

These functions call the MATLAB commands ′ and .′ consistent with the type of input matrices.

See Also

`vdiag`

Purpose

Approximately solves the constant-matrix μ-synthesis problem, via the minimization

$$\min Q \in C^{r \times t} \mu_{\Delta}(R + UQV)$$

for given matrices $R \in \mathbf{C}^{n} \times^{m}$, $U \in \mathbf{C}^{n} \times^{r}$,$V \in \mathbf{C}^{t} \times^{m}$, and a set $\Delta \subset \mathbf{C}^{m} \times^{n}$.

Syntax

```
[bnd,dvec,gvec,qopt] = cmmusyn(R,U,V,blk);
```

Algorithm

This works for CONSTANT or VARYING data in *R*, *U*, and *V*. If two or more matrices are VARYING, the independent variable values of these matrices must be the same.

The approximation to solving the constant matrix μ synthesis problem is two-fold: only the upper bound for μ is minimized, and the minimzation is not convex, hence the optimum is generally not found. If *U* is full column rank, or *V* is full row rank, then the problem can (and is) cast as a convex problem, [PacZPB], and the global optimizer (for the upper bound for μ) is calculated.

The upper bound is returned in `bnd`, and the optimizing *Q* is returned in `qopt`. The scaling matrices associated with the upper bound are in `dvec` and `gvec` and may be unwrapped into block diagonal form using `muunwrap`.

Reference

Packard, A.K., K. Zhou, P. Pandey, and G. Becker, “A collection of robust control problems leading to LMI’s,” *30th IEEE Conference on Decision and Control*, pp. 1245–1250, Brighton, UK, 1991.

See Also

`genmu`, `mu`

cos_tr, sin_tr, step_tr

Purpose

Generate a VARYING matrix containing a cosine, sine, or step signal at an evenly spaced time interval

Syntax

```
out = cos_tr(freq,mag,tinc,lastt)
out = sin_tr(freq,mag,tinc,lastt)
out = step_tr(timedata,stepdata,tinc,lastt)
```

Description

`cos_tr`, `sin_tr`, and `step_tr` generate time signals for use with the `trsp` (time response) command. The following are the input variables provided to `cos_tr` and `sin_tr`.

Inputs to `cos_tr` and `sin_tr`:

`freq`	frequency of the cosine (sine) signal (radians/second)
`mag`	magnitude of signal
`tinc`	time step (increment)
`lastt`	final time (the signal starts at time t=0)

Output:

`out`	time varying matrix of the cosine (sine) signal

Inputs to `step_tr`:

`timedata`	time at which step occurs (vector)
`stepdata`	magnitude of step (vector)
`tinc`	time step (increment)
`lastt`	final time (the signal starts at time t=0)

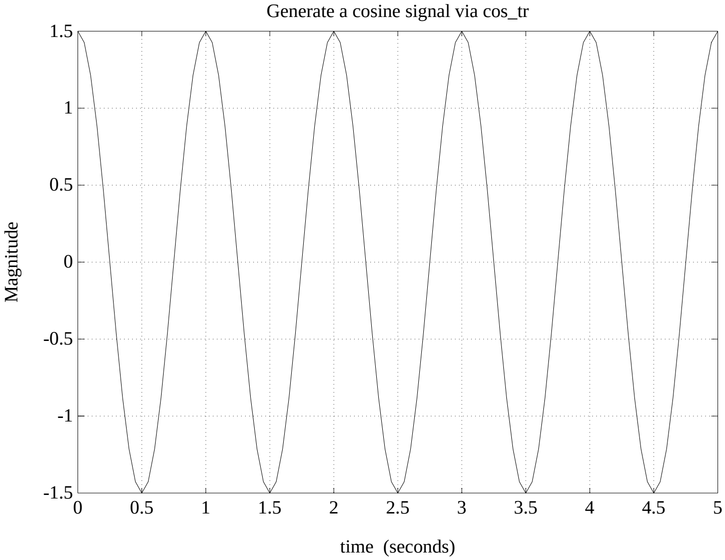
Output:

`out`	time varying matrix of steps

Examples

Synthesize a cosine signal with a 1 Hz frequency and of magnitude 1.5, 5 seconds in length. The time increment is .05 second.

```
out=cos_tr(2*pi,1.5,.05,5);
vplt('iv,d',out)
grid
title('Generate a cosine signal via cos_tr')
xlabel('time (seconds)')
ylabel('Magnitude')
```



Algorithm

`cos_tr` and `sin_tr` call the MATLAB commands `cos` and `sin`, respectively.

See Also

`minfo`, `siggen`, `trsp`, `vpck`

crand, crandn, sysrand, varyrand

Purpose

Generate a random complex CONSTANT, SYSTEM or VARYING matrix

Syntax

```
out = crand(n,m)
out = crandn(n,m)
sys = sysrand(nstates,ninputs,noutputs,stabflag)
vary = varyrand(rdim,cdim,nindv,ivflg)
```

Description

`crand` and `crandn` generate random complex matrices of dimensions `n` × `m`, using the MATLAB `rand` and `randn` commands, respectively. `crand` elements are uniformly distributed and `crandn` elements are normally distributed.

Inputs to `crand` and `crandn`:

`n`	number of rows of the output matrix
`m`	number of columns of the output matrix

Output from `crand` and `crandn`:

`out`	an `n` × `m` complex matrix of random elements

`sysrand` generates a random SYSTEM matrix with `nstates` states, `ninputs` inputs, and `noutputs` outputs. Setting the `stabflag` to 1 will result in a stable, random SYSTEM. The default for `stabflag` is 0.

`varyrand` creates a random VARYING matrix with a specified number of rows (`rdim`), columns (`cdim`), and independent variable values (`nindv`). The optional argument `ivflg` sorts the independent variables to be monotonically increasing if it is set to 0 (default). Otherwise, if `ivflg` is set to a nonzero value, no sorting is done.

Examples

Create a CONSTANT, complex random matrix and a random SYSTEM matrix using the commands crand and sysrand

```
crand(4,3)
ans =
0.4764 + 0.1622i     0.9017 + 0.1351i     0.4103 + 0.4523i
0.3893 + 0.0711i     0.4265 + 0.7832i     0.1312 + 0.8089i
0.2033 + 0.3653i     0.1420 + 0.4553i     0.8856 + 0.9317i
0.0284 + 0.2531i     0.9475 + 0.3495i     0.0922 + 0.6516i

sys=sysrand(2,4,1);
seesys(sys)

0.2190    0.6789  |   0.9347    0.5194    0.0346    0.5297
0.0470    0.6793  |   0.3835    0.8310    0.0535    0.6711
------------------|---------------------------------------
0.0077    0.3834  |   0.0668    0.4175    0.6868    0.5890
```

See Also

minfo, pck, pss2sys, rand, randel, sys2pss, unpck, vpck, vunpck

Purpose

Compute an ordered, complex Schur form matrix

Syntax

```
[v,t,flgout,reig_min] = csord(m,epp,flgord,flgjw,flgeig)
```

Description

The `csord` function produces an ordered, complex Schur form matrix of the input CONSTANT square matrix `m` with

$$v' * m * v = t = \begin{bmatrix} t_{11} & t_{12} \\ 0 & t_{22} \end{bmatrix}$$

The MATLAB function `schur` is called, which results in an unordered Schur form matrix. The subroutine `cgivens` forms a complex Givens rotation matrix, which orders the t matrix as you define it. The v matrix is the transformation matrix. A series of optional input flags can be set.

`epp`	user-supplied zero tolerance (default `epp` = 0)
`flgord=0`	order eigenvalues in ascending real part (default)
`flgord=1`	partial real part ordering, with real parts less than zero first, then the $j\omega$ axis eigenvalues and finally the real parts greater than zero
`flgjw=0`	no exit condition on eigenvalue location (default)
`flgjw=1`	exit if `abs(real(eigenvalues(i)))< epp`
`flgeig=0`	no exit condition on half-plane eigenvalue distribution (default)
`flgeig=1`	exit if length(real(eigenvalues)>0) = length(real(eigenvalues)<0)

The output flag `flgout` is nominally 0. `flout` is set to 1 if there are $j\omega$-axis eigenvalues, set to 2 if there are an unequal number of positive and negative eigenvalues, or set to 3 if both conditions occur. The fourth output argument, `reig_min`, is the minimum, magnitude real part of the eigenvalues of `m`.

The ric_schr routine calls csord to solve for a stabilizing solution to a matrix Riccati equation. In this case, the m matrix has a special structure, and failure modes are flagged to avoid extra, unnecessary computations.

Algorithm

The eigenvalues are reordered by iterating through each of them and interchanging them via a bubble sort based on the input flag, flgord. The subroutine cgivens exchanges the out of order eigenvalues.

Reference

Golub, G.H. and C.F. Van Loan, *Matrix Computations*, The Johns Hopkins University Press, 1983.

See Also

cgivens, ric_schr, rsf2csf, schur

dhfnorm

Purpose

`dhfnorm` computes the H_∞ gain of a stable, discrete-time SYSTEM matrix

Syntax

```
out = dhfnorm(sys,ttol,h,iiloc)
```

Description

`dhfnorm` computes the H_∞ norm of a stable, discrete-time SYSTEM. It converts the SYSTEM to continuous-time via a bilinear transformation and then calls the routine `hinfnorm`.

The method uses the bilinear transformation from the *z*-plane to the *s*-plane given by,

$$z = \frac{1 + sh/2}{1 - sh/2}, \quad s = \frac{2z - 1}{hz + 1}$$

where *h* is the time between samples. This is a transformation between the unit disk and the left half plane. If the system `p` has a pole close to $z = -1$, then a preliminary output feedback is used to move such poles before the bilinear transformation is performed.

Input arguments:

`sys` SYSTEM matrix (discrete-time), CONSTANT or VARYING matrix

`ttol` relative tolerance of accuracy for SYSTEM matrices only,(default = 0.001)

`h` time between samples (default 1)

`iiloc` initial estimate of worst case frequency (optional)

Output arguments:

`out` a 1 × 3 vector giving a lower bound, upper bound and the frequency where the lower bound occurs.

Reference

Doyle, J.C., K. Glover, P. Khargonekar, and B. Francis, “State-space solutions to standard H_2 and H_∞ control problems,” *IEEE Transactions on Automatic Control*, vol. 34, no. 8, pp. 831–847, August 1989.

See Also

`hinfsyne`, `hinffi`, `hinfnorm`, `hinfsyn`, `h2syn`, `h2norm`, `ric_eig`, `ric_schr`, `sdhfnorm`, `sdhfsyn`

Purpose

`dhfsyn` computes an H_∞ controller for a discrete-time SYSTEM interconnection matrix

Syntax

```
[k,g,gfin,ax,ay,hamx,hamy] = dhfsyn(p,nmeas,ncon,...
                        gmin,gmax,tol,h,z0,quiet,ricmethd,epr,epp)
```

Description

`dhfsyn` calculates a discrete-time H_∞ controller that achieves the infinity norm `gfin` for the interconnection structure `p`. The controller, `k`, stablizes the discrete-time SYSTEM matrix `p` and has the same number of states as `p`. The SYSTEM `p` is partitioned

$$\mathrm{p} = \left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{array}\right]$$

where B_1 are the disturbance inputs, B_2 are the control inputs, C_1 are the errors to be kept small, and C_2 are the output measurements provided to the controller. B_2 has column size (`ncon`) and C_2 has row size (`nmeas`).

The closed-loop system is returned in `g`. The same bilinear transformation method described for `dhfnorm` is used. The controller `k` is returned that minimizes the entropy integral,

$$I = -\frac{\mathrm{gfin}^2}{2\pi}\int_{-\pi}^{\pi}\log\,\det(I-\mathrm{gfin}^{-2}\mathrm{g}(e^{j\theta})'\mathrm{g}(e^{j\theta}))\frac{1-|\mathrm{z0}^{-2}|}{|e^{j\theta}-z0^{-1}|^2}d\theta$$

The program calls the continuous-time routine `hinfsyne` and the corresponding conditions and tests need to be satisfied.

Input arguments

`p`	SYSTEM interconnection structure matrix, (stable, discrete time)
`nmeas`	number of measurements output to controller
`ncon`	number of control inputs
`gmin`	lower bound on γ
`gmax`	upper bound on γ
`tol`	relative difference between final γ values
`h`	time between samples (optional)
`z0`	point at which entropy is evaluated (default ∞)
`quiet`	controls printing on the screen 1. no printing 0. header not printed –1. full printing (default)
`ricmethod`	1.Eigenvalue decomposition (with balancing) –1. Eigenvalue decomposition (without balancing) 2. Schur decomposition (with balancing, default) –2. Schur decomposition (without balancing)
`epr`	measure of when a real part of an eigenvalue of the Hamiltonian matrix is zero (default `epr` = 1e–10)
`epp`	positive definite determination of the X_∞ and $Y\infty$ solution (default `epp` = 1e–6)

Output arguments:

`k`	H_∞ (sub) optimal controller (discrete time)
`g`	closed-loop system with H_∞ controller (discrete time)
`gfin`	final γ value associated with `k` and `g`
`ax`	X_∞ Riccati solution as a VARYING matrix with independent variable γ

`ay` Y_∞ Riccati solution as a VARYING matrix with independent variable γ

`hamx` X_∞ Hamiltonian matrix as a VARYING matrix with independent variable γ

`hamy` Y_∞ Hamiltonian matrix as a VARYING matrix with independent variable γ

Note that the outputs `ax`, `ay`, `hamx`, and `hamy` correspond to the equivalent continuous-time problems and can also be scaled and/or balanced.

The `dhfsyn` program outputs several variables, which can be checked to ensure that the above conditions are being met. For each γ value the minimum magnitude, real part of the eigenvalues of the X Hamiltonian matrices is displayed along with the minimum eigenvalue of X_∞, which is the solution to the X Riccati equation. A # sign is placed to the right of the condition that failed in the printout. This additional information can aid you in the control design process.

Algorithm

`dhfsyn` uses the above bilinear transformation to continuous-time and then the formulae described in the Glover and Doyle paper for solution to the optimal H_∞ control design problem.

Subroutines called. `hinfsyne, hinf_st, hinf_gam, hinfe_c:`
`hinf_gam` calls `ric_eig`, `ric_schr`, `csord`, and `cgivens`

Reference

Doyle, J.C., K. Glover, P. Khargonekar, and B. Francis, “State-space solutions to standard H_2 and H_∞ control problems,” *IEEE Transactions on Automatic Control*, vol. 34, no. 8, pp. 831–847, August 1989.

Glover, K., and J.C. Doyle, “State-space formulae for all stabilizing controllers that satisfy an H_∞ `norm` bound and relations to risk sensitivity,” *Systems and Control Letters*, vol. 11, pp. 167–172, 1988.

See Also

`hinfsyne`, `hinffi`, `hinfnorm`, `hinfsyn`, `h2syn`, `h2norm`, `ric_eig`, `ric_schr`, `sdhfnorm`, `sdhfsyn`

Purpose

A script file for μ synthesis via *D*–*K* iteration

Syntax

```
dkit
```

Description

`dkit` is a μ-Tools script file for *D*–*K* iteration. The *D*–*K* iteration procedure is an approximation to μ synthesis control design. It involves a sequence of minimizations, first over the controller variable *K* (holding the *D* variable associated with the scaled μ upper bound fixed), and then over the *D* variable (holding the controller *K* variable fixed). The *D*–*K* iteration procedure is not guaranteed to converge to the minimum μ value, but often works well in practice. A detailed description of the *D*–*K* iteration can be found in Chapter 5.

`dkit` automates the *D*–*K* iteration procedure but requires the initialization of several variables. The file `dk_defin.m` is an example of the information required by `dkit`. You can copy this file from the μ-Tools subroutine directory `mutools/subs` and modify it for your application. This file can also be renamed. After renaming, assign the variable `DK_DEF_NAME` in the MATLAB workspace to the (character string) name of the new file containing the user-defined variables for `dkit`. For example, if the filename containing the setup data is `himat_def.m`, then

```
DK_DEF_NAME = 'himat_def';
```

should be executed in the MATLAB workspace.

The following is a list of what occurs during a single, complete step of the *D*–*K* iteration.

1 Upon running `dkit`, the program prompts you for starting *D*–*K* iteration number.

```
Starting mu iteration #:
```

Type 1 to indicate the first *D*–*K* iteration.

2 (In the 1st iteration, this step is skipped.) The μ calculation (from the previous step) provides a frequency-dependent scaling matrices, D_f. The fitting procedure is interactive (`msf`), and fits these scalings with rational, stable transfer function matrices, $\hat{D}(s)$.

After fitting, plots of

$$\bar{\sigma}(D_f(j\omega)F_L(P,K)(j\omega)D_f^{-1}(j\omega))$$

and

$$\bar{\sigma}(\hat{D}(j\omega)F_L(P, K)(j\omega)\hat{D}_{\mathrm{f}}^{-1}(j\omega))$$

are shown for comparison.

3. (In the 1st iteration, this step is skipped.) The rational $\hat{D}$ is absorbed into the open-loop interconnnection for the next controller synthesis. Using either (based on GMAX_DK_PLAN, see below) the previous frequency-dependent D's or the just-fit rational $\hat{D}$, an estimate of an appropriate value for GMAX_DK is made. This is simply a conservative value of the scaled closed- loop H_∞ norm, using the most recent controller and either a frequency sweep (using the frequency-dependent Ds) or a state-space calculation (with the rational D's).
4. (In the 1st iteration, this step is skipped.) The parameters that will be used in the upcoming hinfsyn are displayed. It is your option to change any/all/ none of these.
5. (The 1st iteration begins at this point.) A controller is designed using H_∞ synthesis on the scaled open-loop interconnection.
   a. The progress of the γ-iteration is displayed.
   b. The singular values of the closed-loop frequency response are plotted.
   c. You are given the option to change the frequency range (OMEGA_DK). If you change it, all relevant frequency responses are automatically recomputed.
   d. You are given the option to rerun the H_∞ synthesis with modified hinfsyn parameters. This is convienient if, for instance, the bisection tolerance was too large, or if GMAX_DK was too small.
6. Using the block structure BLK_DK, bounds for the structured singular value of the closed-loop system are calculated and plotted.
7. An iteration summary is displayed, showing all of the controller order, as well as the peak value of μ of the closed-loop frequency responses.
8. The choice of stopping, or performing another iteration is given.

Subsequent iterations proceed along the same lines without the need to re-enter the iteration number. A summary at the end of each iteration is updated to reflect data from all previous iterations. This often provides valuable information about the progress of the robust controller synthesis procedure.

To continue iterating on a problem that was started earlier, make sure the relevent data is in the workspace, run dkit and type the number of the iteration you would like to begin.

The following is a list of the variables that you must set (in either dk_defin file or the file defined by DK_DEF_NAME) and their meanings.

NOMINAL_DK Nominal plant interconnection structure, a μ-Tools SYSTEM matrix.

NMEAS_DK Number of sensor measurements.

NCONT_DK Number of control inputs.

BLK_DK Block structure forμ calculation, used by mu.

OMEGA_DK Frequency response range.

The following is a list of the *optional* variables that may be set (in either dk_defin file or the file defined by DK_DEF_NAME) and their meanings.

GMIN_DK Lower bound for first H_∞ controller design

GMAX_DK Upper bound for first H_∞ controller design

GTOL_DK H_∞ bisection tolerance for first iteration

GMAX_DK_PLAN Estimates for GMAX_DK in subsequent iterations are made in three manners. The value of GMAX_DK_PLAN determines which formula is used:

0 Use previous GMAX_DK

1 use 1.2*pkvnorm(peak_mu_value)

2 use hinfnorm(mmult(dsysL,clp,minv(dsysR)))

A number of variables are saved in the workspace after each iteration. Some of these variables are required every iteration, hence, it doesn't make sense to recompute them. The other variables are outputs from the *D*–*K* iteration procedure. The variables saved after each iteration are

`bnds_dk(i)` Frequency domain upper and lower bounds for μ associated with the *i*th iteration. The (i) denotes the *i*th iteration which is augmented to the name by the program `dkit`.

`dl_dk(i)` Left state-space *D*-scale associated with *i*th iteration. The (i) denotes the *i*th iteration which is augmented to the name by the program `dkit`. Hence, `dl_dk5` would be the left state-space *D*-scale from the fifth iteration.

`dr_dk(i)` Right state-space *D*-scale associated with *i*th iteration. Same notation.

`Dscale_dk(i)` *D*-scaling data output from `mu` associated with *i*th iteration. `Dscale_dk(i)` is in compressed form. Same notation.

`gf_dk(i)` The H_∞ `norm` of the *i*th iteration closed-loop system. Same notation.

`k_dk(i)` Controller from the *i*th iteration. Same notation.

`nom_dk_g` Frequency response of `NOMINAL_DK` using `OMEGA_DK`.

`sens_dk(i)` Sensitivity data output from the *i*th iteration `mu` calculation. Same notation.

Fitting *D*-Scalings

The *D*-scale fitting procedure is interactive and uses the μ-Tools command `msf`. During step 2 of the *D*–*K* iteration procedure, you are prompted to enter your choice of options for fitting the *D*-scaling data. After pressing return, the following is a list of your options.

```
Enter Choice (return for list):
Choices:
nd          Move to Next D-Scaling
nb          Move to Next D-Block
```

```
i           Increment Fit Order
d           Decrement Fit Order
apf         Auto-PreFit
mx 3        Change Max-Order to 3
at 1.01     Change Auto-PreFit tol to 1.01
0           Fit with zeroth order
2           Fit with second order
n           Fit with n'th order
e           Exit with Current Fittings
s           See Status
```

- `nd` and `nb` allow you to move from one *D*-scale data to another. `nd` moves to the next scaling, whereas `nb` moves to the next scaling block. For scalar *D*-scalings, these are identical operations, but for problems with full *D*-scalings, (perturbations of the form δI) they are different. In the (1,2) subplot window, the title displays the *D*-scaling Block number, the row/column of the scaling that is currently being fit, and the order of the current fit (with `d` for data, when no fit exists).
- The order of the current fit can be incremented or decremented (by 1) using `i` and `d`.
- `apf` automatically fits each *D*-scaling data. The default maximum state order of individual *D*-scaling is 5. The `mx` variable allows you to change the maximum *D*-scaling state order used in the automatic prefitting routine. `mx` must be a positive, nonzero integer. `at` allows you to define how close the rational, scaled μ upper bound is to approximate the actual μ upper bound in a norm sense. Setting `at 1` would require an exact fit of the *D*-scale data, and is not allowed. Allowable values for `at` are greater than 1. This setting plays a role (mildly unpredictable, unfortunately) in determining where in the (*D*,*K*) space the *D*–*K* iteration converges.
- Entering a positive integer at the prompt will fit the current *D*-scale data with that state order rational transfer function.
- `e` exits the *D*-scale fitting to continue the *D*–*K* iteration.
- The variable `s` will display a status of the current and fits.

Examples

An example of using dkit for *D–K* iteration is provided in the “HIMAT Robust Performance Design Example” in Chapter 7.

Reference

Balas, G.J.and J.C. Doyle, “Robust control of flexible modes in the controller crossover region,” *AIAA Journal of Guidance, Dynamics and Control*, Vol. 17, no. 2, pp. 370–377, March-April, 1994.

Balas, G.J., A.K. Packard and J.T. Harduvel, “Application of μ-synthesis techniques to momentum management and attitude control of the space station,” *AIAA Guidance, Navigation and Control Conference*, New Orleans, August 1991.

Doyle, J.C., Doyle, K. Lenz, and A. Packard, “Design examples using μ-synthesis: Space shuttle lateral axis FCS during reentry,” *NATO ASI Series*, Modelling, Robustness, and Sensitivity Reduction in Control Systems, vol. 34, Springer-Verlag, 1987.

Packard, A., J. Doyle, and G. Balas, “Linear, multivariable robust control with a μ perspective,” *ASME Journal of Dynamic Systems, Measurement and Control*, 50th Anniversary Issue, vol. 115, no. 2b, pp. 310–319, June 1993.

Stein, G., and J. Doyle, “Beyond singular values and loopshapes,” *AIAA Journal of Guidance and Control*, vol. 14, no. 1, pp. 5–16, January, 1991.

See Also

hinfsyn, hinfnorm, msf, mu

dkitgui

Purpose

A graphical user interface for μ synthesis via *D–K* iteration

Syntax

```
dkitgui
```

Description

`dkitgui` is a graphical user interface (GUI) for *D–K* iteration. The *D–K* iteration procedure is an approximation to μ synthesis control design. It involves a sequence of minimizations, first over the controller variable *K* (holding the *D* variable associated with the scaled μ upper bound fixed), and then over the *D* variable (holding the controller *K* variable fixed). The *D–K* iteration procedure is not guaranteed to converge to the minimum μ value, but often works well in practice. A more detailed description of the *D–K* iteration can be found in Chapter 5.

The GUI tool, `dkitgui`, has five windows. They are:

- **Main Iteration** window, which is the main interface for the user during the iteration.
- **Setup** window, where initial data is entered.
- **Parameter** window, which is occasionally used to modify properties of the *D–K* iteration, such as H_∞ parameters, and to select the variables that are automatically exported to the workspace each iteration.
- **Frequency Response** window, where the plots of μ and $\bar{\sigma}$ of the closed-loop transfer function matrix are displayed.
- **Scaling** window, where the rational fits of the frequency-dependent *D*-scale data are shown, and can be modified.

`dkitgui` completely automates the *D–K* iteration procedure. A detailed description of `dkitgui` and its use can be found in Chapter 5.

Examples

An example of using `dkitgui` for *D–K* iteration is provided in the “HIMAT Robust Performance Design Example” section in Chapter 7.

Reference

Balas, G.J.and J.C. Doyle, “Robust control of flexible modes in the controller crossover region,” *AIAA Journal of Guidance, Dynamics and Control,* vol. 17, no. 2, pp. 370–377, March-April, 1994.

Balas, G.J., A.K. Packard and J.T. Harduvel, “Application of μ-synthesis techniques to momentum management and attitude control of the space

station," *AIAA Guidance, Navigation and Control Conference,* New Orleans, pp. 1204–1213, August 1991.

Doyle, J.C., Doyle, K. Lenz, and A. Packard, "Design examples using synthesis: Space shuttle lateral axis FCS during reentry," *NATO ASI Series*, Modelling, Robustness, and Sensitivity Reduction in Control Systems, vol. 34, Springer-Verlag, 1987.

Packard, A., J. Doyle, and G. Balas, "Linear, multivariable robust control with a perspective," *ASME Journal of Dynamic Systems, Measurement and Control,* 50th Anniversary Issue, vol. 115, no. 2b, pp. 310–319, June 1993.

Stein, G., and J. Doyle, "Beyond singular values and loopshapes," *AIAA Journal of Guidance and Control,* vol. 14, no. 1, pp. 5–16, January, 1991.

See Also

`dkit`, `hinfsyn`, `hinfnorm`, `mu`, `pkvnorm`

Purpose

Provide an interactive mouse-based log-log sketch and fitting tool

Syntax

```
[sysout,pts] = drawmag(in,init_pts)
```

Description

`drawmag` interactively uses the mouse in the plot window to create a VARYING matrix `pts` and a stable, minimum-phase SYSTEM `sysout`, which approximately fits, in magnitude, the frequency VARYING matrix in `pts`.

Input arguments:

in	either a VARYING matrix which is plotted each time as a reference or a CONSTANT matrix of the form [x_{min} x_{max} y_{min} y_{max}] specifying the plot window on the data.
`init_pts`	optional VARYING matrix of initial set of points

Output arguments:

`sysout`	SYSTEM matrix fitting `pts`.
`pts`	VARYING matrix of points.

While `drawmag` is running, all interaction with the program is through the mouse and/or the keyboard. The mouse, if there is one, must be in the plot window. The program recognizes several commands:

- Clicking the mouse button adds a point at the crosshairs. If the crosshairs are outside the plotting window, the points will be plotted when the fitting, windowing, or replotting modes are invoked. Typing a is the same as clicking the mouse button.
- Typing `r` removes the point with frequency nearest that of the crosshairs.
- Typing any integer between `0-9` fits the existing points with a transfer function of that order. The fitting routine approximately minimizes the maximum error in a log sense. The new fit is displayed along with the points, and the most recent previous fit, if it exists.
- Typing `w` uses the crosshair location as the initial point in creating a window. Moving the crosshairs and clicking the mouse or pressing any key then gives a second point at the new crosshair location. These two points define a new

window on the data, which is immediately replotted. This is useful in fine tuning parts of the data. Windowing may be called repeatedly.

- Typing `p` simply replots the data using a window that covers all the current data points as well as whatever was specified in `in`. Typically used after windowing to view all the data.
- Typing `k` invokes the keyboard using the `keyboard` command. Caution should be exercised when using this option, as it can wreak havoc on the program if variables are changed.

See Also `axis`, `ginput`, `magfitlp`, `musynfit`, `vplot`

Purpose

`dypert` operates on output from a VARYING `mu` (μ) calculation to construct a worst-case, real-rational, stable perturbation

`sisorat` constructs a first-order, real-rational, stable, all-pass transfer function that interpolates a particular complex number at a given, positive frequency.

Syntax

```
pert = dypert(pvec,blk,bnds);
pert = dypert(pvec,blk,bnds,blkindex);
ratfit = sisorat(value);
```

Description

The input to `dypert` consists of the perturbation `pvec`, the block structure `blk`, and lower and upper bounds, `bnds`, produced from a VARYING matrix μ calculation. (See the μ-Tools command `mu` for a more complete description of these variables.) By searching in `bnds`, `dypert` finds the peak value of the lower bound — for example γ, occurring at frequency ω_0. `dypert` then extracts the perturbation (call this matrix Δ_0) from `pvec` at the frequency ω_0. `dypert` constructs a SYSTEM matrix, `pert`, which is stable, and has the block-diagonal structure associated with `blk`, and also satisfies the equations

$$\mathrm{pert}(j\omega_0) = \Delta_0 \qquad \text{and} \qquad \|\mathrm{pert}\|_\infty = \frac{1}{\gamma}$$

The command `dypert` can also be called with four input arguments. In this case, the last argument is a vector of integers. This vector, `blkindex`, specifies which blocks in the perturbation structure specified by the `blk` vector, are to be used to construct the rational perturbation. For instance, if the block structure specified by `blk` is

$$\Delta := \left\{ \begin{matrix} \mathrm{diag}[\Delta_1, \delta_2 I_{4\times4}, \delta_3 I_{2\times2}, \Delta_4, \Delta_5, \delta_6 I_{3\times3}, \Delta_7] : \Delta_1 \in \mathbf{C}^{2\times3}, \\ \delta_2 \in \mathbf{C}, \delta_3 \in \mathbf{C}, \Delta_4 \in \mathbf{C}^{4\times2}, \Delta_5 \in \mathbf{C}^{3\times3}, \delta_6 \in \mathbf{C}, \Delta_7 \in \mathbf{C}^{2\times1} \end{matrix} \right\}$$

and the fourth argument to dypert is [3 6 4 1] (indicating first, third, fourth, and sixth blocks are desired), then the output SYSTEM matrix produced by dypert will be a SYSTEM matrix that looks like

$$\begin{bmatrix} \Delta_1(s) & 0_{2\times 2} & 0_{2\times 2} & 0_{2\times 3} \\ 0_{2\times 3} & \delta_3(s)I_{2\times 2} & 0_{2\times 2} & 0_{2\times 3} \\ 0_{4\times 3} & 0_{4\times 3} & \Delta_4(s) & 0_{4\times 3} \\ 0_{3\times 3} & 0_{3\times 3} & 0_{3\times 2} & \delta_6(s)I_{3\times 3} \end{bmatrix}$$

This is useful when some of the perturbations are not physical perturbations, but correspond to performance objectives, and hence do not need to be constructed. Note that regardless of the order in which the numbers occur in the fourth argument, the perturbations in the output SYSTEM pert are in ascending order. When dypert is called with three arguments, all the perturbations are constructed.

The command sisorat is essentially a scalar version of dypert, and is the main subroutine for dypert. The input to sisorat is value, a 1×1, VARYING matrix, with one independent variable value. The independent variable is interpreted as a frequency, ω_0, and the numerical value of value at that frequency is denoted by γ. The output of sisorat is a single-input/single-output stable, real, SYSTEM matrix, ratfit, satisfying

$$\text{ratfit}(j\omega_0) = \gamma \quad \text{and} \quad |\text{ratfit}(j\omega)| = |\gamma| \ \ \forall\omega$$

Algorithm

The main subroutine of dypert is sisorat, which operates on the following fact. For any complex number γ, and any real frequency $\omega_0 > 0$, there is a real number $\beta > 0$ such that by proper choice of sign, the equality

$$\pm|\gamma| \left.\frac{s-\beta}{s+\beta}\right|_{s = j\omega_0} = \gamma$$

holds. This is exactly how sisorat works.

In order to understand how dypert works, recall in Chapter 4, the section “Complex Structured Singular Value,” Remark 1 after Definition 2.1. There it was shown that each full block of a perturbation that causes singularity can in fact be chosen to be a dyad, and the program `mu` does this at each independent variable value. Hence, the matrix Δ_0, as described above, is a diagonal augmentation of scalar blocks, and dyads. Given that the *i*th block is a dyad, write it as $\Delta_{0_i} = y_i x_i^*$ for complex vectors y_i and x_i. Using several calls to `sisorat`, it is possible to create two stable, rational vectors $h(s)$ (column) and $r(s)$ (row), such that each element of the vectors is of the form generated by `sisorat` (stable, and flat across frequency), and for each k and l

$$h_k(j\omega_0) = y_{i_k}(j\omega_0) \quad , \quad r_l(j\omega_0) = \bar{x}_{i_l}(j\omega_0)$$

If we define $\hat{\Delta}_i(s) := h(s)r(s)$ then it is stable, and

$$\|\hat{\Delta}_i\|_\infty = \bar{\sigma}(\Delta_{0_i}) \quad , \quad \hat{\Delta}_i(j\omega_0) = \Delta_{0_i}$$

Performing this on a block-by-block basis produces the entire rational perturbation. For the case of a repeated scalar block, that part of the perturbation is constructed directly with one call to `sisorat` and diagonally augmented as many times as needed.

See Also

`mu`, `randel`, `svd`, `vsvd`

Purpose

Fit single-input/single-output magnitude data with a real, rational, minimum phase transfer function

Syntax

```
sys = fitmag(magdata,weight,heading,oldfit,...
     dmdi,upbd,blk,blknum)
resp = genphase(d)
sys = fitmaglp(magdata,weight,heading,oldfit,...
     dmdi,upbd,blk,blknum)
[sys,fit] = magfit(magdata,dim,weight)
```

Description

fitmag fits a stable, minimum phase transfer function to magnitude data, magdata, with a supplied frequency domain weighting function, weight. Both of these are VARYING matrices, with identical independent variable values. fitmag uses genphase to generate phase data, and fitsys to do the fit.

genphase uses the complex-cepstrum algorithm to generate a complex frequency response, resp, whose magnitude is equal to the real, positive response d, but whose phase corresponds to a stable, minimum phase function.

fitmaglp has the same inputs, outputs, and user interaction as fitmag , but uses a linear programming approach to do the fitting instead of fitsys and genphase.

fitmag and fitmaglp have the additional input arguments dmdi, upbd, blk, and blknum. These arguments are used exclusively with *D–K* iteration when called by musynfit and musynflp. In this case, the magdata is the dvec output of the mu program and corresponds to the blknum'th frequency varying *D* scale to be fit. weight corresponds to a measure of the sensitivity of mu to changes in the *D* scales at each frequency. This is the sens output from mu. heading is a string variable denoting the title of the plot and oldfit is usually the *D* scalings from the previous *D–K* iteration.

dmdi represents the VARYING matrix analyzed using mu. upbd is the upper bound calculated using mu of dmdi with the perturbation block structure blk. The last argument, blknum, corresponds to the current *D* scale in the block structure being fit with fitmag or fitmaglp. Upon fitting the magnitude data, magdata, the resulting transfer function sys is absorbed into the original matrix dmdi and plotted along with the mu upper bound on the lower graph.

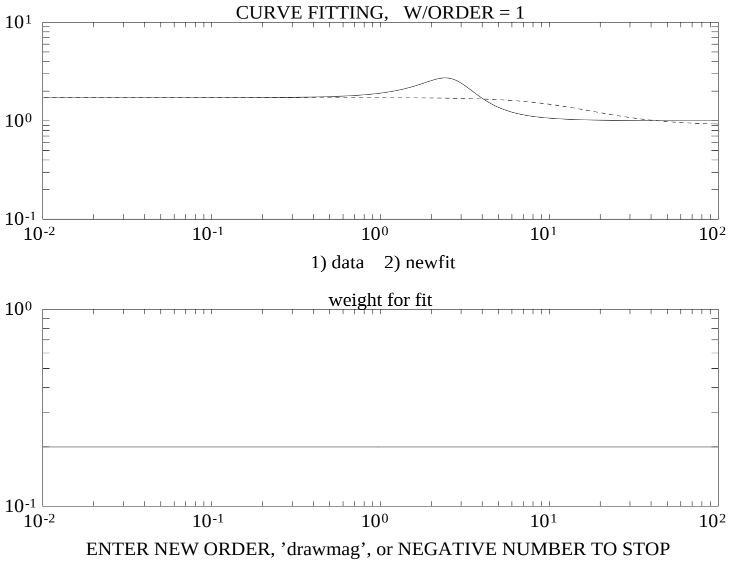
magfit is a batch version of fitmaglp that eliminates the user interaction. The weight is optional, but dim is a required argument of parameters for the linear program. dim has the form [hmax htol nmin nmax] where

- hmax is a measure of the allowable error in the fit.
- htol is a measure of the accuracy with which the optimization is carried out.
- nmin and nmax are the minimum and maximum orders considered for the curve fit.

Examples

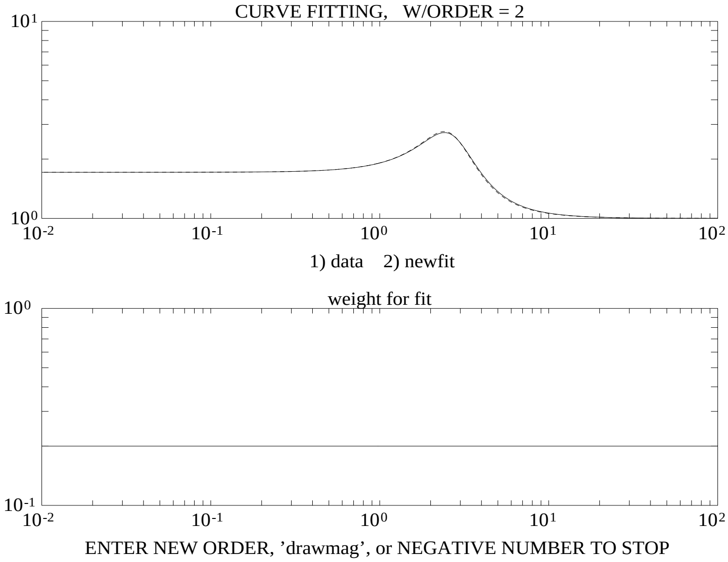
Create a second-order transfer function sys to test fitmag. Fit its magnitude data with a first- and second-order transfer function via fitmag.

```
sys = nd2sys([1 -5 12],[1 2 7]);
w = logspace(-2,2,200);
sysg = frsp(sys,w);
wgt = 0.2;
wgtg = frsp(wgt,w);
sysfit = fitmag(vabs(sysg),wgtg);
ENTER ORDER OF CURVE FIT or 'drawmag' 1
```



A first-order fit does not accurately respresent the frequency data as shown in the above figure. The solid line represents the curve fit and the dashed line represents the original frequency data. You can try and fit the data again with a second-order system.

```
ENTER ORDER, 'drawmag', or NEGATIVE TO STOP 2
```



```
ENTER ORDER, 'drawmag', or NEGATIVE TO STOP -1
```

The second-order fit lies directly on top of the original data, hence it is difficult to distinguish the two plots. A `-1` is entered at the end of this iteration procedure to indicate satisfaction with the results.

Algorithm

The algorithm for `fitmag` is as follows. On a log-log scale, the magnitude data is interpolated linearly, with a very fine discretization. Then, using the complex cepstrum algorithm, the phase, associated with a stable, minimum phase, real, rational transfer function with the same magnitude as the `magdata` variable is generated. This involves two fft's, and logarithmic/exponential conversions. With the new phase data, and the input magnitude data, the MATLAB function `invfreqs` is used to find a real, rational transfer function that fits the data. `heading` is an optional title and `oldfit` is an optional previous fit that can be added to the graphs if they are included. These options are used in the program musynfit.

The algorithm for magfit is as follows. The system sys is derived by solving a linear program and searching over a parameter h according to the following specification. Let m be the given magnitude data, g the transfer function of sys and w the values of weight; then h is found such that at each frequency,

$$1/r < \left|\frac{g}{m}\right| < r,$$

where

$$r = \sqrt{1 + h/w^2}\cdot$$

The order of sys is increased until an h less than hmax is obtained or nmax is reached. The minimum value of h at this order is then determined to an accuracy of htol.

Problems

For problems with very coarse data, fitmag may give incorrect answers, even if the data was generated by taking the frequency response of a linear system. The inaccuracy arises in the log-log interpolation step, which is used in the phase calculation. This step is avoided in magfit. Hence, for coarse data sets, magfit should be used. fitmaglp and magfit appear to be sometimes slower than fitmag.

Reference

Oppenheim, A.V., and R.W. Schaffer, *Digital Signal Processing*, Prentice Hall, New Jersey, 1975, pp. 513.

See Also

fitsys, invfreqs, musynfit, musynflp, muftbtch

Purpose

Fits single-input/single-output (SISO), single-input/multi-output (SIMO) and multi-input/single-output (MISO) frequency response data with SYSTEM matrix

Syntax

```
sys = fitsys(resp,ord,wt,code)
```

Description

`fitsys` fits frequency response (VARYING) data in `resp` with a transfer function of order `ord`, using a frequency dependent weight in `wt` (optional). The frequency response data may be either a row (SIMO) or column (MISO). The optional frequency dependent weight is a VARYING matrix. This weight may be a scalar (1 row, 1 column), or may be the same shape as `resp`.

The fourth argument, `code`, is optional. If set to 0 (default), then the fit is as described. If code = 1, as in the μ-synthesis routines, it forces the fit to be stable, minimum phase, simply by reflecting the poles and zeros if necessary. In this case, the response `resp` comes from the program `genphase` and already corresponds to a stable, minimum phase transfer function. `fitsys` is called by `fitmag` and `msf`.

Examples

An example of how to use `fitsys` to derive a SIMO transfer function model is provided in the “More Sophisticated SYSTEM Functions” section in Chapter 2.

Algorithm

The author of `fitsys` is Xin Hua Yang.

See Also

`fitmag`, `genphase`, `msf`, `vspect`

frsp

Purpose

Calculate the complex frequency response of a linear system

Syntax

```
out = frsp(sys,omega,T,balflg)
```

Description

`frsp` calculates the complex frequency response of a given SYSTEM matrix (`sys`) for a vector of frequency points (`omega`). The output matrix `out` is a frequency dependent VARYING matrix containing the frequency response of the input system `sys` at the frequency values contained in the vector `omega` . For systems with multiple inputs and outputs, a multivariable frequency response is returned.

Input arguments:

`sys` SYSTEM matrix

response calculated at these frequencies. If another VARYING matrix is input here, then its independent variables are used

`T` 0 (default) indicates a continuous system. A nonzero value forces discrete system evaluation with sample time `T` (optional)

`balflg` 0 (default) balances the SYSTEM A matrix prior to evaluation. A nonzero value for `balflg` leaves the state-space data unchanged (optional)

Output arguments

`out` VARYING frequency response matrix

The vector of frequency points is assumed to be real and can be generated from the MATLAB command `logspace` or `linspace`. Given a continuous system `sys`, of the form

$$\text{sys} = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right]$$

and an input vector, `omega` , with N frequencies, $[\omega_1, \omega_2, \ldots, \omega_N]$, `frsp` evaluates the following equation

$C(j\omega_i I - A)^{-1}B + D, \quad i = 1, . . ., N$

You can specify a discrete time evaluation by specifying an optional sampling time, T. For the discrete time case each matrix in the VARYING output is given by

$$C(e^{j\omega_i T} - A)^{-1}B + D, \quad i = 1, \ldots, N$$

Note that setting `T` = 0 implies that a continuous frequency response is to be performed and not to evaluate

$$C(e^{j\omega_i 0} - A)^{-1}B + D$$

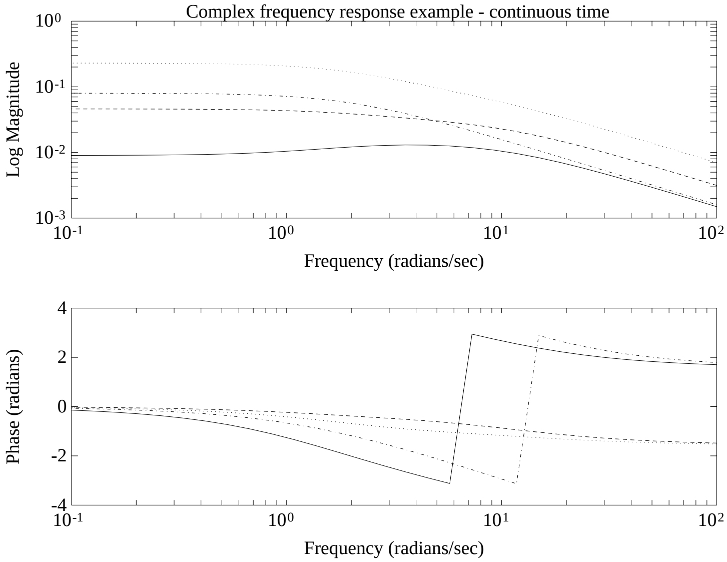
Examples

The SYSTEM matrix `sys` is constructed to have two inputs and two outputs with poles at –2 and –10. A frequency vector `omega` is constructed with 30 points log spaced between .1 and 100 rad/s. The complex frequency response of `sys` is calculated and its values between 3.5 and 4.6 rad/s are displayed.

```
a = [-2 0;0 -10];b = [.2 .12; -.3 .4];c = [.3 .7; 2 -1];
sys = pck(a,b,c);
omega = logspace(-1,2,30);
sysg = frsp(sys,omega);
see(xtract(sysg,3.5,4.6))
2 rows 2 columns

iv = 3.56225

-0.0114 - 0.0062i     0.0292 - 0.0165i
 0.0746 - 0.0949i    -0.0067 - 0.0386i

iv = 4.52035

-0.0125 - 0.0032i     0.0262 - 0.0172i
 0.0577 - 0.0853i    -0.0136 - 0.0294i
```

A frequency response is performed using `frsp` with the default variables set. A plot of the frequency response is shown with the four line types corresponding to the `sysg(1,1)`, `sysg(1,2)`, `sysg(2,1)`, and the `sysg(2,2)` elements.

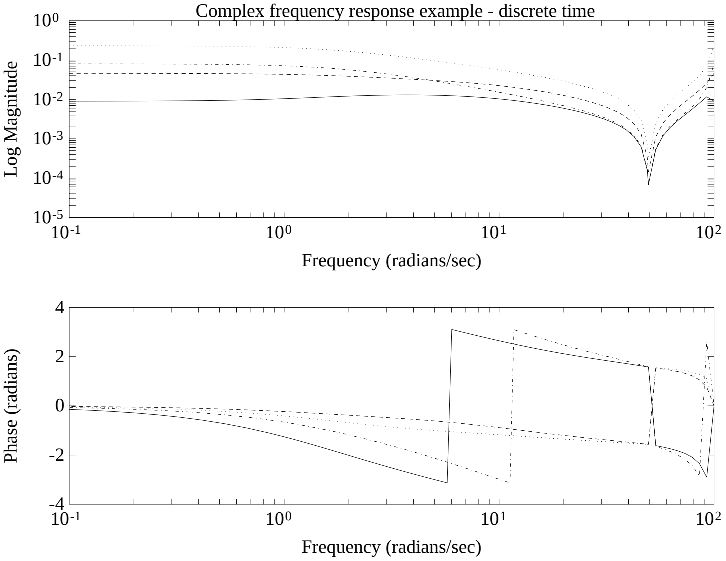
```
vplot('bode',sysg);
title('Complex frequency response example - continuous time')
```



To demonstrate the calculation of a discrete frequency response, convert this system into a digital system via the bilinear transformation. The sample frequency is chosen as 100 radians/second.

```
T = 2*pi/100;
dsys = tustin(sys,T);
omega = sort([omega,logspace(0,2,60)]);
dsysg = frsp(dsys,,T);
vplot('bode',sysg);
title('Complex frequency response example - discrete time')
```

For digital filter design, you can examine the transfer function from 0 to π by specifying T = 1. A Chebyshev type II filter is designed and its magnitude is plotted to demonstrate this feature.

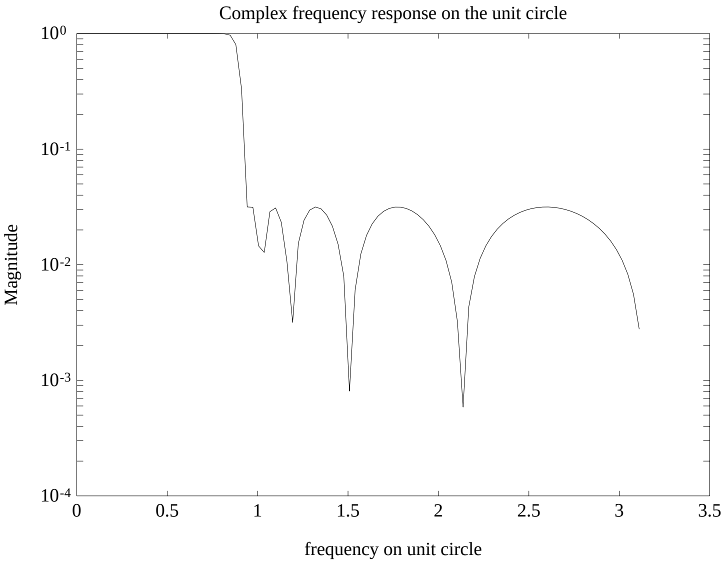


```
[a,b,c,d] = cheby2(11,30,0.3);
dfilt = pck(a,b,c,d);
omega = [0:pi/100:pi*99/100];
dsysg = frsp(dfilt, omega,1);
vplot('iv,lm',dsysg);
xlabel('frequency on unit circle')
ylabel('Magnitude')
title('Complex frequency response on the unit circle')
```

Algorithm

The algorithm to calculate the complex frequency response involves an matrix inverse problem, which is solved via a Hessenberg matrix. If `balflg` is set to 0, the frequency response balances the SYSTEM A matrix (using the MATLAB `balance` command) prior to calculation of the Hessenberg form.

Note Balancing the system may cause errors in the frequency response. If the output of `frsp` is questioned, compare the results with balancing and without balancing the SYSTEM prior to calculating the frequency response.



Reference

Laub, A.J., "Efficient Multivariable Frequency Response Computations," *IEEE Transactions on Automatic Control,* vol. AC–26, No. 2, pp. 407–408, April, 1981.

See Also

`balance`, `hess`, `samhld`, `tustin`, `vplot`

Purpose

gap calculates the gap metric between two SYSTEM matrices

nugap calculates the ν gap between two SYSTEM matrices.

Syntax

```
gap = gap(sys1,sys2, ttol)
nugap = nugap (sys1, sys2, ttol)
```

Description

gap and nugap compute the gap and ν gap metrics between two SYSTEM matrices. Both quantities give a numerical value $\delta(G_0,G_1)$ between 0 and 1 for the distance between a nominal system sys1 (G_0) and a perturbed system sys2 (G_1). The gap metric was introduced into the control literature by Zames and El-Sakkary, 1980, and exploited by Georgiou and Smith, 1990. The ν gap metric was derived by Vinnicombe, 1993. For both of these metrics the following robust performance result holds from Qui and Davidson, 1992, and Vinnicombe, 1993

$$\arcsin b(G_1,K_1) \geq \arcsin b(G_0,K_0) - \arcsin \delta(G_0,G_1) - \arcsin \delta(K_0,K_1)$$

where

$$b(G,K) = \left\| \begin{bmatrix} I \\ K \end{bmatrix} (I - GK)^{-1} [G \ \mathrm{I}] \right\|_\infty^{-1}$$

The interpretation of this result is that if a nominal plant G_0 is stabilized by controller K_0, with “stability margin” $b(G_0,K_0)$, then the stability margin when G_0 is perturbed to G_1 and K_0 is perturbed to K_1 is degraded by no more than the above formula. Note that $1/b(G,K)$ is also the signal gain from disturbances on the plant input and output to the input and output of the controller. The ν gap is always less than or equal to the gap, so its predictions using the above robustness result are tighter. To make use of the gap metrics in robust design, weighting functions need to be introduced. In the above robustness result, G needs to be replaced by W_2GW_1 and K by $W_1^{-1}KW_2^{-1}$ (similarly for G_0, G_1, K_0 and K_1). This makes the weighting functions compatible with the weighting structure in the “Loop Shaping Using H• Synthesis” section in Chapter 3. Model reduction of the system model and controller can be performed by using balanced truncations or Hankel norm approximation of normalized coprime factor representations.

ttol defines the tolerance to which the gap is computed. The default is 0.001.

Algorithm

Tryphon Georgiou and Malcolm Smith wrote the gap program.

The computation of the gap amounts to solving 2-block H_∞-problems, Georgiou, 1988. The particular method used here for solving the H_∞-problems is based on Green *et al.*, 1990. The computation of the nugap uses the method of Vinnicombe, 1993.

Reference

Georgiou, T.T., On the computation of the gap metric, *Systems Control Letters*, vol. 11, pp. 253–257, 1988.

Georgiou, T.T., and M. Smith, “Optimal robustness in the gap metric,” *IEEE Transactions on Automatic Control*, vol. 35, pp. 673–686, 1990.

Green, M., K. Glover, D. Limebeer, and J.C. Doyle, “A J-spectral factorization approach to H_∞ control,” *SIAM J. of Control and Opt.*, 28(6), pp. 1350–1371, 1990.

Qiu, L., and E.J. Davison, “Feedback stability under simultaneous gap metric uncertainties in plant and controller,” *Systems Control Letters*, vol. 18–1, pp. 9–22, 1992.

Vinnicombe, G., “Measuring Robustness of Feedback Systems,” PhD dissertation, Department of Engineering, University of Cambridge, 1993.

Zames, G., and El-Sakkary, “Unstable systems and feedback: The gap metric,” *Proceedings of the Allerton Conference*, pp. 380–385, Oct., 1980.

See Also

`dhfnorm`, `emargin`, `hinfnorm`, `ncfsyn`, `mu`

Purpose

Compute upper bounds for the mixed (real and complex) generalized structured singular value (referred to as generalized mixed μ) of a VARYING/ CONSTANT matrix

Syntax

```
[bnd,dvec,gvec,qmat] = genmu(M,C,blk);
```

Description

Generalized μ allows us to put additional constraints on the directions that $I - M\Delta$ becomes singular. Given a matrix $M \in \mathbf{C}^{n} \times^{n}$, and $C \in \mathbf{C}^{m} \times^{n}$, find the smallest $\Delta \in \Delta$ (described by `blk`) such that

$$\begin{bmatrix} I - \Delta M \\ C \end{bmatrix}$$

is not full column rank. Specifically, define

$$\mu_{\Delta}(M, C) := \frac{1}{\min\left\{\bar{\sigma}(\Delta) : \delta \in \quad \Delta, \operatorname{rank}\begin{bmatrix} I - \Delta M \\ C \end{bmatrix} < n\right\}}$$

This quantity can be bounded above, using standard μ ideas. If there exists a matrix Q such that

$$\mu_{\Delta}(M + QC) < \beta$$

then $\mu_{\Delta}(M,C) < \beta$. Hence,

$$\mu_{\Delta}(M, C) \quad \min_{Q \in \mathbf{C}^{n \times m}} \mu_{\Delta}(M + QC)$$

It is possible to compute the optimal matrix Q which minimizes the standard μ upper bound for $\mu_{\Delta}(M + QC)$. The optimization problem can be reformulated into an affine matrix inequality, [PacZPB], and solved with a combination of heuristics and general purpose AMI solvers. This is how `genmu` computes the upper bound $\mu_{\Delta}(M,C)$.

The upper bound for $\mu_{\Delta}(M,C)$ is returned in `bnd`. The scaling matrices D and G associated with the upper bound are in packed format, in the matrics `dvec` and `gvec`, and can be unwrapped with `muunwrap`. The Q matrix which mimimizes the bound is returned in qmat. If either *M* or *C* are VARYING matrices, then the bound is computed at each value of the independent variable, and the output matrices (`bnd`, `dvec`, `gvec` and `qmat`) are also VARYING matrices.

Reference

Pachard, A., K. Zhou, P. Pandey, and G. Becker, “A collection of robust control problems leading to LMI’s,” *30th IEEE Conference on Decision and Control,* pp. 1245–1250, Brighton, UK, 1991.

See Also

`cmmusyn`, `mu`

getiv, sortiv, tackon

Purpose

Return, sort, or append the independent variable values of a VARYING matrix

Syntax

```
[indv,err] = getiv(mat)
getiv(mat)
[out,err] = sortiv(mat,sortflg,nored,epp)
out = tackon(mat1,mat2)
```

Description

getiv returns the independent variable values of the VARYING matrix mat. If the input matrix mat is a VARYING matrix, the independent variable is returned as a column vector, indv, and the output err is set to 0. If mat is not a VARYING matrix, then indv is set to empty, and err is set to 1.

sortiv will reorder the independent variable and associated VARYING matrix to be monotonically increasing or decreasing. The optional sortflg is set to 0 (default) for monotonically increasing sorting or nonzero for monotonically decreasing sorting. sortiv can be used in conjunction with tackon to mesh together two different VARYING matrices. The optional third input argument, nored, is set to 0 (default) which does not reduce the number of independent variables even if there are repeated ones. Setting nored to a nonzero value causes repeated independent variables to be collapsed down if their corresponding matrices are the same. If they are not, an error message is displayed and only the first independent variable and corresponding matrix is kept. The output argument err, which is nominally 0, is set to 1 if an error message is displayed. The optional fourth input argument, epp, is a vector used for checking closeness of two variables. If two independent variables are within epp(1), and the norm of the difference between the two matrices at these points is within epp(2), sortiv collapses these two independent variable values down to one. If the two independent variables are within epp(1), and the norm condition is not satisfied, an error message is displayed and out is set to the null matrix. When nored is nonzero, the default value for epp is $[1e-9;1e-9]$.

tackon strings together two VARYING matrices placing mat1 on top of mat2. mat1 and mat2 must have the same row and column dimensions.

Examples

The frequency response VARYING matrix from the frsp example is a two-input/two-output matrix containing 30 points. These independent variables vary from 0.1 rad/sec to 100 rad/sec.

```
minfo(sysg)
varying: 30 pts2 rows2 cols
seeiv(sysg)

1.000e-01    1.269e-01    1.610e-01    2.043e-01    2.593e-01
3.290e-01    4.175e-01    5.298e-01    6.723e-01    8.532e-01
1.083e+00    1.374e+00    1.743e+00    2.212e+00    2.807e+00
3.562e+00    4.520e+00    5.736e+00    7.279e+00    9.237e+00
1.172e+01    1.487e+01    1.887e+01    2.395e+01    3.039e+01
3.857e+01    4.894e+01    6.210e+01    7.880e+01    1.000e+02
```

Typing getiv without any arguments outputs a brief description of its calling sequence. All μ-Tools commands have this feature. The xtract command selects the independent variables between 1 and 5 rad/sec and the getiv command returns these independent variables from sysg and stores them in indv.

```
getiv
usage: [indv,err] = getiv(mat)
[indv] = getiv(xtract(sysg,l,5));
indv'
ans =
1.0826    1.3738    1.7433    2.2122    2.8072
3.5622    4.5204
```

The sortiv command (with an optional second argument) resorts the independent variable of the frequency response of sys in decreasing order.

```
syslg = sortiv (xtract ( sysg ,1 , 5 ) , 1 ) ;
seeiv(syslg)

4.520e+00   3.562e+00   2.807e+00   2.212e+00   1.743e+00
1.374e+00   1.083e+00
```

Algorithm

getiv and sortiv manipulate VARYING matrices.

See Also

indvcmp, sort, xtract, xtracti

Purpose

Calculate the H_2, H_∞ norms of a SYSTEM matrix

Syntax

```
out = h2norm(sys)
out = hinfnorm(sys,tol)
```

Description

`h2norm` calculates the 2-norm of a stable, strictly proper SYSTEM matrix. The output is a scalar, whose value is the 2-norm of the system.

The output from `hinfnorm` is a 1×3 vector, `out`, which is made up (in order) of a lower bound for $\|\mathtt{sys}\|_\infty$, an upper bound for $\|\mathtt{sys}\|_\infty$, and a frequency, ω_o, at which the lower bound is achieved.

$$\mathrm{out}(1) = \bar{\sigma}(\mathrm{sys}(j \cdot \mathrm{out}(3))) \le \|\mathrm{sys}\|_\infty \le \mathrm{out}(2)$$

The $\|\cdot\|_\infty$ norm calculation is an iterative process and requires a test to stop. The variable `tol` specifies the tolerance used to calculate the $\|\mathtt{sys}\|_\infty$. The iteration stops when

(the current upper bound) ≤ (1 + `tol`) × (the current lower bound).

The default value of `tol` is 0.001.

Algorithm

The H_2 norm of a SYSTEM follows from the solution to the Lyapunov equation.

$AX + XA' + BB' = 0$,

with $\|\mathtt{sys}\|_2 = trace\,(CXC')$.

Calculation of the H_∞ `norm` requires checking for $j\omega$ axis eigenvalues of a Hamiltonian matrix, $H\alpha$, which depends on a parameter α. If $H\alpha$ has no $j\omega$ axis eigenvalues, then the $\|\cdot\|_\infty$ `norm` of the SYSTEM matrix is less than α. If the matrix $H\alpha$ does have $j\omega$ axis eigenvalues, then these occur at the frequencies where the transfer matrix has a singular value (not necessarily the maximum) equal to α. By iterating, the value of the $\|\cdot\|_\infty$ `norm` can be obtained.

Reference

Boyd, S., K. Balakrishnan and P. Kabamba, “A bisection method for computing the H_∞ norm of a transfer matrix and related problems,” *Math Control Signals and Systems*, 2(3), pp. 207–219, 1989.

Boyd, S., and K. Balakrishnan, “A regularity result for the singular values of a transfer matrix and a quadratically convergent algorithm for computing its H_∞ norm,” *Systems and Control Letters*, vol. 15–1, 1990.

Bruinsma, O., and M. Steinbuch, “A fast algorithm to compute theH_∞ norm of a transfer function matrix,” *Systems and Control Letters,* vol. 14, pp. 287–293, 1990.

See Also

`hinfsyn`, `h2syn`, `ric_eig`, `ric_schr`

Purpose

Compute the optimal H_2 controller given a SYSTEM interconnection matrix

Syntax

```
[k,g,norms,kfi,gfi,hamx,hamy] =h2syn(p,nmeas,ncon,ricmethod)
```

Description

`h2syn` calculates the H_2 optimal controller `k` and the closed-loop system `g` for the linear fractional interconnection structure `p`. `nmeas` and `ncon` are the dimensions of the measurement outputs from `p` and the controller inputs to `p`. The optional fourth argument, `ricmethod`, determines the method used to solve the Riccati equations. The interconnection structure, `p`, is defined by

$$\mathrm{p} = \left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{array}\right]$$

Input arguments:

`p` SYSTEM interconnection structure matrix

`nmeas` number of measurements output to controller

`ncon` number of control inputs

`ricmethd`
 1 Eigenvalue decomposition with balancing
–1 Eigenvalue decomposition with no balancing
 2 Schur decomposition with balancing. (default)
–2 Schur decomposition with no balancing.

Output arguments:

`k` H_2 optimal controller

`g` closed-loop system with optimal controller

`norms` norms of four different quantities, full information control cost (FI), output estimation cost (OEF), disturbance feedforward cost (DFL) and full control cost (FC), norms = [FI OEF DFL FC];

`kfi` full information/state feedback control law

`gfi` full information/state feedback closed-loop system

hamx	H_2 Hamiltonian matrix
hamy	H_2 Hamiltonian matrix

The equations and corresponding nomenclature are taken from the Doyle, *et al.*, 1989, reference. The full information cost is given by the equation$(\mathrm{trace}(B_1' X_2 B_1))^{\frac{1}{2}}$. The output estimation cost is given by$(\mathrm{trace}(F_2 Y_2 F_2'))^{\frac{1}{2}}$, where $F_2 =: -(B_2' X_2 + D_{12}' C_1)$. The disturbance feedforward cost is $(\mathrm{trace}(L_2' X_2 L_2))^{\frac{1}{2}}$, where L_2 is defined by$-(Y_2 C_2' + B_1 D_{21}')$ and the full control cost is given by $(\mathrm{trace}(C_1 Y_2 C_1'))^{\frac{1}{2}}$. X_2 and Y_2 are the solutions to the X and Y Riccati equations, respectively.

The H_2 solution provides an upper bound on γ for use in the hinfsyn program.

Examples

Design an H_2 optimal controller for a system matrix, himat_icn, with two sensor measurements (nmeas), two error signals, two actuator inputs (ncont), and eight states. himat_icn differs from the SYSTEM interconnection structure himat_ic by the fact that the D_{11} term of himat_ic is set to be zero. The Schur decompostion method, ricmethd = 2, will be used for solution of the Riccati equations. The program outputs the minimum eigenvalue of X_2 and Y_2 during the computation.

```
nmeas = 2;
ncont = 2;
ricmethd = 2;
minfo(himat_icn)
system:8 states6 outputs6 inputs
[k,g] = h2syn(himat_icn,nmeas,ncont,ricmethd);
minimum eigenvalue of X2: 2.260000e-02
minimum eigenvalue of Y2: 2.251670e-02
```

The H_∞ and H_2 norm of the resulting closed-loop system g can be calculated via the commands hinfnorm and h2norm.

```
hinfnorm(g)
norm between 2.787 and 2.79
achieved near 29.9
h2norm(g)
1.594e+01
```

Algorithm

h2syn is an M-file in μ-Tools that uses the formulae described in the Doyle, *et al.*, 1989, reference for solution to the optimal H_2 control design problem. A Hamiltonian is formed and solved via a Riccati equation (ric_eig and ric_schr). The D matrix associated with the input disturbances and output errors is *restricted* to be zero.

Subroutines called. ric_eig, ric_schr, csord, and cgivens.

Reference

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

Glover, K., and J.C. Doyle, "State-space formulae for all stabilizing controllers that satisfy an H_∞ norm bound and relations to risk sensitivity," *Systems and Control Letters*, 1988. vol. 11, pp. 167–172, August 1989.

See Also

hinfsyn, hinffi, h2norm, hinfnorm, ric_eig, ric_schr

Purpose

Compute H_∞ full-information controller for a SYSTEM interconnection matrix

Syntax

```
[k,g,gfin,ax,hamx] =
  hinffi(p,ncon,gmin,gmax,tol,ricmethd,epr,epp)
```

Description

`hinffi` calculates an H_∞ full information controller that achieves the infinity norm `gfin` for the interconnection structure `p`. The controller, `k`, stablizes the SYSTEM matrix `p` and is constant gain. The system `p` is partitioned

$$\mathbf{p} = \left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \end{array}\right]$$

where B_1 are the disturbance inputs, B_2 are the control inputs, and C_1 are the errors to be kept small. B_2 has the column size `ncon`. Within the `hinffi` program, the SYSTEM matrix `p` is augmented with state and disturbance measurements; i.e., the identity matrix with size equal to the number of states of `p` and the identity matrix with size equal to the number of disturbances. Be careful when closing the loop with the full information controller since the extra measurements are only augmented *inside* the command `hinffi`. The internal system used for control design is

$$\left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ \begin{bmatrix} I \\ 0 \end{bmatrix} & \begin{bmatrix} 0 \\ I \end{bmatrix} & \begin{bmatrix} 0 \\ 0 \end{bmatrix} \end{array}\right]$$

The controller is returned in `k` and the closed-loop system is returned in `g`. The program provides a γ iteration using the bisection method. Given a high and low value of γ, `gmax` and `gmin`, the bisection method is used to iterate on the value of γ in an effort to approach the optimal full information H_∞ control design. If `gmax` = `gmin`, only one γ value is tested. The stopping criteria for the bisection algorithm requires the relative difference between the last γ value that failed and the last γ value that passed be less than `tol`. You can select either the eigenvalue or Schur method with or without balancing for solving the Riccati equations. The eigenvalue method is faster but can have numerical problems, while the Schur method is slower but generally more reliable.

The algorithm employed requires tests to determine whether a solution exists for a given γ value. `epr` is used as a measure of when the Hamiltonian matrix has imaginary eigenvalues and `epp` is used to determine whether the Riccati solution is positive semi-definite. The selection of `epr` and `epp` should be based on your knowledge of the numerical conditioning of the interconnection structure `p`. The conditions checked for the existence of a solution are

- H Hamiltonian matrix must have no $j\omega$-axis eigenvalues
- the stabilizing solution, $X\infty$, of the associated Riccati equation must exist, and be must be positive, semi-definite.

Input arguments:

`p`	SYSTEM interconnection structure matrix
`ncon`	number of controller outputs
`gmin`	lower bound on γ
`gmax`	upper bound on γ
`tol`	relative difference between final γ values, iteration stopping criteria
`ricmethd`	1 Eigenvalue decomposition with balancing. –1 Eigenvalue decomposition with no balancing 2 Schur decomposition with balancing (default) –2 Schur decomposition with no balancing
`epr`	measure of when a real eigenvalue of the Hamiltonian matrix is zero (default `epr` = 1e–10, optional)
`epp`	positive definite determination of the $X\infty$ solution (default `epp` = 1e–6, optional)

Output arguments:

`k`	$H\infty$ full information controller
`g`	closed-loop system with $H\infty$ full information controller
`gfin`	final γ achieved
`ax`	Riccati solution as a VARYING matrix with independent variable γ
`hamx`	Hamiltonian matrix as a VARYING matrix with independent variable γ

Note that the outputs `ax`, `ay`, `hamx`, and `hamy` may correspond to scaled or balanced data. The following assumptions are made in the implementation of the `hinfsyn` algorithm and must be satisfied.

(i) (A,B_2) is stabilizable

(ii) D_{12} is full column rank

(iii) $\begin{bmatrix} A - j\omega I & B_2 \\ C_1 & D_{12} \end{bmatrix}$ has full column rank for all ω.

where ω denotes the frequency variable.

On return, there must be no $j\omega$-axis eigenvalues associated with the H Hamiltonian matrices and the eigenvalues of the Riccati solution, X_∞, must all be ≥ 0, for the closed-loop system to be stable and to have an H_∞ `norm` less than γ. The bisection algorithm iterates on the value of γ to approach the optimal H_∞ full information controller.

The `hinffi` program outputs several variables, which can be checked to ensure that the above conditions are being met. For each γ value the minimum magnitude, real part of the eigenvalues of the H Hamiltonian matrices is displayed along with the minimum eigenvalue of X_∞, which is the solution to the Riccati equation. A # sign is placed to the right of the condition that failed in the printout.

Examples

Given an interconnection structure `sys` with one control input, it is desired to synthesize a full information controller. The upper bound on γ is 1.0 and the lower bound is specified as 0.1. A tolerance of 0.02 is selected for the stopping condition for the γ iteration and the Schur method is used to solve the Riccati equations. The command `hinffi` outputs the display shown for each value of γ. The final γ value achieved is 0.2547.

```
  ncont = 1;% number of control inputs
  gmin = .1;% minimum gamma value to be tested
  gmax = 1;% maximum gamma value to be tested
  tol = .02;% tolerance on the gamma stopping value
  ricmethd = 2;% Riccati solution via the Schur method
  seesys(p)% plant interconnection structure
p =

  3    1  |   4
  0    0  |   1
  -------------
  1    0  |   0

  [k,g,gf,ax,hx] = hinffi(p,1,.1,1,.02,2);
  Test bounds:0.1000<    gamma<= 1.0000

  gamma       ham_eig       x_eig         pass/fail
  1.000       4.90e+00      5.27e-01      pass
  0.550       4.66e+00      6.03e-01      pass
  0.325       3.94e+00      1.06e+00      pass
  0.213       1.69e+00     -7.63e-01#     fail
  0.269       3.34e+00      2.94e+00      pass
  0.241       2.78e+00     -4.55e+00#     fail
  0.255       3.10e+00      1.04e+01      pass

  Gamma value achieved:0.2547
```

Algorithm

`hinffi` uses the formulas similar to the ones described in the Glover and Doyle, 1988 paper for solution to the H_∞ control design problem. See the `hinfsyn` command for more information.

Subroutines called. `hinffi_t`, `hinffi_p`, `hinffi_c`, and `hinffi_g`
`hinffi_g` calls: `ric_eig`, `ric_schr`, `csord`, and `cgivens`

Reference

Doyle, J.C., K. Glover, P. Khargonekar, and B. Francis, “State-space solutions to standard H_2 and H_∞ control problems,” *IEEE Transactions on Automatic Control*, vol. 34, no. 8, pp. 831–847, August 1989.

Glover, K., and J.C. Doyle, “State-space formulae for all stabilizing controllers that satisfy an H_∞ `norm` bound and relations to risk sensitivity,” *Systems and Control Letters*, vol. 11, pp. 167–172, 1988.

See Also

`h2syn`, `h2norm`, `hinfsyn`, `hinfsyne`, `hinfnorm`, `ric_eig`, `ric_schr`

Purpose

Compute an H_∞ controller for a SYSTEM interconnection matrix

Syntax

```
[k,g,gfin,ax,ay,hamx,hamy] =
  hinfsyn(p,nmeas,ncon,gmin,gmax,tol,ricmethd,epr,epp)
```

Description

hinfsyn calculates an H_∞ controller, which achieves the infinity norm gfin for the interconnection structure p. The controller, k, stabilizes the SYSTEM matrix p and has the same number of states as p. The SYSTEM p is partitioned

$$\mathrm{p} = \left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{array}\right]$$

where $B_1$are the disturbance inputs, B_2 are the control inputs, C_1 are the errors to be kept small, and C_2 are the output measurements provided to the controller. B_2 has column size (ncon) and C_2 has row size (nmeas).

The closed-loop system is returned in g. The program provides a γ iteration using the bisection method. Given a high and low value of γ, gmax and gmin, the bisection method is used to iterate on the value of γ in an effort to approach the optimal H_∞ control design. If the value of gmax is equal to gmin, only one γ value is tested. The stopping criteria for the bisection algorithm requires the relative difference between the last γ value that failed and the last γ value that passed be less than tol. You can select either the eigenvalue or Schur method the Riccati equations. The eigenvalue method is faster but can have numerical problems, while the Schur method is slower but generally more reliable.

The algorithm employed requires tests to determine whether a solution exists for a given γ value. epr is used as a measure of when the Hamiltonian matrix has imaginary eigenvalues and epp is used to determine whether the Riccati solutions are positive semi-definite. The conditions checked for the existence of a solution are:

- H and J Hamiltionian matrices (which are formed from the state-space data of P and the γ level) must have no imaginary-axis eigenvalues.
- the stabilizing Ricatti solutions X_∞ and Y_∞ associated with the Hamiltionian matrices must exist and be positive, semi-definite.
- spectral radius of (X_∞, Y_∞) must be less than or equal to γ^2.

The selection of `epr` and `epp` should be based on your knowledge of the numerical conditioning of the interconnection structure `p`. The following assumptions are made in the implementation of the `hinfsyn` algorithm and *must* be satisfied.

(i) (A,B_2) is stabilizable and (C_2,A) detectable.

(ii) D_{12} and D_{21} have full rank.

(iii) $\begin{bmatrix} A - j\omega I & B_2 \\ C_1 & D_{12} \end{bmatrix}$ has full column rank for all $\omega \in \mathbf{R}$.

(iv) $\begin{bmatrix} A - j\omega I & B_1 \\ C_2 & D_{21} \end{bmatrix}$ has full row rank for all $\omega \in \mathbf{R}$.

Inputs arguments:

`p`	SYSTEM interconnection structure matrix
`nmeas`	number of measurements output to controller
`ncon`	number of control inputs
`gmin`	lower bound on γ
`gmax`	upper bound on γ
`tol`	relative difference between final γ values
`ricmethod`	1 Eigenvalue decomposition with balancing 2 Schur decomposition with balancing (default) –2 Schur decomposition with no balancing
`epr`	measure of when a real part of an eigenvalue of the Hamiltonian matrix is zero (default `epr` = 1e–10)
`epp`	positive definite determination of the X_∞ and Y_∞ solution (default `epp` = 1e–6)

Output arguments:

`k`	H_∞ (sub) optimal controller
`g`	closed-loop system with H_∞ controller
`gfin`	final γ achieved
`ax`	X_∞ Riccati solution as a VARYING matrix with independent variable γ
`ay`	Y_∞ Riccati solution as a VARYING matrix with independent variable γ
`hamx`	H_∞ Hamiltonian matrix as a VARYING matrix with independent variable γ
`hamy`	J_∞ Hamiltonian matrix as a VARYING matrix with independent variable γ

Note that the outputs `ax`, `ay`, `hamx`, and `hamy` correspond to scaled or balanced data.

The `hinfsyn` program displays several variables, which can be checked to ensure that the above conditions are being satisfied. For each γ value being tested, the minimum magnitude, real part of the eigenvalues of the X and Y Hamiltonian matrices are displayed along with the minimum eigenvalue of X_∞ and Y_∞, which are the solutions to the X and Y Riccati equations, respectively. The maximum eigenvalue of $X_\infty Y_\infty$, scaled by γ^{-2},is also displayed. A # sign is placed to the right of the condition that failed in the printout.

Note When a Hamiltonian has repeated eigenvalues, solving the Riccati equation via the eigenvalue method (`ric_eig`) may have problems. This is due to the MATLAB command `eig` incorrectly selecting the eigenvectors associated with the repeated roots.

Examples

This example is taken from the “HIMAT Robust Performance Design Example” section in Chapter 7. himat_ic contains the open-loop interconnection structure. Design an H_∞ (sub)optimal controller for the SYSTEM matrix, himat_ic, with two sensor measurements, two error signals, two actuator inputs, two disturbances, and eight states. The range of γ is selected to be between 1.0 and 10.0 with a tolerance, tol, on the relative closeness of the final γ solution of 0.1. The Schur decompostion method, ric_schr, is used for solution of the Riccati equations. The program outputs at each iteration the current γ value being tested, and eigenvalue information about the H and J Hamiltonian matrices and X_∞ and Y_∞ Riccati solutions. At the end of each iteration a (p) denoting the tested γ value passed or an (f) denoting a failure is displayed. Upon finishing, hinfsyn prints out the γ value achieved.

```
nmeas = 2;% number of sensor measurements
ncont = 2;% number of control inputs
gmin = 1;% minimum gamma value to be tested
gmax = 10;% maximum gamma value to be tested
tol = .1;% tolerance on the gamma stopping value
ric = 2;% Riccati equation solved via the Schur method
minfo(himat_ic)% SYSTEM interconnection structure
system: 8 states6 outputs6 inputs

[k,g] = hinfsyn(himat_ic,nmeas,ncon,gmin,gmax,tol,ric);
Test bounds: 1.0000 < gamma <=10.0000

gamma   hamx_eig   xinf_eig   hamy_eig  yinf_eig   nrho_xy  p/f
10.000  2.3e-02    2.1e-10    2.3e-02  -3.7e-11    0.022     p
5.500   2.3e-02    2.1e-10    2.3e-02  -0.0e+00    0.075     p
3.250   2.3e-02    2.2e-10    2.3e-02  -0.0e+00    0.222     p
2.125   2.3e-02    2.2e-10    2.3e-02  -0.0e+00    0.564     p
1.562   2.3e-02    2.4e-10    2.3e-02  -0.0e+00    1.198#    f
1.844   2.3e-02    2.3e-10    2.3e-02  -0.0e+00    0.789     p
1.703   2.3e-02    2.3e-10    2.3e-02  -2.1e-11    0.959     p
1.633   2.3e-02    2.3e-10    2.3e-02  -0.0e+00    1.068#    f

Gamma value achieved:1.7031
```

Algorithm

hinfsyn uses the formulae described in the Glover and Doyle, 1988, paper for solution to the optimal H_∞ control design problem. There are a number of research issues that need to be addressed for the "best" solution of the Riccati equations but only two of the standard methods are included.

Subroutines called. hinf_st, hinf_sp, hinf_c, and hinf_gam
hinf_gam calls: ric_eig, ric_schr, csord, cgivens

Reference

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

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

See Also

hinffi, hinfnorm, hinfsyne, h2syn, h2norm, ric_eig, ric_schr

hinfsyne

Purpose

Compute an $H\infty$ controller for a SYSTEM interconnection matrix that minimizes the entropy integral at a specific frequency

Syntax

```
[k,g,gfin,ax,ay,hamx,hamy] = hinfsyne(p,nmeas,ncon,...
   gmin,gmax,tol,s0,quiet,ricmethd,epr,epp)
```

Description

`hinfsyne` is a variation of `hinfsyn` and calculates an H_∞ controller that achieves the infinity norm `gfin` for the interconnection structure `p`. The controller, `k`, stablizes the SYSTEM matrix `p` and has the same number of states as `p`. Of the controllers achieving this `norm` bound, `k` is chosen to minimize an entropy integral relating to the point `s0`; i.e.,

$$I = -\frac{\mathbf{gfin}^2}{2\pi}\int_{-\infty}^{\infty}\ln\left|\det(I-\gamma^{-2}\mathbf{g}(j\omega)'\mathbf{g}(j\omega))\right|\left[\frac{\mathbf{s0}^2}{\mathbf{s0}^2+\omega^2}\right]d\omega$$

where `g` is the closed-loop transfer function. In addition, the amount of printing on the screen can be controlled.

Input arguments:

`p`	SYSTEM interconnection structure matrix
`nmeas`	number of measurements output to controller
`ncon`	number of control inputs
`gmin`	lower bound on γ
`gmax`	upper bound on γ
`tol`	relative difference between final γ values
`s0`	point at which entropy is evaluated (default ∞)
`quiet`	controls printing on the screen 1 no printing 1 header not printed –1 full printing (default)
`ricmethod`	1 Eigenvalue decomposition with balancing –1 Eigenvalue decomposition with no balancing 2 Schur decomposition with balancing (default) –2 Schur decomposition with no balancing

`epr` measure of when a real part of an eigenvalue of the Hamiltonian matrix is zero (default `epr` = 1e–10)

`epp` positive definite determination of the X_∞ and Y_∞ solution (default `epp` = 1e–6)

Output arguments:

`k` H_∞ (sub) optimal controller

`g` closed-loop system with H_∞ controller

`gfin` final γ value achieved

`ax` X_∞ Riccati solution as a VARYING matrix with independent variable γ

`ay` Y_∞ Riccati solution as a VARYING matrix with independent variable γ

`hamx` H_∞ Hamiltonian matrix as a VARYING matrix with independent variable γ

`hamy` J_∞ Hamiltonian matrix as a VARYING matrix with independent variable γ

Note that the outputs `ax`, `ay`, `hamx`, and `hamy` correspond to scaled or balanced data.

The `hinfsyne` program outputs several variables, which can be checked to ensure that the above conditions are being met. For each γ value the minimum magnitude, real part of the eigenvalues of the X Hamiltonian matrices is displayed along with the minimum eigenvalue of X_∞, which is the solution to the X Riccati equation. A # sign is placed to the right of the condition that failed in the printout. This additional information can aid you in the control design process.

Algorithm

hinfsyne uses the formulas similar to the ones described in the Glover and Doyle paper for solution to the H_∞ control design problem. See the hinfsyn command for more information.

Subroutines called. hinf_st, hinf_gam, hinfe_c, hinf_gam calls: ric_eig, ric_schr, csord, and cgivens

Reference

Doyle, J.C., K. Glover, P. Khargonekar, and B. Francis, “State-space solutions to standard H_2 and H_∞ control problems,” *IEEE Transactions on Automatic Control*, vol. 34, no. 8, pp. 831–847, August 1989.

Glover, K., and J.C. Doyle, “State-space formulae for all stabilizing controllers that satisfy an H_∞norm bound and relations to risk sensitivity,” *Systems and Control Letters*, vol. 11, pp. 167–172, 1988.

See Also

dhfsyn, hinfsyn, hinffi, hinfnorm, h2syn, h2norm, ric_eig, ric_schr, sdhfsyn

Purpose

Compare the independent variable data of two VARYING matrices

Syntax

```
code = indvcmp(mat1,mat2,errcrit)
indvcmp(mat1,mat2,errcrit)
```

Description

indvcmp compares the data for two VARYING matrices. If the two sets of independent variables are within a specified tolerance of one another, then the VARYING matrices are assumed to have identical independent variables, and the VARYING matrices can be combined (i.e., added, subtracted, multiplied, etc.). The results are displayed if an output argument is not provided.

Input arguments:

mat1, mat2=	matrices to be compared
errcrit=	1 x 2 optional matrix containing the relative error and absolute error bounds. The relative error is used to test the error in independent variables whose magnitude is greater than 1e-9, while the absolute error bound used for smaller independent variable values. Default values are 1e-6, and 1e-13, respectively.

Output arguments:

code=0	independent variable data is different
code=1	independent variable data is identical
code=2	different number of points
code=3	at least one matrix isn't a VARYING matrix

Examples

Compare the two frequency response matrices, mat has its independent variable at 0.01 and 0.1 and mat2 has its independent variable at 0.011 and 0.1. Given the default comparison criteria, the independent variable is different. Changing the tolerance leads to the command checking different indvcmp variations in the independent variable.

```
see(mat)

  2 rows   3 columns
  indep variable   0.01
  1   2   3
  4   5   6
  indep variable   0.1
   7    8    9
  10   11   12


  see(mat2)

  2 rows   3 columns
  indep variable   0.011
  10   20   30
  40   50   60
  indep variable   0.1
   70    80    90
  100   110   120

  indvcmp(mat,mat2)

  varying data is DIFFERENT
  code = indvcmp(mat,mat2)

  code =
  0
```

Changing the relative and absolute error bounds in indvcmp leads these two independent variables to be deemed the same.

```
indvcmp(mat,mat2,[1 1])
varying data is the same
```

Algorithm

indvcmp uses standard MATLAB commands.

See Also

getiv, sortiv, vunpck, xtract, xtracti

Purpose

Add and subtract CONSTANT, SYSTEM, and VARYING matrices

Syntax

```
out = madd(mat1,mat2,,matN)
out = msub(mat1,mat2,,matN)
```

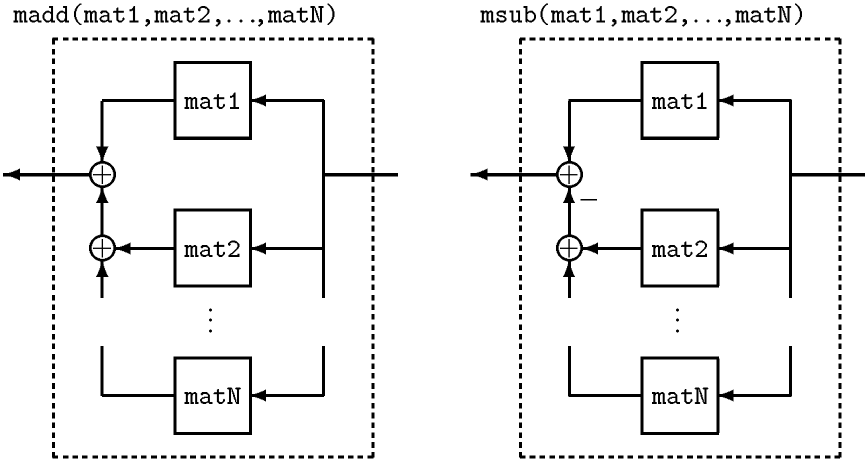
Description

madd (msub) allows the addition or subtraction of matrices, regardless of their type, as long as their dimensions are compatible. CONSTANT, SYSTEM, and VARYING matrices can be added to or subtracted from one another based on the following table.

mat1	mat2	**CONSTANT**	**SYSTEM**	**VARYING**
CONSTANT		yes	yes	yes
SYSTEM		yes	yes	no
VARYING		yes	no	yes

For compatibility, the number of rows and columns of mat1 must equal the number of rows and columns of mat2. In the case of SYSTEM matrices, the number of inputs and outputs of mat1 must equal the number of inputs and outputs of mat2. The same is true for VARYING matrices and in addition, the independent variables of the VARYING matrices must be identical. Up to nine matrices of compatible dimension can be added or subtracted by including them as input arguments.

Pictorial Representationof Functions



Examples

Create two SYSTEM matrices p and p1 and a CONSTANT matrix.

```
a = -10;b = 3;c = 10;d = 0;
p = pck(a,b,c,d);
minfo(p)
system:                1 states1 outputs1 inputs
a1 = -2; b1 = 3; c1 = 1; d1 = .1;
p1 = pck(a1,b1,c1,d1);
minfo(p1)
seesys(p,'3.2g')

-10  |  3
-----|----
 10  |  0

seesys(p1,'3.2g')

-2  |  3
----|----
 1  |  .1
```

Adding two SYSTEM matrices returns a SYSTEM matrix with the same number of inputs and outputs as p and p1.

```
out = madd(p,p1);
seesys(out,'3.2g')

-10   0  |   3
0    -2  |   3
---------|------
10    1  |   .1

minfo(out)
system:    2 states    1 outputs   1 inputs
```

Adding a SYSTEM matrix and a CONSTANT matrix returns a SYSTEM matrix with the CONSTANT term added to the D-term of the state-space system.

```
out = madd(p,a);
seesys(out,'%3.2g')

-10   |    3
------|-----
 10   |  -10
```

Subtracting a SYSTEM matrix and a CONSTANT returns a SYSTEM matrix with the CONSTANT term added to the D-term of the state-space system.

```
out = msub(p,a);
seesys(out,'%3.2g')

-10   |    3
------|-----
 10   |  -10
```

Algorithm

madd and msub call the MATLAB + and – commands consistent with the type of matrices.

See Also

+, -, mmult, mscl

Purpose

Matrix assignment for VARYING and SYSTEM matrices

Syntax

```
out = massign(matin,rowindex,colindex,data)
```

Description

Performs a matrix assignment like operation on VARYING and SYSTEM matrices. It is functionally equivalent to

```
matin(rowindex,colindex) = data
```

where rowindex and colindex are vectors specifying the rows and columns (or outputs and inputs if matin is a SYSTEM) to be changed.

data must either be a constant or of the same type as matin. The dimensions of data must be consistent with the lengths of rowindex and colindex.

Note When applied to a SYSTEM, the result will almost always be nonminimal.

Examples

In the first example a VARYING matrix with two independent variables is formed with identical (and obvious) data for each matrix.

```
tl = [11,12,13,14;
21,22,23,24;
31,32,33,34;
41,42,43,44];

vmat = vpck([tl;tl],[0.1,0.2]);
```

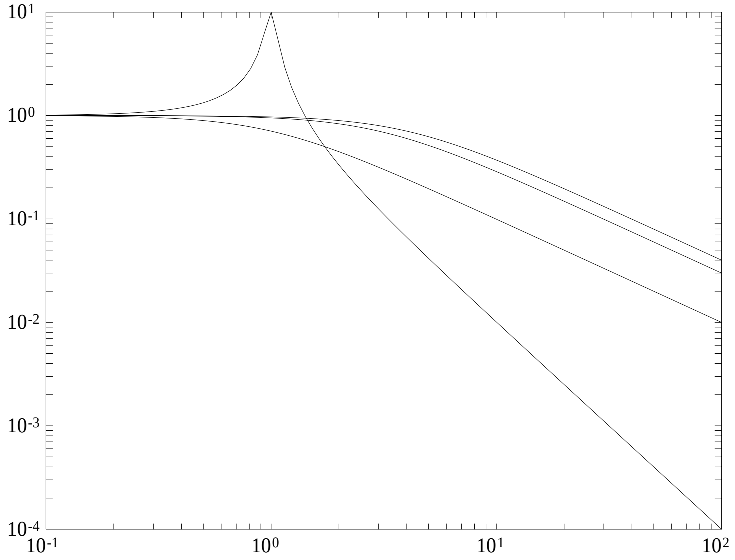
Now make a 2 × 2 data matrix and insert it into the VARYING matrix. Changing the order of the row and column indices has the effect of permuting the result. This is identical to the constant matrix case.

```
ri = [1,3];
ci = [4,2];
data = [0.001, 0.002; 0.003, 0.004];
vmatl = massign(vmat,ri,ci,data);
see(vmatl)
4 rows                  4 columns
```

```
iv = 0.1

11.0000    0.0020    13.0000    0.0010
21.0000   22.0000    23.0000   24.0000
31.0000    0.0040    33.0000    0.0030
41.0000   42.0000    43.0000   44.0000

iv = 0.2

11.0000    0.0020    13.0000    0.0010
21.0000   22.0000    23.0000   24.0000
31.0000    0.0040    33.0000    0.0030
41.0000   42.0000    43.0000   44.0000
```

In the following example part of a system is replaced by the massign function. The initial system is a diagonal system of first order lags. The 2,2 element is replaced by a lightly damped system and the result plotted. Note that the modified system is no longer minimal.

```
a = diag([1,2,-3,-4]);
b = eye(4);
c = diag([1,2,3,4]);
sys = pck(a,b,c);
subsys = nd2sys(1,[1,0.1,1]);
sys = massign(sys,2,2,subsys);
minfo(sys)

system: 10 states  4 outputs   4 inputs

omega = logspace(-1,2,100);
sys_g = frsp(sys,omega);
vplot('liv,lm',sys_g,'-')
```



See Also `sel`

Purpose

Generate SYSTEM representations of a Bessel, Butterworth, Chebyshev or RC filter

Syntax

```
sys = mfilter(fc,ord,type,psbndr)
```

Description

Calculates, as a SYSTEM, a single-input, single-output analog low-pass filter. These filters are often encountered in experimental arrangements and must be accounted for in experimental data processing and control design. For more sophisticated filters see the Signal Processing Toolbox functions.

The cutoff frequency (Hertz) is `fc` and the filter order is `ord`. The string variable, `type`, specifies the type of filter and can be one of the following.

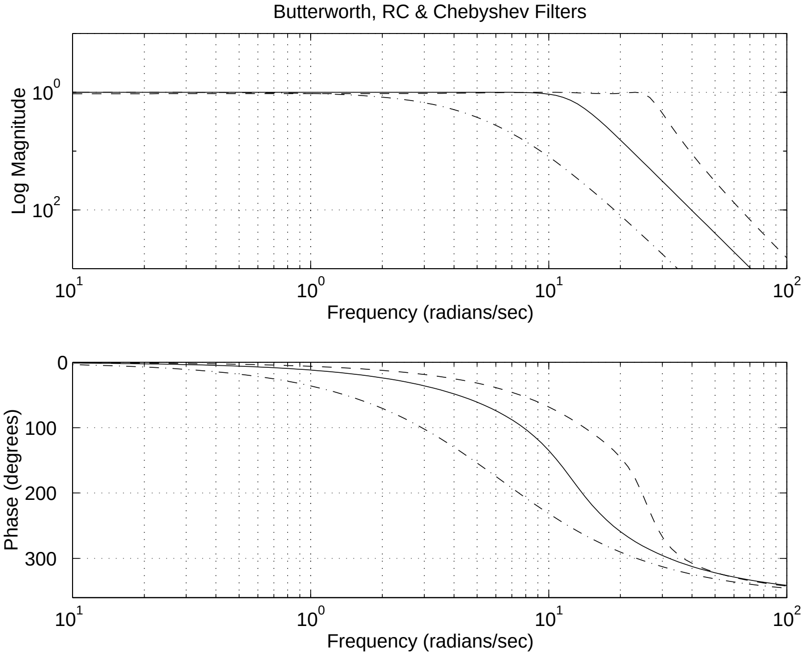
`butterw`	Butterworth
`cheby`	Chebyshev
`bessel`	Bessel
`rc`	series of resistor/capacitor filters

The dc gain of each filter (except even order Chebyshev) is set to unity. The argument `psbndr` specifies the Chebyshev passband ripple (in dB). At the cutoff frequency, the magnitude is `-psbndr` dB. For even order Chebyshev filters the DC gain is also `-psbndr` dB.

The Bessel filters are calculated using the recursive polynomial formula. This is poorly conditioned for high order filters (order > 8).

Examples

```
butw = mfilter(2,4,'butterw');
cheb = mfilter(4,4,'cheby',0.5);
rc = mfilter(1,4,'rc');
omega = logspace(-1,2,100);
butw_g = frsp(butw,omega);
cheb_g = frsp(cheb,omega);
rc_g = frsp(rc,omega);
vplot('bode_gl',[.1 100 .001 10],[.1 100 -180 180],...
   butw_g,'-',cheb_g,'-',rc_g,'-.')
subplot(211),title('Butterworth, RC Chebyshev Filters')
```


Butterworth, RC & Chebyshev Filters
Log Magnitude
10^{0}
10^{2}
10^{1}
10^{0}
10^{1}
10^{2}
Frequency (radians/sec)
Phase (degrees)
0
100
200
300
10^{1}
10^{0}
10^{1}
10^{2}
Frequency (radians/sec)

Purpose

Provide matrix information

Syntax

```
minfo(matin)
[systype,rowdata,coldata,pointdata] = minfo(matin)
```

Description

minfo returns information about the data type, and size of the matin. With no output assignment, minfo returns text output to the screen. The information is determined from the data structure as defined in the “The Data Structures” section in Chapter 2.

With output arguments, minfo returns four arguments. The first argument, systype, is a string variable that can take one of four values. The interpretation of the three additional output arguments is based on the variable systype.

systype == 'vary'	matin is a VARYING matrix. pointdata tells how many independent variable values there are, rowdata is the row dimension of a matrix, and coldata is the column dimension
systype == 'syst'	matin is a SYSTEM matrix. pointdata is the number of states, rowdata is the number of outputs, and coldata is the number of inputs.
systype == 'cons'	matin is a regular MATLAB matrix, pointdata is set to NaN, rowdata is the number of rows, and coldata is the number of columns.
systype == 'empt'	matin is an empty MATLAB matrix; also pointdata, rowdata, and coldata are set to empty.

Examples

minfo identifies the type of matrix being manipulated. Compare the displays for a CONSTANT, SYSTEM, and VARYING matrix.

```
a = rand(2,2);b = rand(2,3);c = rand(1,2);
minfo(a)
constant:              2 rows  2 cols
sys = pck(a,b,c);
minfo(sys)
system:                2 states1 outputs3 inputs
sys_g = frsp(sys,[.1 .5 .9 1.4]);
minfo(sysg)
varying:               4 pts   1 rows 3 cols
4 pts between 0.1 and 0.4
sys = sysrand(2,3,1);
[mtype,mrows,mcols,mnum] = minfo(sys);
mtype
mtype =
syst
[mrows,mcols,mnum]
ans =
   1           3           2
```

See Also

pck, pss2sys, sys2pss, unpck, vpck, vunpck

Purpose

Calculate the inverse of CONSTANT, SYSTEM, and VARYING matrices. (The matrices must be square.)

Syntax

```
out = minv(mat)
out = vinv(mat)
```

Description

`minv` calculates the inverse of the input matrix. For VARYING matrices, `minv` returns a VARYING matrix with the inverse of each independent variable matrix. For SYSTEM matrices, the inverse is defined as

$$\mathbf{mat} = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right], \qquad \mathbf{out} = (\mathbf{mat})^{-1} = \left[\begin{array}{c|c} A - BD^{-1}C & -BD^{-1}C \\ \hline D^{-1}C & D^{-1} \end{array}\right]$$

`vinv` is the same command as `minv`, but works only on CONSTANT and VARYING matrices.

Examples

Determine the inverse of a SYSTEM matrix and a VARYING matrix.

```
sys = pck(-2,3,1,.1);
omega = [1 10];
sysg = frsp(sys,omega);
sysi = minv(sys);
seesys(sysi)

-3.2e+01  |   -3.0e+01
----------|-----------
1.0e+01   |    1.0e+01

sysig = frsp(sysi,omega);
see(sbs(sysg,sysig,minv(sysig)))
1 row   3 columns
iv =
1.3000 - 0.6000i   0.6341 + 0.2927i   1.3000 - 0.6000i
iv = 10
0.1577 - 0.2885i   1.4591 + 2.6690i   0.1577 - 0.2885i
```

Algorithm `minv` and `vinv` call the MATLAB `inv` command.

See Also `inv`, `vdet`

Purpose

Multiply CONSTANT, SYSTEM, and VARYING matrices

Syntax

```
out = mmult(mat1,mat2,,matN)
```

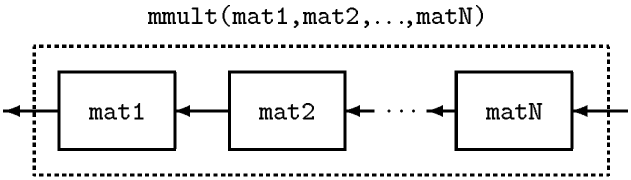
Description

`mmult` allows the multiplication of matrices, `mat1` and `mat2` regardless of their type, provided their dimensions are compatible. CONSTANT, SYSTEM and VARYING matrices can be multiplied by one another based on the following table.

mat1	mat2	CONSTANT	SYSTEM	VARYING
CONSTANT		yes	yes	yes
SYSTEM		yes	yes	no
VARYING		yes	no	yes

For compatibility, the number of columns of `mat1` must equal the number of rows of `mat2`. In the case of SYSTEM matrices, the number of inputs of `mat1` must equal the number of outputs of `mat2`. (An alternative term for the multiplication of two SYSTEM matrices is cascade.) Similarly restrictions apply for VARYING matrices. Up to nine matrices of compatible dimension can be multiplied via the same command by including them as input arguments.

Pictorial Representation of Function



Examples

The multiplication (cascade) of two SYSTEM matrices is shown below.

```
seesys(p1)

-1.0e+01  |  1.0e+00
----------|----------
1.0e+01   |  1.0e+00

minfo(p1)
system:   1 states   1 outputs   1 inputs
seesys(p2)

-3.0e+00  |  2.0e+00
----------|----------
 4.0e+00  |  1.0e-01

minfo(p2)
system:   1 states   1 outputs   1 inputs
out = mmult(p1,p2)
seesys(out,%5.2g')

-10   4  |  0.1
  0  -3  |  2
---------|------
 10   0  |  0
```

Algorithm

mmult uses the MATLAB " * " command when the multiplication does not involve two SYSTEM matrices. The equation for the multiplication of two subsystems is given by

$$\texttt{sys1} = \left[\begin{array}{c|c} A_1 & B_1 \\ \hline C_1 & D_1 \end{array}\right], \qquad \texttt{sys2} = \left[\begin{array}{c|c} A_2 & B_2 \\ \hline C_2 & D_2 \end{array}\right],$$

$$\rightarrow \texttt{mmult(sys1,sys2)} = \left[\begin{array}{cc|c} A_1 & B_1C_2 & B_1D_2 \\ 0 & A_2 & B_2 \\ \hline C_1 & D_1C_2 & D_1D_2 \end{array}\right]$$

See Also

*, abv, madd, mscl, msub

mprintf

Purpose

Format output of matrix to screen

Syntax

```
mprintf(matin,'format','end of line characters')
```

Description

mprintf displays a matrix in formatted form. The optional 'format' specifies the format exactly as in the MATLAB function sprintf. If no 'format' is specified the default is '%.1e'. This routine is primarily for use in seesys and does not work well for f format when the minimum field width is too small. There is no input checking, so you can wreak havoc if you use mprintf incorrectly. See sprintf for more details. The optional 'end of line characters' is exactly what it says. The default is the newline C escape sequence (\n). To get no newline at the end of each line use mprintf(matin,'format',[]).

Examples

The mprintf command displays any type of matrix. An example of its use for SYSTEM and VARYING matrices follows.

```
mprintf(m)

1.7e+01   8.7e+00   -6.4e+00   -1.3e+01
5.9e-01  -1.4e+01    5.8e+00   -1.3e+01
1.8e+01  -7.0e+00   -3.6e+00    9.8e+00
2.6e+00   1.2e+01   -1.4e+00   -4.5e-01

mprintf(m,'%6.2f ')

16.96    8.72  -6.39   -13.49
0.59   -14.46   5.77   -12.70
17.97   -7.01  -3.60     9.85
2.64    12.46  -1.36    -0.45

mprintf (m, '%3.f ')

17     9    -6     -13
 1   -14     6     -13
18   -7     -4      10
 3    12    -1      -0
```

See Also

fprintf, rifd, see, seesys, sprintf

Purpose

Interactive *D*-scaling rational fit routine used in μ-synthesis

Syntax

```
[dsysL,dsysR] = msf(Mg,bnds,dvec,sens,blk)
[dsysL,dsysR] = msfbatch(Mg,bnds,dvec,sens,blk,maxord)
```

Description

`msf` fits the block diagonal, frequency-dependent matrices $D_L(\omega)$ and $D_R(\omega)$ (contained in the VARYING matrix `dvec`, with block structure implied by the entries of `blk`) with rational, stable, minimum-phase $\hat{D}_L(s)$ and $\hat{D}_R(s)$ such that

$$\max\bar{\sigma}[D_L(\omega)M(j\omega)D_R^{-1}(\omega)] \approx \left\|\hat{D}_L M \hat{D}_R^{-1}\right\|_\infty$$

`msf` returns the stable, minimum phase system matrices `dsysL` and `dsysR`.

Note Typically, there is no need to call `msf` directly. The standard use of `msf` is a subroutine within μ-synthesis. The programs `dkit` and/or `dkitgui` are fully functional μ-synthesis routines.

Input arguments:

`Mg` is the frequency response upon which the μ calculation was performed.

`bnds` is the upper bound from the μ calculation.

`dvec` is a frequency varying vector containing the *D*s (obtained from `mu`).

`sens` is the sensitivity of the upper bound in the μ calculation on the *D*s. The sensitivity `sens` is a frequency domain weight calculated by `mu`.

`blk` is the uncertainty block structure. This should correspond with the block strcucture used in the μ calculation (and produced `bnds`, `dvec` and `sens`).

Output arguments:

`dsysL` is the left (i.e., output) block diagonal scaling matrix. It is a SYSTEM matrix (it may be CONSTANT)

`dsysR` is the right (i.e., input) block diagonal scaling matrix. It is the same type as `dsysL`

`msftbatch` is a batch version of `msf` with no user interaction.

See Also

`dkit`, `dkitgui`, `fitmag`, `fitmaglp`, `fitsys`, `genphase`, `magfit`

Purpose

Scale CONSTANT, SYSTEM and VARYING matrices

Syntax

```
matout = mscl(matin,fac); matout = sclin(matin,inputs,fac); matout
   = sclout(matin,outputs,fac);
```

Description

The `mscl` command scales the individual elements of a CONSTANT or VARYING matrix by the scalar `fac`. `mscl` produces a CONSTANT (VARYING) matrix, `matout`, given `matin` is a CONSTANT (VARYING) matrix, and `fac` is a scalar. When the output matrix, `matout`, is VARYING, it has the same independent variable values as `matin`, and is equal to `matin` multiplied by the scalar value `fac`. If `matin` is a SYSTEM matrix, then `mscl` produces a SYSTEM matrix, `matout`, in which the A and C matrices of `matin` are unchanged, and the B and D matrices have been scaled by `fac`. `fac` must be a scalar.

The command `sclin` scales the input channels to the matrix `matin` (which can be a CONSTANT, SYSTEM or VARYING matrix) defined by the variable `inputs`. `inputs` is a vector of integers, which selects the inputs channels (columns) to be scaled by `fac`. The command `sclout` performs the similar scaling on the outputs (or rows). `fac` can be a scalar, single-input/single-output (SISO) SYSTEM or a VARYING, 1 × 1, matrix. If `matin` and `fac` are both VARYING matrices, they must have the same independent variables to be used with `sclin` and `sclout`. The following options can be performed using `sclin` and `sclout`:

- Scale select inputs and outputs of a SYSTEM or CONSTANT (`matin`) matrix by a scalar or a SISO SYSTEM (`fac`).
- Scale select inputs and outputs of a VARYING matrix by a scalar.
- Scale select inputs and outputs of a VARYING matrix by a VARYING (`matin`), 1 × 1, matrix (`fac`).

Examples

mscl scales the three input, two output VARYING matrix, matin, by –2.5.

```
minfo(matin)
varying- 2 pts2 rows3 cols
see(matin)
2 rows                 3 columns
indep variable0.2
   3       13      23
   4       14      24
indep variable0.3
   4       14      24
   5       15      25
matout = mscl(matin,-2.5);
see(matout)
2 rows                 3 columns
indep variable0.2
   -7.5000 -32.5000-57.5000
   -10.0000-35.0000-60.0000
indep variable         0.3
   -10.0000-35.0000-60.0000
   -12.5000-37.5000-62.5000
```

Use the sclin command to scale the first input of a SYSTEM matrix by –3.

```
sys = pck(ones(3,3),2*ones(3,2),3*ones(1,3),4*ones(1,2));
seesys(sys)

1.0e+00    1.0e+00    1.0e+00  |  2.0e+00    2.0e+00
1.0e+00    1.0e+00    1.0e+00  |  2.0e+00    2.0e+00
1.0e+00    1.0e+00    1.0e+00  |  2.0e+00    2.0e+00
-----------------------------------------------------
3.0e+00    3.0e+00    3.0e+00  |  4.0e+00    4.0e+00

sysout = sclin(sys,1,-3);
```

```
seesys(sysout)

1.0e+00    1.0e+00    1.0e+00  |   -6.0e+00    2.0e+00
1.0e+00    1.0e+00    1.0e+00  |   -6.0e+00    2.0e+00
1.0e+00    1.0e+00    1.0e+00  |   -6.0e+00    2.0e+00
-------------------------------------------------------
3.0e+00    3.0e+00    3.0e+00  |   -1.2e+01    4.0e+00
```

The sclout command can be used to scale the first and third outputs of a SYSTEM matrix by the first order transfer function 10/(s + 10).

```
sys = sysrand(2,3,1);
seesys(sys,'%11.2e)

2.19e-01    6.79e-01  |  9.35e-01
4.70e-02    6.79e-01  |  3.84e-01
----------------------------------
5.19e-01    5.35e-02  |  7.70e-03
8.31e-01    5.30e-01  |  3.83e-01
3.46e-02    6.71e-01  |  6.68e-02

sysout=sclout(sys,[1 3],nd2sys(10,[1 10]));
seesys(sysout,'%11.2e')

-1.00e+01   0.00e+00   -1.64e+00   -1.69e-01 |  -2.43e-02
0.00e+00   -1.00e+01   -1.09e-01   -2.12e+00 |  -2.11e-01
0.00e+00    0.00e+00    2.19e-01    6.79e-01 |   9.35e-01
0.00e+00    0.00e+00    4.70e-0     6.79e-01 |   3.84e-01
---------------------------------------------|------------
-3.16e+00   0.00e+00    0.00e+00    0.00e+00 |   0.00e+00
0.00e+00    0.00e+00    8.31e-01    5.30e-01 |   3.83e-01
0.00e+00   -3.16e+00    0.00e+00    0.00e+00 |   0.00e+00
```

```
matin = vpck([ones(3,2);2*ones(3,2);3*ones(3,2)],[1;2;3]);
seesys(matin)
3 rows      2 columns

iv = 1
l.0e+00     l.0e+00
l.0e+00     1.0e+00
l.0e+00     l.0e+00

iv = 2
2.0e+00     2.0e+00
2.0e+00     2.0e+00
2.0e+00     2.0e+00

iv = 3
3.0e+00     3.0e+00
3.0e+00     3.0e+00
3.0e+00     3.0e+00

fac = vpck([1;2;3],[1;2;3]);
sysout = sclout(matin,l,fac)i
seesys(matin)
3 rows2 columns

iv = 1
l.0e+00     l.0e+00
l.0e+00     l.0e+00
l.0e+00     l.0e+00

iv = 2
4.0e+00      4.0e+00
2.0e+00      2.0e+00
2.0e+00      2.0e+00

iv = 3
9.0e+00      9.0e+00
3.0e+00      3.0e+00
3.0e+00      3.0e+00
```

See Also `*`, `mmult`, `scliv`, and `sel`

mu, muunwrap, randel, unwrapd, unwrapp

Purpose

Compute upper and lower bounds for the complex and mixed (real and complex) structured singular value (referred to as mixed μ) of a VARYING/ CONSTANT matrix

Syntax

```
[bnds,dvec,sens,pvec,gvec] = mu(matin,blk,options);
[dl,dr,gl,gm,gr,pert] = muunwrap(dvec,gvec,pvec,blk);
[dl,dr,gl,gm,gr] = muunwrap(dvec,gvec,blk);
[dl,dr] = muunwrap(dvec,blk);
pert = randel(blk,nrm,opt);
[dl,dr] = unwrapd(dvec,blk);
pert = unwrapp(pvec,blk);
```

The functions associated with mixed μ are

`mu`	general complex and mixed computation
`muunwrap`	extract block-diagonal *D*- and *G*- scalings from row vectors containing the scalings
`randel`	create random block structured matrix
`unwrapd`	extract block-diagonal *D*-scalings from row vector containing the scalings
`unwrapp`	extract block-diagonal perturbation from row vector containing the perturbation

Description

Input arguments:

`matin` A CONSTANT or VARYING matrix

`blk` An array that describes the perturbation block structure. Its size is `nblk` × 2, where `nblk` is the total number of blocks in the perturbation structure. The *i*th row of `blk` defines the dimensions of the *i*th perturbation block. If `blk(i,:) = [-r 0]`, then the *i*th block is an *r* × *r* repeated, diagonal real scalar perturbation, while if `blk(i,:) = [r 0]`, then the *i*th block is an *r* × *r* repeated, diagonal complex scalar perturbation, and if is `blk(i,:) = [r c]`, then the *i*th block is an *r* × *c* complex full-block perturbation. If `blk` is omitted, the default is all 1 × 1 complex blocks, and results in an error if `matin` is not square.

`options` An optional character string describing the desired computations. It can consist of the following characters:
`'l'` - compute lower bound using a power iteration
`'t'` - use more iterations in the lower bound
`'R'` - start power iteration with RANDOM vectors
`'Rj'` - restart lower bound `j` times with RANDOM vectors where `j` is an integer between 1 and 9
`'u'` - compute upper bound using a balanced/LMI technique
`'c'` - compute upper bound to greater accuracy`'C'`
`'C'` - compute tighest upper bound (may be slow)
`'f'` - compute a fast but crude upper bound
`'r'` - restart computation at EACH independent variable
`'s'` - suppress progress information
`'w'` - suppress warnings
`'L'` - compute only the lower bound
`'U'` - compute only the upper bound

The default value of options is `'lu'`, meaning that a lower bound will be computed using the power method, Young and Doyle 1990 and Packard *et al.* 1988, and an upper bound will be computed, using the balanced/AMI technique, Young *et al.*, 1992, for computing the upper bound from Fan *et al.*, 1991.

Output arguments:

`bnds` A 1 × 2 vector. If `matin` is VARYING, so is `bnds`, whereas if `matin` is a CONSTANT matrix, then `bnds` is CONSTANT. The first column of `bnds` contains an upper bound to mixed μ of `matin`, and the second column contains a lower bound to mixed μ.

`dvec` and `gvec` Row vectors which contain the D and G scaling matrices that have produced the upper bound in `bnds`. `dvec` and `gvec` are the same data type as `bnds` and are stored as vectors to save memory. They can be unwrapped into the appropriate D and G matrices by using the command, `muunwrap`:

```
    [dl,dr,gl,gm,gr] = muunwrap(dvec,gvec,blk);
```

The upper bound in `bnds` for a matrix M is a number $\beta > 0$ such that there are scaling matrices D_l, D_r, G_l, G_m, G_r (see Young *et al.*, 1992, for details) satisfying

$$\bar{\sigma}\left((I + G_l^2)^{-\frac{1}{4}}\left(\frac{D_l M D_r^{-1}}{\beta} - jG_m\right)(I + G_r^2)^{-\frac{1}{4}}\right) \le 1$$

`sens` is a row vector which contains the sensitivity of

$$\bar{\sigma}(D_l M D_r^{-1})$$

with respect to the values in D_l (and D_r). It is calculated in an ad-hoc manner, and is mainly used when fitting frequency varying Ds with rational functions via the routines in `dkit`, `dkitgui`, and `autodkit`.

`pvec` is a row vector containing a perturbation matrix that has the structure defined by `blk`. As with `dvec` and `gvec`, `pvec` is the same data type as `matin` and is stored as a vector. It can be unwrapped into the actual perturbation with the command, `unwrapp`.

```
pert = unwrapp(pvec,blk);
```

After being unwrapped, the perturbation matrix `pert` satisfies three conditions:

- It has the block structure defined by `blk`;
- The maximum singular value of `pert` is equal to the reciprocal of the lower bound in `bnds` (when the lower bound is not zero);
- The matrix `mmult(matin,pert)` has an eigenvalue equal to 1 at each independent variable.

Examples

Suppose `sys` is a system matrix with four inputs and four outputs, and that it is stable. `sys_g` is a frequency response of `sys`.

```
% Δ is 4 1 × 1 perturbation blocks
blk = ones(4,2);

% Calculate μ on frequency response
[bnds,dvec,sens,pvec] = mu(sys_g,blk);

% Unwrap the D scaling matrices
[dl,dr] = unwrapd(dvec,blk);

% Generate scaled matrix
dmdi = mmult(dl,sys_g,minv(dr));

% Verify the upper bound
vplot('liv,m',vnorm(dmdi),sel(bnds,1,1))

% Unwrap the perturbation
actpert = unwrapp(pvec,blk);

% Check that perturbation is correct structure
see(xtracti(actpert,1))

% Check lower bound and perturbation
vplot('liv,lm',sel(bnds,1,2),vnorm(actpert))
```

% Form $M\Delta$

```
mdel = mmult(sys_g,actpert);
```

% Verify $M\Delta$ has an eigenvalue at 1

```
veig(mdel)
```

Looking at the same frequency response sys_g with a mixed real/complex block structure.

% Δ is 4 1×1 perturbation blocks

% with the first 2 real, and the last 2 complex

```
blk = [-1 0; -1 0; 1 0; 1 0];
```

% Calculate mixed μ on frequency response

```
[bnds,dvec,sens,pvec,gvec] = mu(sys_g,blk);
```

% Unwrap the *D* and *G* scaling matrices

```
[dl,dr,gl,gm,gr] = muunwrap(dvec,gvec,blk);
```

% Generate scaled matrix

```
dmd = mmult(dl,sys_g,minv(dr));
oobdmd = veval('*',dmd,minv(sel(bnds,1,1)));
oobdmdjg = msub(oobdmd,mscl(gm,j));
scall = madd(eye(4),mmult(gl,gl));
scalr = madd(eye(4),mmult(gr,gr));
scall = veval(,scall,-0.25);
scalr = veval(”,scalr,-0.25);
scaledmat = mmult(scall,oobdmdjg,scalr);
```

% Verify the upper bound (scaledmat should have `norm` ≤ 1)

```
vplot('liv,m',vnorm(scaledmat))
```

% Unwrap the perturbation

```
pert = unwrapp(pvec,blk);
```

% Check that perturbation is correct structure

```
see(xtracti(pert,1))
```

% Check lower bound and perturbation

```
vplot('liv,m',sel(bnds,1,2),minv(vnorm(pert)))
```

```
% Form MΔ
 mdel = mmult(sys_g,pert);

% Verify MΔ has an eigenvalue at 1
 see(veig(mdel))
```

For more examples of computing bounds for μ, please refer to the "Computational Exercise with the mu Command" and "Computational Exercise with the mu Command — Mixed Perturbations" sections in Chapter 4 as well as the robust multivariable control design examples in Chapters 6 and 7.

Algorithm

Peter Young and Matt Newlin helped write the `mu` program and supporting routines.

The lower-bound power algorithm is from Young and Doyle, 1990, and Packard *et al.* 1988.

The upper-bound is an implementation of the bound from Fan *et al.*, 1991, and is described in detail in Young *et al.*, 1992. In the upper bound computation, the matrix is first balanced using either a variation of Osborne's method (Osborne, 1960) generalized to handle *repeated scalar* and *full* blocks, or a Perron approach. This generates the standard upper bound for the associated complex μ problem. The Perron eigenvector method is based on an idea of Safonov, (Safonov, 1982). It gives the exact computation of μ for positive matrices with scalar blocks, but is comparable to Osborne on general matrices. Both the Perron and Osborne methods have been modified to handle *repeated scalar* and *full* blocks. Perron is faster for small matrices but has a growth rate of n^3, compared with less than n^2 for Osborne. This is partly due to the MATLAB implementation, which greatly favors Perron. The default is to use Perron for simple block structures and Osborne for more complicated block structures. A sequence of improvements to the upper bound is then made based on various equivalent forms of the upper bound. A number of descent techniques are used which exploit the structure of the problem, concluding with general purpose AMI optimization (Boyd *et al.*), 1993, to obtain the final answer.

Reference

Boyd, S. and L. El Ghaoui, “Methods of centers for minimizing generalized eigenvalues,” *Linear Algebra and Its Applications,* vol. 188–189, pp. 63–111, 1993.

Fan, M. A. Tits, and J. Doyle, “Robustness in the presence of mixed parametric uncertainty and unmodeled dynamics,” *IEEE Transactions on Automatic Control,* vol. AC–36, pp. 25–38, 1991.

Osborne, E. “On preconditioning of matrices,” *Journal of Associated Computer Machines,* vol. 7, pp. 338–345, 1960.

Packard, A.K., M. Fan and J. Doyle, “A power method for the structured singular value,” *Proc. of 1988 IEEE Conference on Control and Decision,* pp. 2132–2137, December 1988.

Safonov, M. “Stability margins for diagonally perturbed multivariable feedback systems,” *IEEE Proc.,* vol. 129, Part D, pp. 251–256, 1982.

Young, P. and J. Doyle, “Computation of with real and complex uncertainties,” *Proceedings of the 29th IEEE Conference on Decision and Control,* pp. 1230–1235, 1990.

Young, P., M. Newlin, and J. Doyle, “Practical computation of the mixed problem,” *Proceedings of the American Control Conference,* pp. 2190–2194, June, 1992.

See Also

`blknorm`, `dypert`, `genmu`, `norm`, `vnorm`, `vrho`, `vsvd`

musynfit, musynflp, muftbtch

Purpose

Interactive *D*-scaling rational fit routines used in old μ-synthesis routines

---

Note These routines are included only for backwards compatibility with versions 1.0 and 2.0. They will not be supported in future versions. They should not be used by new users of the toolbox. All new routines should be based on the new routine, `msf`, which is described on page ???.

Syntax

```
[dsysL,dsysR] = musynfit(pre_dsysL,dvec,sens,blk,...
                             nmeas,ncntrl,clpg,upbd,wt)
[dsysL,dsysR] = musynflp(pre_dsysL,dvec,sens,blk,...
                             nmeas,ncntrl,clpg,upbd,wt)
[dsysL,dsysR] = muftbtch(pre_dsysL,dvec,sens,blk,...
                             nmeas,ncntrl,dim)
```

Description

`musynfit` fits the magnitude curve obtained by multiplying the old *D* frequency response (from `pre_dsysl`) with the `dvec` data. `musynfit` returns stable, minimum phase system matrices `dsysL` and `dsysR`, which can be absorbed into the original interconnection structure. Once absorbed, a H_∞ design is performed with `hinfsyn` completing another *D–K* iteration of μ-synthesis.

For the first μ-synthesis iteration, set the variable `pre_dsysl` to the string `first`. In subsequent iterations, `pre_dsysl` should be the previous (left) rational *D*-scaling system matrix, `dsysL`. Essentially, the element-by-element magnitudes of the matrices `mmult(unwrapd(dvec,blk),frsp(pre_dsysL,getiv(dvec)))`, and `frsp(dsysL,getiv(dvec))` are equal.

The (optional) variable `clpg` is the VARYING matrix that produced the `dvec`, `sens`, and `upbd` data output from `μ`. The fitting procedure is interactive (`musynfit` or `musynflp`), and fits (in magnitude) these scalings with rational, stable transfer function matrices, $\hat{D}(s)$. After fitting the `dvec` data, plots of

$$\bar{\sigma}(D_f(j\omega)\cdot \mathrm{clpg}(j\omega)\cdot D_f^{-1}(j\omega))$$

and

$$\bar{\sigma}(\hat{D}(j\omega)\cdot \mathrm{clpg}(j\omega)\cdot \hat{D}^{-1}(j\omega))$$

are shown in the lower graph window for comparison. At this point, you have the option of refitting the *D* data. If `clpg` and `upbd` are not provided, the default is to plot the `sens` variable in the the lower graph.

Note You are strongly discouraged from calling `musynfit` and `musynflp` directly and are encouraged to use `dkit` or `dkitgui` to perform μ-synthesis calculations.

Input arguments:

`pre_dsysl` is set to the character string `'first'` for the first iteration. As the iteration proceeds, it should be the previous `dsysL`.

`sens` is the sensitivity of the upper bound in the μ calculation on the *D*s. The sensitivity `sens` is a frequency domain weight, which is obtained from `mu`.

`dvec` is a frequency varying row vector containing the *D*s (from `mu`).

`blk` is the block structure, same block structure used in `mu`.

`nmeas` is the number of measurements in control problem.

`ncntrl` is the number of controls in control problem.

`clpg` is the frequency response upon which the calculation was performed (optional).

`upbd` is the upper bound from the μ calculation (optional).

`wt` is a weight used to influence the frequency range in which the data is to be fit more accurately (optional).

`dim` is the highest order fit to be used (only used in `muftbtch`).

Output arguments:

`dsysL` is the output (left) block diagonal, SYSTEM scaling.

`dsysR` is the input (right) block diagonal, SYSTEM scaling (needs to be inverted before being absorbed into the interconnection structure).

`musynflp` has the same inputs and outputs and user interaction as `musynfit` but uses a linear programing routine to do the fitting. `muftbtch` is a batch version of `musynflp` that has no user interaction. The extra argument `dim` is a required argument of parameters for the linear program. `dim` has the form `[hmax htol nmin nmax]` where

- `hmax` is a measure of the allowable error in the fit.
- `htol` is a measure of the accuracy with which the optimization is carried out.
- `nmin` and `nmax` are the minimum and maximum orders considered in the problem

For more detail about the role of `hmax` and `htol`, see the reference pages for `fitmaglp` and `magfit`. Reasonable choices are `hmax` = .26 and `htol` = .1.

The `musynflp` and `musynfit` commands provide the option of fitting the frequency varying *D*-scale data by hand using the μ-Tools `drawmag` command. You can invoke this option with the string '`drawmag`' in response to the prompt

```
ENTER ORDER OF CURVE FIT or 'drawmag'
```

The mouse is used in the plot window to identify the data to be fit with a stable, minimum-phase system. See the `drawmag` command for more information.

Examples

`musynfit` is used within a *D–K* iteration (μ-synthesis) to fit the *D*-scales, which are output from the `mu` command. The first step in the *D–K*iteration is to design an $H\infty$ control law. The closed-loop system is analyzed with `mu` based on the block structure `blk` defined. The optimal *D*-scaling output from `mu`, which are real coeeficients, are fit with real, rational, minimum-phase stable transfer functions via `musynfit`. These fitted *D*-scales are wrapped back around the orginal interconnection structure `P`. After absorbing the *D*-scales, another *D–K* iteration is performed, starting with the design of an H_∞ control law for the modified plant. This process usually continues until the value of μ doesn't change significantly between control design iterations.

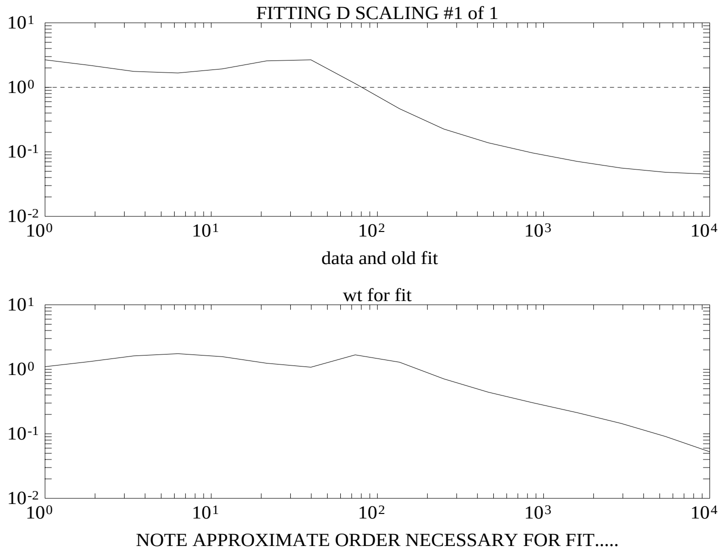
This example is taken from the "HIMAT Robust Performance Design Example" section in Chapter 7. `himat_ic` contains the open-loop interconnection structure. It has one multiplicative input perturbation, which is two by two, and has two error signals, and two external disturbances. There are two measurements, and two control inputs to the system. The block structure for the μ-analysis problem is given by `blk=[2 2; 2 2]`.

First step in a *D*–*K* iteration is to design an H_∞ controller and analyze the closed loop system with μ.

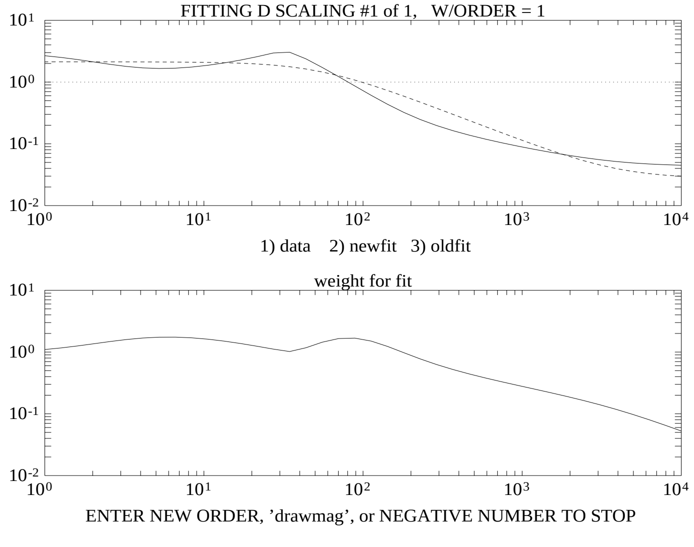
```
mkhic
omega = logspace(0,4,40);
blk = [2 2; 2 2];
[k1,g1,gf1] = hinfsyn(himat_ic,2,2,0.8,6,0.05,2);
g1_g = frsp(g1,omega);
[bnds1,dvec1,sens1,rp1] = mu(g1_g,blk);
```

The *D*-scalings output from the μ-analysis problem needs to be fit with real, rational, stable, minimum-phase transfer functions. This is done with `musynfit`. The first *D*-scale is fit with both a first and third-order transfer function, with the third order transfer function selected.

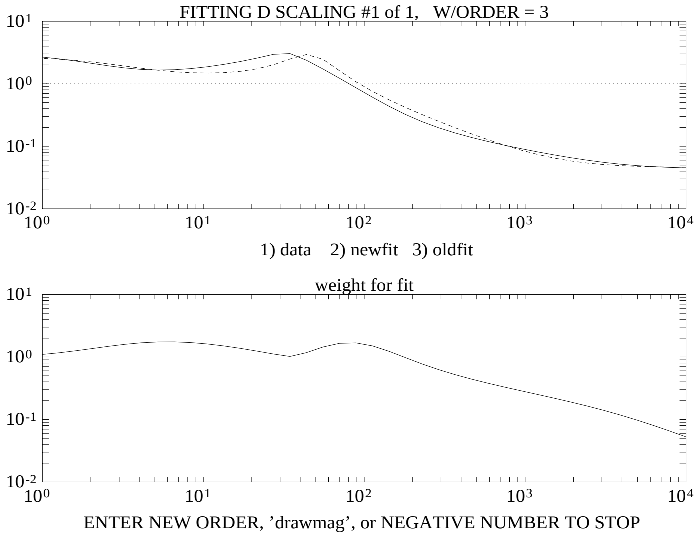
```
[dsysL1,dsysR1] = musynfit('first',dvec1,sens1,blk,2,2);
```



```
ENTER ORDER OF CURVE FIT or 'drawmag' 1
```



```
ENTER ORDER OF CURVE FIT or 'drawmag' 3
```



```
ENTER ORDER OF CURVE FIT or 'drawmag' -1
```

Now the fitted *D*-scales are absorbed into the interconnection structure, himat_ic, to generate himat_ic2.

```
mu_ic1 = mmult(dsysL1,himat_ic,minv(dsysR1));
```

The modified interconnection structure, mu_ic1, is used in the second iteration of μ-synthesis. An H_∞ control law is designed for the modified system, and then analyzed again using mu.

```
[k2,g2] = hinfsyn(mu_ic1,2,2,.9,1.3,.04,2);
g2_g = frsp(g2,omega);
[bnds2,dvec2,sens2,pvec2] = mu(g2_g,blk);
```

The new *D*-scales are fit again using the previous *D*-scale information.

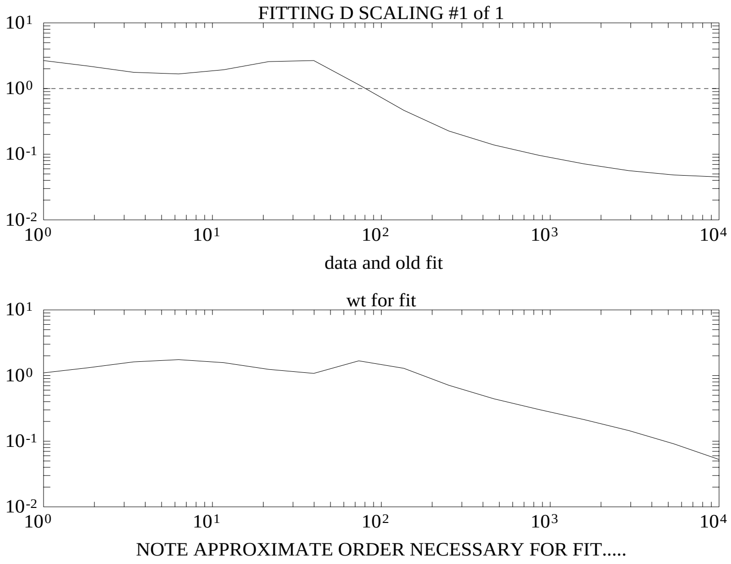
```
[dsysL2,dsysR2] = musynfit(dsysL1,dvec2,sens2,blk,2,2);
```

The graphs displayed with `musynfit` for this iteration are not included. Wrap the new fitted *D*-scales around the *original* plant interconnection structure and start *D–K* iteration again.

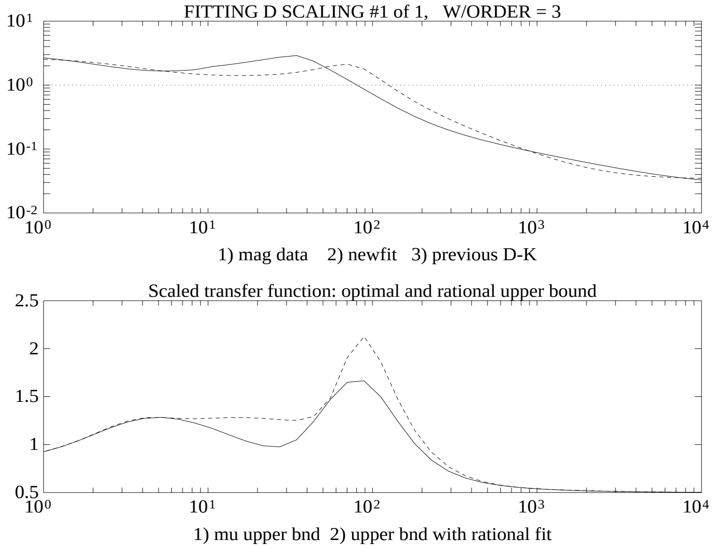
```
mu_ic2 = mmult(dsysL2,himat_ic,minv(dsysR2));
[k3,g3] = hinfsyn(mu_ic2,2,2,.9,1.3,.05,2);
```

`musynfit` can be called as before with the frequency response of the closed-loop system analyzed using `mu`, `g1_g`, and the `mu` upper bound, `sel(bnds1,1,1)` passed. The first *D*-scale is fit with both a first- and third-order transfer function and the first order transfer function selected. As you can see from the scaled upper bound plots (the lower graph), the first-order fit does a better job minimizing the scaled upper bound.

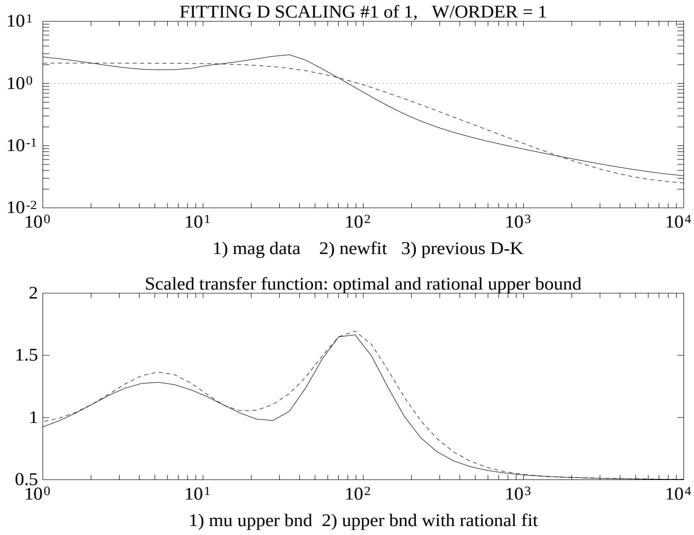
```
[dsysL1,dsysR1] = ...
musynfit('first',dvec1,sens1,blk,2,2,gl_g,sel(bnds1,1,1)
```



```
ENTER ORDER OF CURVE FIT or 'drawmag' 3
```



```
ENTER ORDER OF CURVE FIT or 'drawmag' 1
```



```
ENTER ORDER OF CURVE FIT or 'drawmag' -1
```

As before you would aborb the fitted *D*-scales into the interconnection structure, `himat_ic`, to generate `himat_ic2`.

Algorithm

A frequency response is done on the previous rational *D*-scaling matrix. This is multiplied by the *current* data in `dvec`, to produce the frequency varying scaling that needs to be fit. The fit is *only in magnitude*, and the freedom in the phase allows the rational function to be defined as stable, and minimum phase. `musynfit` calls `fitsys`, which calls `fitmag`, `flatten`, and `genphase`. The curve fitting is done the `fitsys` command.

`musynflp` is an alternative program that uses linear programming to do the fit. `musynflp` fits the data very well *within* the frequency response window at the expense of perhaps large variations outside the data window. This may lead to problems in *D–K* iteration. `muftbtch` is a batch version of `musynflp`.

Reference

Doyle, J.C., K. Lenz, and A.K. Packard, “Design examples using μ-synthesis: Space shuttle lateral axis FCS during reentry,” *NATO ASI Series, Modelling, Robustness and Sensitivity Reduction in Control Systems*, vol. F34 R.F. Curtin, Editor, Springer-Verlag, Berlin-Heidelberg, 1987.

See Also

`drawmag`, `fitmag`, `fitmaglp`, `fitsys`, `flatten`, `genphase`, `invfreqs`, `magfit`, `msf`

ncfsyn, cf2sys, emargin

Purpose

ncfsyn synthesizes an H_∞ controller to robustly stabilize a family of systems given by a ball of uncertainty in the normalized coprime factors of the system description

cf2sys calculates a SYSTEM/CONSTANT/VARYING matrix from a coprime factorization.

emargin calculates the normalized coprime factor/gap metric robust stability margin $b(P,K)$ as defined in the “Loop Shaping Using H• Synthesis” section in Chapter 3.

Syntax

```
[sysk,emax,sysobs] = ncfsyn(sysgw,factor,opt)
sysout = cf2sys(sysrcf)
emarg = emargin(sys_g,sysw,tol)
```

Description

A method of designing controllers is to use a combination of loop shaping and robust stabilization as proposed in McFarlane and Glover. The first step is to design a pre- and post-compensator $W_1(s)$ and W_2, so that the gain of $W_2(s)P(s)W_1(s)$ is sufficiently high at frequencies where good disturbance attenuation is required and is sufficiently low at frequencies where good robust stability is required. The second step is to design a feedback controller, $K\infty$, so that

$$\frac{1}{b(W_2PW_1, K_\infty)} := \left\| \begin{bmatrix} I \\ K_\infty \end{bmatrix} (I - W_2PW_1, K_\infty)^{-1} [W_2PW_1, I] \right\|_\infty \text{ð} \ \frac{1}{\varepsilon}$$

which will also give robust stability of the perturbed weighted plant

$$(N + \Delta_1)(M + \Delta_2)^{-1} \ \text{ for } \ \left\| \begin{bmatrix} \Delta_1 \\ \Delta_2 \end{bmatrix} \right\|_\infty < b(W_2PW_1, K_\infty)$$

where $NM^{-1} = W_2PW_1$is a normalized coprime factorization satisfying $N(jw)^*N(jw) + M(jw)^*M(jw) = I$. This stability margin is always less than 1 and gives a good indication of robust stability to a wide class of unstructured perturbations, with values of $\varepsilon > 0.2 - 0.3$ generally satisfactory.

The closed-loop H_∞-norm objective has the standard signal gain interpretation. Finally it can be shown that the controller, $K\infty$, does not substantially affect the loop shape in frequencies where the gain of W_2PW_1 is either high or low, and will guarantee satisfactory stability margins in the frequency region of gain cross-over. In the regulator set-up, the final controller to be implemented is $W_1K\infty W_2$.

When the option `'ref'` is specified, the controller includes an extra set of reference inputs as proposed in Vinnicombe, 1993, and should be implemented as `u = sysk *` $\begin{bmatrix} y \\ r \end{bmatrix}$ where `y = sysgw * u` and `r` is a reference input. The closed-loop response will then be `y = Nr`, where `N` is the numerator of a normalized right coprime factor of `sysgw`.

Input arguments

`sysgw` the weighted system to be controlled

`factor` =1 implies that an optimal controller is required.
>1 implies that a suboptimal controller is required achieving a performance FACTOR less than optimal.

`opt` `'ref'` the controller includes an extra set of reference input and should be implemented as in Figure 3–12 on page 3-? (optional).

Output arguments

`sysk` H_∞ loopshaping controller

`emax` Stability margin as an indication robustness to unstructured perturbations. `emax` is always less than 1 and values of `emax` greater than 0.3 generally indicate good robustness margins.

`sysobs` H_∞ loopshaping observer controller. This variable is created only if `factor`>1 and `opt = 'ref'`

`cf2sys` creates a SYSTEM/CONSTANT/VARYING matrix from a coprime factorization.

`emargin` calculates the normalized coprime factor/gap metric robust stability margin $b(P,K)$ as defined in the “Loop Shaping Using H• Synthesis” section in

Chapter 3. The default value for the tolerance `tol` supplied to the `hinfnorm` computation is 0.001.

Algorithm

See the McFarlane and Glover reference for details.

Reference

McFarlane, D.C.and K. Glover, Robust Controller Design using Normalised Coprime Factor Plant Descriptions, Springer Verlag, *Lecture Notes in Control and Information Sciences*, vol. 138, 1989.

McFarlane, D.C., and K. Glover, “A Loop Shaping Design Procedure using Synthesis,” *IEEE Transactions on Automatic Control*, vol. 37, no. 6, pp. 759–769, June 1992.

Vinnicombe, G., “Measuring Robustness of Feedback Systems,” PhD dissertation, Department of Engineering, University of Cambridge, 1993.

See Also

`gap`, `hinfsyn`, `hinfnorm`, `nugap`

Purpose

Convert single-input/single-output (SISO) transfer functions to SYSTEM matrices

Syntax

```
sys = nd2sys(num,den,gain)
sys = zp2sy(zeros,poles,gain)
```

Description

nd2sys converts a numerator/denominator (num,den) SISO transfer function into a SYSTEM matrix, sys. This function uses the MATLAB command tf2ss for the conversion. zp2sys converts zeros and poles of a SISO transfer function into a SYSTEM matrix, sys. This function uses the MATLAB command zp2ss to do the conversion. An optional argument gain can be supplied to scale the transfer function. The default value of gain is 1. The output SYSTEM matrix, sys, is balanced with a call to the μ-Tools command sysbal prior to being return if the real parts of the poles are all less than zero.

Examples

Convert the single-input/single-output transfer function sys1 into a SYSTEM matrix.

$$sys = \frac{4s^2 + 5s + 1}{7s^4 + 3s^3 + 6s^2 + 2s + 8}$$

```
sys = nd2sys([4 5 1],[7 3 6 2 8]);
minfo(sys)
system:4 states1 outputs1 inputs
see(sys)
see (sys)

A matrix

-0.4286    -0.8571     -0.2857   -1.1429
 1.0000          0           0         0
      0     1.0000           0         0
      0          0      1.0000         0
```

```
B matrix
   1
   0
   0
   0

C matrix
0    0.5714    0.7143    0.1429

D matrix
   0
```

Note zp2sys requires the Signal Processing or Control Toolbox.

Algorithm

nd2sys and zp2sys realize the transfer functions using the MATLAB commands tf2ss and zp2ss.

See Also

pss2sys, pck, tf2ss

negangle

Purpose

Calculate the angle of elements of a matrix, which is always between [–2π,0]

Syntax

```
y = negangle(x)
```

Description

negangle returns the phase angles, in radians, of a matrix with complex valued elements. The returned value is always in the range 0 to –2π radians.

Examples

```
a = [1+i,1-i];
see(angle(a))
   0.7854-0.7854
see(negangle(a))
   -5.4978-0.7854
```

See Also

angle

Purpose

Convert to and from CONSTANT and SYSTEM matrices

Syntax

```
sys = pss2sys(mat,n)
sys = pck(A,B,C,D)
sys = pck(A,B,C)
[A,B,C,D] = unpck(sys)
mat = sys2pss(sys)
```

Description

pss2sys translates a regular MATLAB matrix that is in packed form into a SYSTEM matrix. mat contains [A B; C D] which describes the individual components of a SYSTEM matrix with n being the number of states (size of A). sys2pss returns the CONSTANT matrix, mat = [A B; C D], from the input SYSTEM matrix sys.

pck takes consistent state-space data and forms a SYSTEM matrix with the data structure defined. Consistent state-space data requires a square A matrix, a B matrix with the same number of rows as A, a C matrix with the same number of columns as A, and a D matrix with same number of columns of B and rows of C. If the fourth input argument is omitted, then the D matrix is assumed to be identically zero, of appropriate dimensions. unpck is the inverse operation of pck, taking a SYSTEM matrix sys and converting it to A, B, C and D CONSTANT matrices.

Note that based on the data structure definition, a -Inf in the bottom right corner of a matrix denotes a SYSTEM matrix, with the top, right corner element of the matrix containing the number of states.

Examples

Create a SYSTEM matrix from MATLAB CONSTANT matrices via pss2sys. Define matrices A, B, C, and D as follows.

```
A = [-1 1; -1 -3]; B =[2 2; 2 2]i C = [3 3]; D = [4 4]
mat = [A B; C D];
sys = pss2sys(mat,2);
minfo(mat)
3 rows4 cols: regular MATLAB matrix
mat =
   -1          1               22
   -1          -3              22
   3           3               44
minfo(sys)
system: 2 states1 outputs2 inputs
seesys(sys)

-1.0e+00     1.0e+00  |  2.0e+00     2.0e+00
-1.0e+00    -3.0e+00  |  2.0e+00     2.0e+00
----------------------|----------------------
 3.0e+00     3.0e+00  |  4.0e+00     4.0e+00
```

The same SYSTEM matrix can be constructed using the pck command.

```
sys = pck(A,B,C,D);
minfo(sys)
system: 2 states1 outputs2 inputs
seesys(sys)

-1.0e+00     1.0e+00  |  2.0e+00     2.0e+00
-1.0e+00    -3.0e+00  |  2.0e+00     2.0e+00
----------------------|----------------------
 3.0e+00     3.0e+00  |  4.0e+00     4.0e+00
```

Matrices A, B, C, and D can be constructed from sys using unpck.

```
[A,B,C,D] = unpck(sys);
A
A =
   -1          1
   -1          3
B
B =
   2           2
   2           2
C
C =
   3           3
D
D =
   4           4
mat = sys2pss(sys);
mat
mat =
   -1          12          2
   -1          -32         2
   3           34          4
```

See Also

minfo, nd2sys, vpck, vunpck, zp2sys

pkvnorm, vnorm

Purpose

Find the norm or peak norm of a VARYING matrix

Syntax

```
[peak,indv,index] = pkvnorm(matin,p)
out = (matin)
out = (matin,p)
```

Description

pkvnorm sweeps through the independent variable, calculating the norm of each matrix as specified by the input argument p, following the convention from MATLAB's norm command. The default for p is the largest singular value of matin. The three output arguments all pertain to the peak and its location: peak value, peak, the independent variable's value, indv, and the independent variable's index, index.

vnorm is a VARYING matrix version of MATLAB's norm command. The operation of the norm command is identical to vnorm, except that vnorm also works on CONSTANT and VARYING matrices, which produces a CONSTANT or VARYING output. vnorm returns the matrix out with its norm at each independent variable value.

Examples

The two-input/two-output VARYING matrix matin has its independent variable at 0.2 and 0.6. The vnorm command finds the largest singular value of the VARYING matrix at each value of the indepedent variable. pkvnorm returns the largest singular value of the VARYING matrix, its independent variable value, and the index of the independent variable.

```
minfo(matin)
varying:2 pts2 rows2 cols
see(matin)
2 rows2 columns
indep variable 0.2
 1.0000      0
 0           0.5000
indep variable 0.6
 4           0
-5           1
nrm = vnorm(matin,2);
see(nrm)
```

```
1 row    1 column
indep variable 0.2
  1
indep variable 0.6
  6.4510
[peak,indv,index] = pkvnorm(matin);
peak
peak =
  6.4510
indv
indv =
  0.6000
index
index ==
  2
```

Algorithm

pkvnorm and vnorm call the MATLAB norm command.

See Also

norm, vsvd

Purpose

Solution of a Riccati equation via eigenvalue decomposition

Syntax

```
[x1,x2,fail,reig_min] = ric_eig(ham,epp)
x = x2/x1;
```

Description

`ric_eig` (along with a call to `x=x2/x1`) solves the Riccati equation,

$$A'X + XA + XRX - Q = 0$$

with the constraint that the matrix $A + RX$ has all of its eigenvalues in open left-half plane. The data matrices A, R and Q come from the input Hamiltonian matrix, `ham`, in the form

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

and it is assumed that $R = R'$, $Q = Q'$.

If `ham` has no $j\omega$ axis eigenvalues, then there exists $n \times n$ matrices `x1` and `x2` such that `[x1; x2]` spans the n-dimensional stable, invariant subspace of `ham`. If the matrix `x1` is indeed invertible, then `X := x2 * x1-1` satisfies the Riccati equation and results in $A + RX$ being stable. It is the only such matrix with these properties.

`ric_eig` has internal error checking and returns a `fail` value of 1 if $j\omega$ axis eigenvalues of `ham` are found. If this occurs, there is no n-dimensional, stable invariant subspace, and hence no stabilizing Riccati solution. An eigenvalue is considered to be purely imaginary if the magnitude of the real part is less than `epp`. The minimum real part of the eigenvalues is returned in `reig_min`. `epp` is an optional argument and its default value is $1e - 10$.

Note When a Hamiltonian has repeated eigenvalues, solving the Riccati equation via the eigenvalue method may have problems. This is due to the MATLAB command `eig` incorrectly selecting the eigenvectors associated with the repeated roots.

Algorithm

Under the assumption that the Hamiltonian matrix has a full set of eigenvectors, the stable-invariant subspace is spanned by the eigenvectors associated with the stable eigenvalues. Hence, an eigenvalue-eigenvector decomposition can obtain the stable invariant subspace of the Hamiltonian matrix, `ham`. Assuming there are no $j\omega$ axis eigenvalues, and that there is a full set of eigenvectors, the two components, `x1` and `x2`, can be generated by choosing the eigenvectors associated with the stable eigenvalues. The `ric_eig` subroutine operates on the assumption that the Jordan form of the Hamiltonian is diagonal, and returns the stable invariant subspace, as spanned by the eigenvectors, in the two block form described above.

See Also

`eig`, `h2syn`, `hinfsyn`, `hinffi`, `ric_schr`

ric_schr

Purpose

Solve a Riccati equation via Schur decomposition

Syntax

```
[x1,x2,fail,reig_min] = ric_schr(ham,epp)
x = x2/x1
```

Description

`ric_schr` (along with a call to x=x2/x1) solves the Riccati equation,

$$A'X + XA + XRX - Q = 0$$

such that $A + RX$ is stable. A real Schur decomposition can obtain the stable invariant subspace of the Hamiltonian matrix, `ham`. The data matrices A, R, and Q come from the input Hamiltonian matrix in the form

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

and it is assumed that and it is assumed that $R = R'$, $Q = Q'$.

If `ham` has no $j\omega$ axis eigenvalues, then there exists $n \times n$ matrices `x1` and `x2` such that `[x1; x2]` spans the `n`-dimensional stable, invariant subspace of `ham`. If x1 is invertible, then `X := x2 * x1-1` satisfies the Riccati equation and results in $A + RX$ being stable. The output flag `fail` is nominally 0. If there are $j\omega$-axis eigenvalues, `fail` is set to 1. If there are an unequal number of positive and negative eigenvalues, `fail` is set to 2, and if both conditions occur, `fail` = 3.

Algorithm

`ric_schr` calls `csord` to produce an ordered complex Schur form, which is converted to a real Schur form, and yields a stable, invariant subspace of the Hamiltonian. The `csord` command orders the solution with negative real eigenvalues in the top half of the matrix and the positive real eigenvalues on the bottom, and returns the stable solution. The input matrix is assumed to be a Hamiltonian matrix of size 2*n* with `n` stable eigenvalues and *n* unstable eigenvalues. The minimum real part of the eigenvalues is output to `reig_min`. `epp` is an optional argument and its default value is `1e-10`.

See Also

`csord`, `h2syn`, `hinfsyn`, `hinffi`, `ric_eig`, `schur`

Purpose

Display the real, imaginary, frequency and damping ratios of a CONSTANT input vector

Syntax

```
rifd(vec)
```

Description

`rifd` displays the real, imaginary, frequency, and damping ratios of a CONSTANT input vector. The *i*th frequency is given by the

$$(\mathbf{real(sys(i))}^2 + \mathbf{imag(sys(i)}^2)^{\frac{1}{2}}$$

and its damping ratio is defined as `real(sys(i))/ω(i)`.

Examples

Display the poles of the SYSTEM matrix, sys.

```
sys = pck(rand(4,4),rand(4,2),rand(1,4));
rifd(spoles(sys))

    real        imaginary       frequency      damping
-7.6287e-02    0.0000e+00    7.6287e-02    l.0000e+00
 1.0820e–01   –4.6815e–01    4.8049e–01   –2.2519e–01
 1.0820e–01    4.6815e–01    4.8049e–01   –2.2519e–01
 1.4095e+00    0.0000e+00    1.4095e+00   -l.0000e+00
```

See Also

`eig`, `imag`, `spoles`, `real`

samhld

Purpose

Create a sample-hold approximation of a continuous system

Syntax

```
discout = samhld(sys,T)
```

Description

`samhld` applies a sample-hold to the input of the continuous-time systems `sys`, and samples the output, to produce a discrete-time system, `discout`. The sampling time is the same at the input and output, and is specified by `T`.

Examples

Construct the system `sys` via the `nd2sys` command and verify that all of its poles are in the closed left-half plane. Perform a sample hold of the system for a 200 Hz sample rate. All the poles for the discretized system, `discout`, are within the unit disk.

```
sys = nd2sys([1 2 4 5],[2 7 9 2]);
spoles(sys)

-1.6114 + 1.0049i
-1.6114 - 1.0049i
-0.2773
discout = samhld (sys ,1 / 2 0 0)
seesys (discout, '%8 .4f ')

0.9826    -0.0223   -0.0050  |  0.0050
o.0050     0.9999   -0.0000  |  0.0000
0.0000     0.0050    1.0000  |  0.0000
---------------------------|---------
-0.7500    -0.2500   2.0000  |  0.5000

spoles(discout)

0.9920 + 0.0050i
0.9920 - 0.0050i
0.9986
```

Algorithm

Suppose that the continuous-time system has state-space representation

$$\text{sys} = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right].$$

If $u(t)$ is held constant over the interval $[kT,(k+1)T]$, then over that interval, the state evolution is governed by the differential equation

$$\begin{bmatrix} \dot{x} \\ \dot{u} \end{bmatrix} = \begin{bmatrix} A & B \\ 0_{n_u \times n} & 0_{n_u \times n} \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix}, \quad \begin{bmatrix} x(kT) \\ u(kT) \end{bmatrix} = \begin{bmatrix} x_k \\ u_k \end{bmatrix},$$

which captures the behavior of the continuous-time system, over one sample period, while the input $u(t)$ is held constant.

$$\text{Let } \tilde{A} := \begin{bmatrix} A & B \\ 0_{n_u \times n} & 0_{n_u \times n} \end{bmatrix}$$

Define $W \in \mathbf{R}^{(n+n_u)\times(n+n_u)}$ as $W := e^{\tilde{A}T}$. Then W appears as

$$W = \begin{bmatrix} W_{11} & W_{12} \\ 0 & I \end{bmatrix}$$

Define $A_{\text{disc}} := W_{11}$, and $B_{\text{disc}} := W_{12}$, and define the discretized system as

$$\text{discout} = \left[\begin{array}{c|c} A_{\text{disc}} & B_{\text{disc}} \\ \hline C & D \end{array}\right]$$

See Also

`exp`, `expm`, `tustin`

Purpose

Scale the independent variable values of a VARYING matrix with an affine transformation

Syntax

```
vout = scliv(vin,factor,offset)
```

Description

`scliv` scales the independent variable of a VARYING matrix in the following manner. Let $\mathtt{indv}_i$ and $\mathtt{newindv}_i$ denote the independent variable's *i*th value, before and after applying `scliv`. Then, for each i, they are related as

$$\mathtt{newindv}_i = (\mathtt{factor} \times \mathtt{indv}_i) + \mathtt{offset}$$

The default value for `offset` is zero.

Examples

Scale the independent value of vin by a factor of 3 and offset it from its original value by 0.5.

```
seeiv(vin)
1.000e+002.000e+003.000e+00 4.000e+00 5.000e+00
vout = scliv(vin,3,0.5)
seeiv(vout)
3.500e+006.500e+009.500e+00 1.250e+01 1.550e+01
```

See Also

`getiv`, `seeiv`

sdhfnorm

Purpose

`sdhfnorm` calculates the induced `norm` of a sampled-data system

Syntax

```
[gaml,gamu]=sdhfsyn(p,k,h,delay,tol)
```

Description

`sdhfnorm` is concerned with the control of a continuous-time system by a discrete-time controller. The continuous-time interconnection structure structure, `p` of type SYSTEM, has state-space realization partitioned as follows.

$$\mathrm{p} = \left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & 0 & 0 \\ C_2 & 0 & 0 \end{array}\right]$$

where the continuous-time disturbance inputs enter through B_1, the outputs from the controller are held constant between sampling instants and enter through B_2, the continuous-time errors to be kept small correspond to the C_1 partition, and the output measurements that are sampled by the controller correspond to the C_2 partition. B_2 has column size (`ncon`) and C_2 has row size (`nmeas`). Note that the D matrix is assumed to be zero.

`sdhfnorm` calculates the maximum gain from the L_2 `norm` of the disturbance inputs to the L_2 `norm` of the error outputs.

Input arguments:

`p`	SYSTEM interconnection structure matrix, (continuous-time)
`k`	discrete-time controller
`h`	sampling period
`delay`	number of samples computational delay (default = 0) (integer ≥ 0 with default =0)
`tol`	required relative accuracy

Output arguments:

`gaml`	lower bound on the norm
`gamu`	upper bound on the norm

Examples

An illustrative example is given in the "Discrete-Time and Sampled-Data H• Control" section in Chapter 3.

Algorithm

`sdhfnorm` uses variations of the formulae described in the Bamieh and Pearson paper to obtain an equivalent discrete-time system. (These variations are done to improve the numerical conditioning of the algorithms.) A preliminary step is to determine whether the `norm` of the continuous-time system over one sampling period without control is less than the given γ-value. This requires a search and is, computationally, a relatively expensive step.

Subroutines called. `dhfsyn`, `ham2schr`, and `compnorm`

Reference

Bamieh, B.A., and J.B. Pearson, "A General Framework for Linear Periodic Systems with Applications to Sampled-Data Control," *IEEE Transactions on Automatic Control,* vol. AC–37, pp. 418-–435, 1992.

See Also

`dhfsyn`, `hinfsyne`, `hinffi`, `hinfnorm`, `hinfsyn`, `h2syn`, `h2norm`, `ric_eig`, `ric_schr`

Purpose

sdhfsyn computes an H_∞ controller for a sampled-data SYSTEM interconnection matrix

Syntax

```
[k,gfin] = sdhfsyn(p,nmeas,ncon,gmin,gmax,tol,h,...
   delay,ricmethd,epr,epp)
```

Description

sdhfsyn is concerned with the control of a continuous-time system by a discrete-time controller. The continuous-time interconnection structure structure, p of type SYSTEM, has state-space realization partitioned as follows

$$\mathrm{p} = \left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & 0 & 0 \\ C_2 & 0 & 0 \end{array}\right]$$

where the continuous-time disturbance inputs enter through B_1, the outputs from the controller are held constant between sampling instants and enter through B_2, the continuous-time errors to be kept small correspond to the C_1 partition, and the output measurements that are sampled by the controller correspond to the C_2 partition. B_2 has column size (ncon) and C_2 has row size (nmeas). Note that the D matrix is assumed to be zero.

sdhfsyn synthesizes a discrete-time controller to achieve a given norm (if possible) or find the minimum possible norm to within some tolerance.

sdhfsyn provides a γ iteration using the bisection method. Given a high and low value of γ, gmax and gmin, the bisection method is used to iterate on the value of γ in an effort to approach the optimal H_∞ control design. If gmax = gmin, only one γ value is tested. The stopping criteria for the bisection algorithm requires the relative difference between the last γ value that failed and the last γ value that passed be less than tol. You can select either the eigenvalue or Schur method for solution of the Riccati equations with and without balancing. The eigenvalue method is faster but can have numerical problems, while the Schur method is slower but generally more reliable.

The algorithm employed calculates an equivalent purely discrete-time problem for each value of γ and then calls dhfsyn with $\gamma = 1$. The screen printing is then derived from the tests performed by dhfsyn.

Input arguments

`p`	SYSTEM interconnection structure matrix
`nmeas`	number of measurements output to controller
`ncon`	number of control inputs
`gmin`	lower bound on γ
`gmax`	upper bound on γ
`tol`	relative difference between final γ values
`delay`	number of samples computational delay (default = 0)
`h`	time between samples
`ricmethod`	1 Eigenvalue decomposition with balancing –1 Eigenvalue decomposition with no balancing 2 Schur decomposition with balancing (default) –2 Schur decomposition with no balancing
`epr`	measure of when a real part of an eigenvalue of the Hamiltonian matrix is zero (default `epr` = 1e–10)
`epp`	positive definite determination of the X_∞ and Y_∞ solution (default `epp` = 1e–6)

Output arguments

`k`	H_∞ (sub) optimal controller
`gfin`	final γ value achieved

You might design a first controller using the `dhfsyn` function on the SYSTEM (`samhld(p,h)`), followed by `sdhfnorm` to determine an upper bound `gmax` to use for the start of this sampled data control design iterative process.

The `sdhfsyn` program outputs several variables, which can be checked to ensure that the above conditions are being met. For each γ value the minimum magnitude, real part of the eigenvalues of the H Hamiltonian matrices is displayed along with the minimum eigenvalue of X_{∞}, which is the solution to the X Riccati equation. A # sign is placed to the right of the condition that failed in the printout. This additional information can aid you in the control design process.

Examples

An illustrative example is given in the “Discrete-Time and Sampled-Data H• Control” section in Chapter 3.

Algorithm

`sdhfsyn` uses variations of the formulae described in the Bamieh and Pearson paper to obtain an equivalent discrete-time system. (These variations are done to improve the numerical conditioning of the algorithms.) A preliminary step is to determine whether the norm of the continuous-time system over one sampling period without control is less than the given γ-value, this requires a search and is computationally a relatively expensive step.

Subroutines called. `dhfsyn`, `ham2schr`, `compnorm`

Reference

Bamieh, B.A., and J.B. Pearson, “A General Framework for Linear Periodic Systems with Applications to Sampled-Data Control,” *IEEE Transactions on Automatic Control,* vol. AC–37, pp. 418–435, 1992.

See Also

`dhfsyn`, `hinfsyne`, `hinffi`, `hinfnorm`, `hinfsyn`, `h2syn`, `h2norm`, `ric_eig`, `ric_schr`

see, seeiv

Purpose

Display a SYSTEM or VARYING matrix

Syntax

```
see(mat,iv_low,iv_high)
see(matin)
seeiv(mat)
```

Description

see displays the A, B, C, and D matrices of matin for a SYSTEM matrix or the independent variable and the matrix at that variable if matin is a VARYING matrix. iv_low and iv_high are the optional range of the independent variables to be displayed. see displays the matrix itself if the input is CONSTANT.

seeiv displays only the independent variable of the input VARYING matrix mat. An error message is displayed if the input matrix is not a VARYING matrix.

Examples

The see command displays any type of matrix. An example of its use for SYSTEM and VARYING matrices follows.

```
see(sys)
A matrix
   1   1
   1   1
press any key to move to B matrix
B matrix
   2   2
   2   2
press any key to move to C matrix
C matrix
   3   3
press any key to move to D matrix
D matrix
   4   4
sysg = frsp(sys,[0.4 0.9]);
see(sysg)
1 row2 columns
iv = 0.4
   -1.7692 - 1.1538i-1.7692 - 1.1538i
iv = 0.9
   -0.9896 - 2.2453i-0.9896 - 2.2453i
```

See Also `getiv`, `mprintf`, `rifd`, `seesys`, `sortiv`

seesys

Purpose

Display a SYSTEM or VARYING matrix with sprintf formatting

Syntax

```
seesys(matin,'format')
```

Description

seesys displays a SYSTEM matrix in packed form if matin is a SYSTEM matrix, or the independent variable and the matrix at that variable if matin is a VARYING matrix. The optional string format specifies the format exactly as in the MATLAB function sprintf. seesys is similar to see, but prints a SYSTEM matrix in packed form and gives more control over formatting. The default format is %.1e. An incorrect format string can cause erroneous output displays. See sprintf and mprintf for more details.

Examples

The seesys command displays any type of matrix. An example of its use for SYSTEM and VARYING matrices follows.

```
seesys(sys)

l.0e+00    l.0e+00  |  2.0e+00    2.0e+00
l.0e+00    l.0e+00  |  2.0e+00    2.0e+00
--------------------|---------------------
3.0e+00    3.0e+00  |  4.0e+00    4.0e+00

seesys(sys,'%1.0f')

1   1  |  2   2
1   1  |  2   2
-------|--------
3   3  |  4   4

sysg = frsp(sys, [0.4 0.9]);
seesys (sysg)
1 row                  2 columns
iv = .4
-1. 8e+00 -1. 8e+00
iv = .9
-9.9e-01 -9.9e-01
```

See Also

getiv, mprintf, rifd, see, sortiv

Purpose

Select rows/outputs and columns/inputs from a CONSTANT, SYSTEM or VARYING matrix or reorder the SYSTEM states

Syntax

```
out = sel(mat,rows,cols)
out = sel(sys,outputs,inputs)
sysout = reordsys(sys,index)
```

Description

sel selects desired rows and columns from a CONSTANT/VARYING matrix, or outputs and inputs from a SYSTEM matrix. For CONSTANT and VARYING matrices, the rows and cols input arguments are row vectors with the desired rows/columns of mat specified. For SYSTEM matrices, outputs and inputs are row vectors with the desired inputs/outputs specified. Use the string ':' to specify all rows (inputs) and/or columns (outputs).

reordsys reorders the states of SYSTEM matrix sys as defined by the vector of position variables, index. The index variable is restricted to be the same length as the number of states of sys. This command can be used in conjunction with strans and sresid to reduce the states of a SYSTEM matrix.

Examples

You can use the sel command with any matrix type. First, construct and display a one state, two output, three input SYSTEM matrix.

```
minfo(sys)
system:1 states2 outputs3 inputs
seesys(sys)

1.0e+00  |  2.0e+00   3.0e+00   4.0e+00
---------|------------------------------
5.0e+00  |  6.0e+00   7.0e+00   8.0e+00
9.0e+00  |  1.0e+01   1.1e+01   1.2e+01
```

Reorder the outputs of sys to be output 2, 1 and repeat output 2; also reorder the inputs to be input 3, 1 and 2.

```
sys2 = sel(sys,[2 1 2],[3 1 2]);
minfo(sys2)
system:1 states3 outputs3 inputs
seesys(sys2)

1.0e+00  |  4.0e+00   2.0e+00   3.0e+00
---------|-----------------------------
9.0e+00  |  1.2e+01   1.0e+01   1.1e+01
5.0e+00  |  8.0e+00   6.0e+00   7.0e+00
9.0e+00  |  1.2e+01   1.0e+01   1.1e+01
```

The same manipulations can be done on a VARYING matrix.

```
sysg =frsp(sys,logspace(-1,0,2));
see(sysg)
2 rows3 columns
indep variable 0.1

–3.9010 – 0.9901i    –7.8515 – 1.4851i   –11.8020 – 1.9802i
–7.8218 – 1.7822i   –15.7327 – 2.6733i   –23.6436 – 3.5644i

indep variable 1

1.0000 – 5.0000i   –0.5000 –  7.5000i   –2.0000 – 10.0000i
1.0000 – 9.0000i   –2.5000 – 13.5000i   –6.0000 – 18.0000i
```

Select the second and first outputs and the third and second inputs and display them.

```
part = sel(sysg,[2 1],[3 2]);
see(part)
2 rows2 columns
indep variable 0.1

–23.6436 – 3.5644i   –15.7327 – 2.6733i
–11.8020 – 1.9802i    –7.8515 – 1.4851i
```

```
indep variable 1

–6.0000 – 18.0000i    –2.5000 – 13.5000i
–2.0000 – 10.0000i    –0.5000 –  7.5000i
```

The command reordsys interchanges states in a SYSTEM matrix. The matrix (sys) has four states, two inputs, and one output and it is to be reordered so that states three and four interchange with states one and two in the state-space system.

```
minfo(sys)
system: 4 states1 outputs2 inputs
seesys(sys)

1.5e-01   -1.3e-01   0.0e+00   0.0e+00  |  2.4e-01   2.3e-01
1.3e-01    1.5e-01   0.0e+00   0.0e+00  | -4.4e-01  -2.2e-01
0.0e+00    0.0e+00  -6.2e-01   0.0e+00  | -6.5e-01  -3.9e-01
0.0e+00    0.0e+00   0.0e+00   2.0e+00  |  1.2e+00   7.2e-01
----------------------------------------|--------------------
-9.7e-02  -1.2e-01  -1.3e-01   1.1e+00  |  0.0e+00   0.0e+00

sysout = reordsys(sys,[3 4 1 2]);
minfo(sysout)
system: 4 states1 outputs2 inputs
seesys(sysout)

-6.2e-01    0.0e+00   0.0e+00  0.0e+00  |  -6.5e-01  -3.9e-01
 0.0e+00    2.0e+00   0.0e+00  0.0e+00  |   1.2e+00   7.2e-01
 0.0e+00    0.0e+00   1.5e-01 -1.3e-01  |   2.4e-01   2.3e-01
 0.0e+00    0.0e+00   1.3e-01  1.5e-01  |  -4.4e-01  -2.2e-01
---------------------------------------|---------------------
-1.3e-01   1.1e+00  -9.7e-02 -1.2e-01  |   0.0e+00   0.0e+00
```

See Also

getiv, sresid, strunc, unpck, vunpck

siggen

Purpose

Generate VARYING matrix functions

Syntax

```
y = siggen('function(t)',t)
```

Description

siggen is a general purpose signal generator. You can provide a timebase with the argument t, and the function to be evaluated with the first argument (a string), function(t). The output, y, will be a VARYING matrix. t could also be VARYING, in which case the timebase is the independent variables contained in t.

function(t) is not necessarily dependent on t. In the cases where it doesn’t depend on t, siggen can be slow. This is because function is evaluated with a MATLAB eval call for every element in t. For example, consider generating a random vector. The command

```
u = siggen('rand(size(t))',[0:100]);
```

generates a 1 × 1 VARYING matrix with 101 independent variables. This is very fast because it depends on t. To a 2 × 1 VARYING matrix of random values, you can use the command

```
u = siggen('rand(2,1)',[0:100]);
```

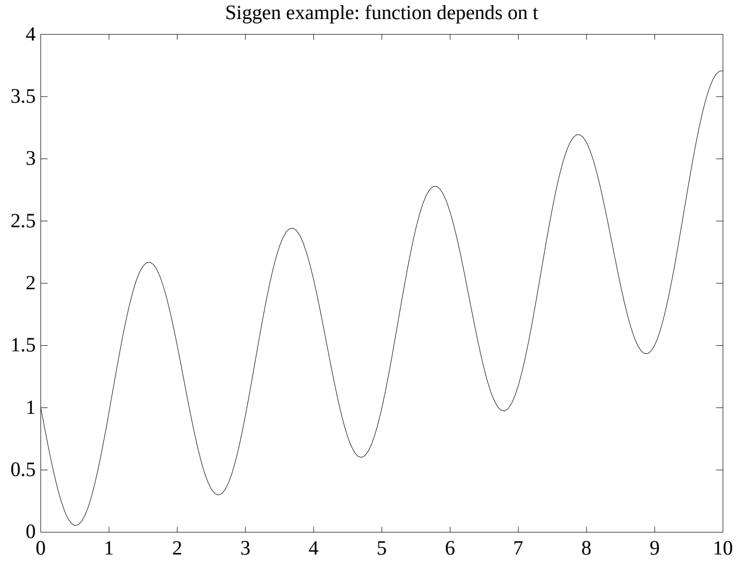
This is slow because 101 evaluations of rand(2,1) are performed in a MATLAB for loop. For vectors of random signals, a much faster alternative is to use vpck with a random matrix of the appropriate size. In the above example this would be

```
u = vpck(rand(202,1),[0:100]);
```

Examples

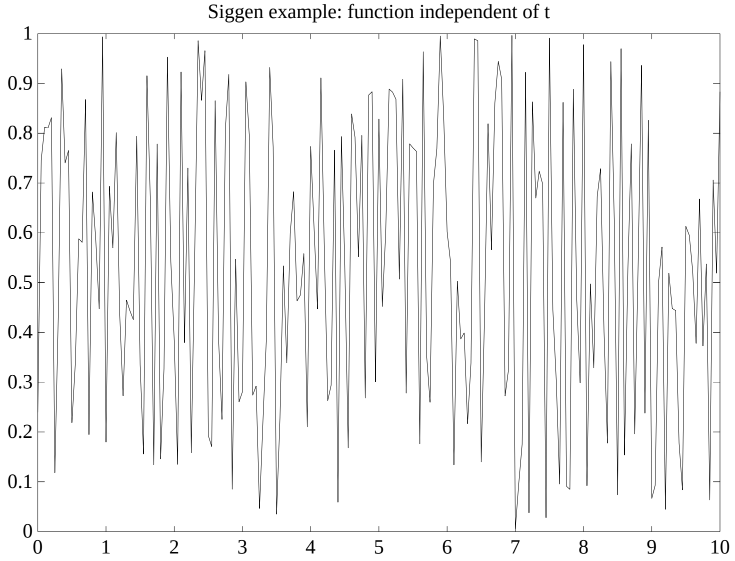
The first example illustrates what is perhaps the most common use of siggen. A single-input single-output signal is created from MATLAB mathematical functions. It is important to use t as the independent variable in the function string.

```
timebase = [0:0.05:10];
y1 = siggen('exp(0.1*t) - sin(3*t)',timebase);
minfo(y1)
varying:201 pts1 rows1 cols
vplot(y1)
title('siggen example: function depends on t')
```



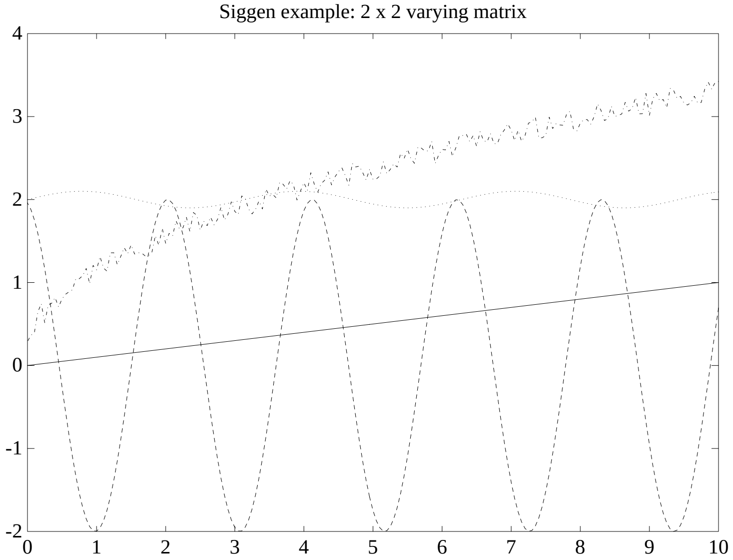
The second example illustrates that the second argument can make use of the independent values of a VARYING matrix. Note also that the specified function is independent of `t`, and is executed at each instance of `t`. This example is included to illustrate that the function string need not depend on `t`. In practice the string `rand(size(t))` is orders of magnitude faster than `rand`.

```
y2 = siggen('rand',y1);
minfo(y2)
varying:201 pts1 rows1 cols
vplot(y2)
title('siggen example: function independent of t')
```



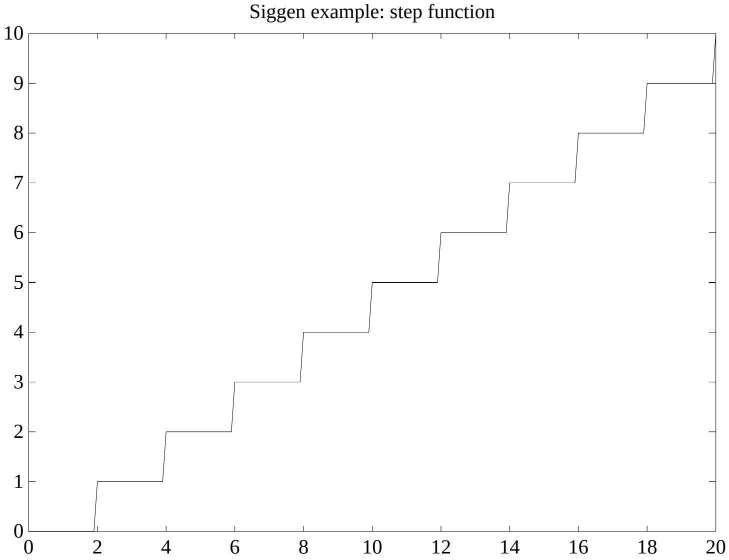
The use of siggen is not restricted to single-input, single-output signals. The following example creates a 2 × 2 VARYING matrix. In this example the matrix elements are all functions of t and the results plotted in the figure. They could equally well have all been independent of t, for example 'rand(3,2)' could be the function string. (Note that they must all be one type or the other (this restriction does not apply if t is a single number).

```
func = '[t/max(t) 2*cos(3*t+0.2); 2+0.1*sin(2*t) ';
func = [func ' sqrt(t)+0.3*rand(size(t))]'];
y3 = siggen(func,timebase);
minfo(y3)
varying:201 pts2 rows2 cols
vplot(y3)
title('Siggen example: 2 x 2 varying matrix')
```



siggen cannot generate stair-step signals with user-specified values. You can do this however (for single-input/single-output signals) using the command vpck or the μ-Tools commmand step_tr. Vectors of signals can be created with vpck and abv. The following example demonstrates the use of vpck. vinterp is used to plot a meaningful representation of the signal.

```
y4 = vpck([0:10]',[0:2:20]');
minfo(y4)
varying:11 pts1 rows1 cols
vplot(vinterp(y4,0.1))
title('Siggen example: step function')
```



See Also `cos_tr`, `sin_tr`, `step_tr`, `vpck`, `vinterp`

Purpose

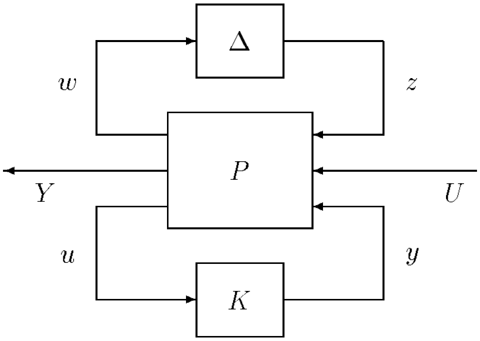
A graphical user interface for time simulations of linear fractional transformations

Syntax

```
simgui
```

Description

`simgui` provides the ability to simulate linear fractional models and plot their responses. The standard linear fractional model considered is shown below.



Standard Linear Fractional Model

The *P* block corresponds to the open-loop plant interconnection model, referred to as Plant in `simgui`. The *K* block corresponds to a state-space controller, Controller in `simgui`. The Δ block corresponds to the perturbation to the model, Perturbation in `simgui`. The Input Signal to the time simulation is denoted by *U* in the figure. The individual systems are formed using the `starp` command. The time simulation outputs available for plotting correspond to the variable *Y*. Three types of simulation are possible: continuous-time using `trsp`, discrete-time using `dtrsp`, and sample-data using `sdtrsp`.

`simgui` has two interface windows and up to six plot windows:

- Main **Simulation** window, which is the main interface for the user.
- **Parameter** window, which is used to modify properties of the time simulation, such as the final time, integration step size, initial conditions, and which variables are automatically exported to the workspace.
- **Plot** windows, where the plots of time responses are displayed. You can open up to six of these windows.

A detailed description of the use and application of `simgui` is provided in the "LFT Time Simulation User Interface Tool: simgui" section in Chapter 6.

Examples

An example of using `simgui` for simulation of a linear fractional transformation is shown in the "LFT Time Simulation User Interface Tool: simgui" section in Chapter 6.

See Also

`dtrsp`, `sdtrsp`, `starp`, `trsp`, `vplot`

Purpose

Calculate the eigenvalues of a SYSTEM A matrix

Syntax

```
out = spoles(sys)
```

Description

spoles returns the eigenvalues of the A matrix from the SYSTEM matrix sys.

Examples

Find the poles of the two input, one output, three state SYSTEM matrx sys.

```
A = [1 1 1; 3 1 1; 1 1 -2];
B = 2*ones(3,2);
C = 3*ones(1,3);
D = 4*ones(1,2);
sys = pck(A,B,C,D);
minfo(sys)
system: 3 states1 outputs2 inputs
spoles(sys)
   3.1474
   -0.8186
   -2.3289
eig(A)
ans =
   3.1474
   -0.8186
   -2.3289
```

Algorithm

spoles uses the MATLAB command schur to find the eigenvalues of the SYSTEM A matrix. This is a more numerically reliable method than using the eig function.

See Also

eig, rifd, schur, szeros

srelbal, sfrwtbal, sfrwtbld, sncfbal, sdecomp

Purpose

srelbal calculates the stochastically balanced realization of a SYSTEM matrix, sfrwtbal, the frequency weighted balanced realization for performing relative error, and sfrwtbld, the frequency weighted model order reduction. sncfbal calculates the normalized coprime factorizations. sdecomp decomposes a SYSTEM matrix as the sum of stable and unstable systems.

Syntax

```
[sysb,relsv,sysfact] = srelbal(sys,tol)
[sys1,sig1] = sfrwtbal(sys,wt1,wt2)
syshat = sfrwtbld(sys1hat,wt1,wt2)
[sysnlcf,signcf,sysnrcf] = sncfbal(sys,tol)
[sysst,sysun] = sdecomp(sys,bord,fl)
```

Description

srelbal performs a stochastically balanced realization of the input SYSTEM matrix. sys must be stable and be of full column rank at infinite frequency, but not necessarily square or minimum phase. Difficulties may occur if sys has zeros on the imaginary axis. sysb will have the same transfer function as sys, and sysfact gives the stable minimum phase system such that sys~ sys = sysfact sysfact~. Note that sys~ denotes cjt(sys). If [a,b,c,d] = unpck(sysb) and [af,bf,cf,df] = unpck(sysfact), then the realization [a,bf,c,0] will be balanced with Hankel singular values relsv, and will also equal the stable part of sys*sysfact~$^{-1}$. A reduced-order system can be obtained by strunc(sysb,k) that will have guaranteed performance in the relative error.

sfrwtbal performs a frequency-weighted balancing. It calculates the stable part of (wt1) ~$^{-1}$ * sys* (wt2)~$^{-1}$ and sys1 is a balanced realization of this, with Hankel singular values sig1. wt1 and wt2 must be stable and minimum phase, square and of compatible dimensions with sys. wt2 has the identity as default value. sys must be stable. The resulting system sys1 can then be approximated by sys1hat of order k using, for example, hankmr, and an approximation syshat to sys is obtained by sfrwtbld, which calculates the stable part of (wt1)~ *sys1hat*(wt2)~ using sdecomp.

A general lower bound on the frequency weighted approximation error is given by

$$(\mathrm{wt1})\ \sim^{-1}(\mathrm{sys}\ -\ \mathrm{syshat})(\mathrm{wt2})\sim^{-1}\|_{\infty} \geq \mathrm{sig1}(k + 1)$$

where in the relative error case wt1 is the identity and wt2 = sysfact.

sncfbal produces balanced realizations of the normalized left and right coprime factorizations of a SYSTEM matrix. That is for a transfer function G, balanced state-space realizations are calculated for $\begin{bmatrix} N_l & M_l \end{bmatrix}$ and $\begin{bmatrix} N_r \\ M_r \end{bmatrix}$, where $N_l N_l^{\sim} + M_l M_l^{\sim} = I$, $N_r N_r^{\sim} + M_r M_r^{\sim} = I$, $G = M_l^{-1} N_l = N_r M_r^{-1}$, and N_l, M_l, N_r, and M_r are all stable. The Hankel singular values of both $\begin{bmatrix} N_l & M_l \end{bmatrix}$ and $\begin{bmatrix} N_r \\ M_r \end{bmatrix}$ are given by the column vector signcf. Model reduction for these systems can then be performed using strunc or hankmr. The method is well suited to plant or controller reduction in feedback systems.

sdecomp decomposes a system into the sum of two systems, sys = madd(sysst,sysun). sysst has the real parts of all its poles < bord and sysun has the real parts of all its poles ≥ bord. bord has default value 0. The D matrix for sysun is zero unless fl = 'd' when that for sysst is zero.

srelbal, sfrwtbal , sfrwtbld, sncfbal, and sdecomp are restricted to be used on continuous-time SYSTEM matrices.

Examples

Given the system $\text{sys} = \dfrac{(s+1)(s+10)(s+90)}{(s+2)(s+91)(s+100)}$ reduce the system to two and one states, respectively. An approximate system of order 1 or 2 can be obtained as follows.

```
sys = zp2sys([-1 -10 -90],[-2 -91 -100]);
[sysb,relsv,sysfact] = srelbal(sys);
disp(relsv')
   8.5985e-012.0777e-012.1769e-04
sysrel1 = strunc(sysb,1);
sysrel2 = strunc(sysb,2);
```

The relative error in the second-order model will be negligible since relsv(3) is very small; however, with a first-order model, it will be substantial.

The reduced-order models of order k can be obtained in the frequency weighted case as follows.

```
wt1 = nd2sys([1 10],[1 1]);
[sys1,sig1] = sfrwtbal(sys,wt1);
disp(sig1');
   4.1873e-014.6472e-031.0280e-04
sys1hat = hankmr(sys1,sig1,1,'d');
syshat = sfrwtbld(sys1hat,wt1);
disp(hinfnorm(mmult(msub(sys,syshat),minv(wt1))));
   4.6471e-034.6517e-03Inf
```

In this example the method nearly reaches the lower bound, but this cannot be claimed in general.

Now consider approximating the unstable third order system,

$$\text{sys} = \frac{10}{s(s-1)(s+10)}$$

using sncfbal. First the balanced realization of the normalized left coprime factors is calculated, then this is truncated to two states and the reduced-order system recovered from these normalized coprime factors using starp.

```
sys = zp2sys([],[0 1 -10],10);
[sysnlcf,signcf] = sncfbal(sys);
disp(signcf')
   9.6700e-015.2382e-012.3538e-02
sysnlcfr = strunc(sysnlcf,2);
sysr = starp(mmult([1;1],msub(sysnlcfr,[0 1])),-1,1,1)
```

If this is the transfer function of a plant to be controlled, then signcf(1) can be used to predict the possible robust stability to perturbations in the coprime factors, and the potential for model-order reduction of the controller is given by signcf(2:3), McFarlane and Glover (1989). In this example the maximum stablizable perturbations in the coprime factors is given by $\sqrt{1-\textbf{signcf(1)}^2} = 0.20770$. Furthermore if a controller is designed to be optimal for the second-order reduced model, then its stability margin will be at least

```
0.25477 - 2 * signcf(3) = 0.20770
```

Algorithm

The algorithms are based on the results in the following papers.

Reference

Anderson, B.D.O., and Y. Liu, “Controller reduction: Concepts and Approaches,” *IEEE Transactions on Automatic Control*, vol. AC–34, pp. 802–812, 1989.

Desai, U.B., and D. Pal, “A transformation approach to stochastic model reduction,” *IEEE Transactions on Automatic Control*, vol. AC–29, pp. 1097–1100, 1984.

Glover, K., “Multiplicative approximation of linear multivariable systems with error bounds,” *Proceedings of the American Control Conference*, Seattle, pp. 1705–1709, 1986.

Latham, G.A., and B.D.O. Anderson, “Frequency-weighted optimal Hankel norm approximation of state transfer functions,” *Systems and Control Letters*, vol. 5, pp. 229–236, 1985.

McFarlane, D.C., and K. Glover, *Robust Controller Design using Normalised Coprime Factor Plant Descriptions*, Springer-Verlag, Lecture Notes in Control and Information Sciences, vol. 138, 1989.

Wang, W., and M.G. Safonov, “A tighter relative error bound for balanced stochastic truncation,” *Systems and Control Letters*, vol. 14, pp. 307–317, 1990.

See Also

`hankmr`, `sysbal`, `sresid`, `strunc`

sresid, strunc

Purpose

Reduce the state dimension of a SYSTEM matrix

Syntax

```
sysout = sresid(sys,ord)
sysout = strunc(sys,ord)
```

Description

`sresid` residualizes the last states of a SYSTEM matrix `sys`. `sresid` accounts for the DC contribution of the last columns and rows of the SYSTEM A matrix and the corresponding rows and columns of `B` and `C`. `sresid` assumes that the SYSTEM matrix is ordered so that the last states are to be residualized. If the orignal SYSTEM matrix is partitioned as

$$\mathrm{p} = \left[\begin{array}{c|cc} A_{11} & A_{12} & B_1 \\ \hline A_{21} & A_{22} & B_2 \\ C_1 & C_2 & D \end{array}\right]$$

with A_{11} of size `ord` × `ord`, then the command

```
sysout = sresid(sys,ord)
```

results in

$$\mathrm{sysout} = \mathrm{pss2sys}\left(\begin{bmatrix} A_{11} & B_1 \\ C_1 & D \end{bmatrix} - \begin{bmatrix} A_{12} \\ C_2 \end{bmatrix} A_{22}^{-1} \begin{bmatrix} A_{21} & B_2 \end{bmatrix}, \ \mathbf{ord}\right)$$

`strunc` truncates the states of the input system matrix `sys`, to a system with state dimension equal to `ord`. `strunc` can be used in conjunction with the model reduction routines `sysbal` and `hankmr`.

The resulting SYSTEM output matrix is

```
sysout = pss2sys ([A_11 B_1; C_1 D]);
```

Examples

A two input, one output, four state SYSTEM is reduced down to a two input, one output, two state SYSTEM via `sresid` and `strunc`. The only difference between the two reduced-order systems is the value of their *D* matrices.

```
seesys(sys)
-1.2e-01   0.0e+00   0.0e+00   0.0e+00 | 9.le-01   5.2e-01
 0.0e+00  -3.2e-01   0.0e+00   0.0e+00 | 6.le-02   3.2e-01
 0.0e+00   0.0e+00  -4.3e+00   0.0e+00 | 9.le-01   9.9e-01
 0.0e+00   0.0e+00   0.0e+00  -9.9e+01 | 5.le-01   4.9e-01
---------------------------------------|-------------------
 2.7e-01   9.le-02   9.5e-01   7.4e-02 | 0.0e+00   0.0e+00


sys_strunc = strunc(sys,3);
seesys(sys_strunc)
-1.2e-01    0.0e+00   0.0e+00  |  9.le-01   5.2e-01
 0.0e+00   -3.2e-01   0.0e+00  |  6.le-02   3.2e-01
 0.0e+00    0.0e+00  -4.3e+00  |  9.le-01   9.9e-01
-------------------------------|---------------------
 2.7e-01    9.le-02   9.5e-01  |  0.0e+00   0.0e+00

sys_resid = sresid(sys,3)
seesys(sys_resid)

-1.2e-01   0.0e+00   0.0e+00  |  9.le-01   5.2e-01
 0.0e+00  -3.2e-01   0.0e+00  |  6.le-02   3.2e-01
 0.0e+00   0.0e+00  -4.3e+00  |  9.le-01   9.9e-01
------------------------------|--------------------
 2.7e-01   9.le-02   9.5e-01  |  3.8e-04   3.7e-04

sysstrunc = strunc(sys,3); seesys(sysstrunc)
sysresid = sresid(sys,3); seesys(sysresid)
```

See Also

`rifd`, `statecc`, `strans`

starp

Purpose

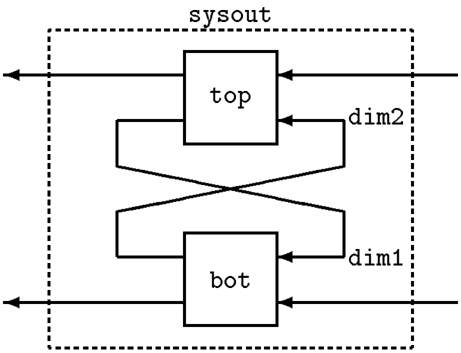
Form the Redheffer star product of two VARYING/SYSTEM/CONSTANT matrices. The star product is a generalization of a linear fractional transformation

Syntax

```
sysout = starp(top,bot,dim1,dim2)
```

Description

Connects the two matrices top and bot in the star product loop shown below.



The last `dim1` outputs of `top` are fed to the first `dim1` inputs of `bot`, and the first dim2 outputs of `bot` are fed into the last `dim2` inputs of `top`. The remaining inputs and outputs constitute `sysout`. By this description, the dimensions must satisfy

min(dim_out(top),dim_in(bot)) ≥ `dim1`

min(dim_out(bot),dim_in(top)) ≥ `dim2`

Further restrictions also arise

`IF`	`dim1 = dim_out(top)  &  dim2 = dim_out(bot)`
`THEN`	there are no outputs remaining in the interconnection
`IF`	`dim1 = dim_in(bot)  &  dim2 = dim_in(top)`
`THEN`	there are no inputs remaining in the interconnection

In either case, it is unclear what to return as sysout, so it is returned empty. There is one exception to this situation. If either top or bot is a SYSTEM matrix with a nonzero number of states, and all of the equalities in the both of the above IF conditions hold (hence there are no inputs or outputs in the interconnection), then sysout will be a CONSTANT matrix, and will be the A matrix governing the internal dynamics of the loop.

As usual, the only types of matrices that cannot be combined are SYSTEM matrices with VARYING matrices.

If only two arguments are given (i.e., no dimensions specified),

```
out = starp(top,bot);
```

then the operation is equivalent to

```
dim1 = min(ynum(top),unum(bot));
dim2 = min(unum(top),ynum(bot));
out = starp(top,bot,dim1,dim2);
```

Algorithm

The “m-Tools Commands for LFTs” section in Chapter 4 provides details of the star product formulae.

Reference

Redheffer, R., “Inequalities for a matrix Riccati equation,” *Journal of Mathematics and Mechanics*, vol. 8, no. 3, 1959.

See Also

sysic

statecc, strans

Purpose

Apply state-coordinate transformation to a SYSTEM matrix

Syntax

```
sysout = statecc(sysin,t)
[sysout,v] = strans(sys)
```

Description

statecc applies a state coordinate transformation to the matrix, yielding a new SYSTEM matrix with

```
sysout = pck(tA*t,tB,C*t,D)
```

where A, B, C, and D are the state-space entries of the matrix sysin. t is restricted to be square and have the same dimensions as the A matrix.

strans transforms the A matrix of sys in bidiagonal form with the complex conjugate roots in real 2 × 2 form. sysout contains the transformed SYSTEM matrix and v is the transformation matrix. The A matrix is ordered by increasing magnitude of its eigenvalues. strans calls the MATLAB eig command to do the reordering.

Note strans may be inaccurate when a SYSTEM/CONSTANT matrix has repeated eigenvalues. This is due to the potential defective eigensystem, and the lack of a full set of eigenvectors.

Examples

The strans command shows the individual contributions of the modes of the SYSTEM matrix. In this example sys, which has four states, two inputs and one output is transformed into bidiagonal form.

```
see(sys)
A matrix

0.2190   0.9347   0.0346   0.0077
0.0470   0.3835   0.0535   0.3834
0.6789   0.5194   0.5297   0.0668
0.6793   0.8310   0.6711   0.4175
```

```
B matrix

0.6868   0.5269
0.5890   0.0920
0.9304   0.6539
0.8462   0.4160

C matrix

0.7012      0.9103    0.7622    0.2625

D matrix

0      0

sys=strc
see(sys)

A matrix

-0.0763          0           0         0
      0     0.1082     –0.4681         0
      0          0           0    1.4095

B matrix

-0.4731    -0.1839
 0.5971     0.3199
 0.2869     0.5542
-1.7033    -0.8132
C matrix

–0.1150     0.0298    0.3214     –1.0477

D matrix

0      0
```

See Also

eig, sclin, sclout, veig

Purpose

Calculate the balanced realization and optimal Hankel norm approximation of a SYSTEM matrix. sysbal and hankmr are restricted to be used on continuous-time SYSTEM matrices whose poles have negative real part

Syntax

```
[sysb,hanksv] = sysbal(sys,tol)
[sysred,sysanti,siganti] = hankmr(sysb,hanksv,k,opt)
```

Description

sysbal performs a truncated balanced realization of the input SYSTEM matrix. The result is truncated to retain all Hankel singular values greater than tol. If tol is omitted then it is set to

$$\max(\mathrm{hanksv}(1) * 1.0^{-12}, 1.0^{-16})$$

The second output argument of sysbal is the vector hanksv, which contains the Hankel singular values of the input system, sys. One method to get a reduced-order model is to truncate the balanced system sysb using strunc.

hankmr returns sysred, the optimal Hankel norm approximation of order k to the SYSTEM matrix, sysb, which is a balanced realization with Hankel singular values hanksv and is of order n (n > k). The fourth optional input argument, opt, may be omitted in which case sysanti contains the anti-causal term such that the $L\infty$ norm of (sysb - sysred - sysanti) is hanksv(k+1) or set to

'a' when sysout also includes the anti-casual term, and sysanti=0

'd' when sysout includes a D matrix to reduce the H_∞ error norm of (sys – sysout)

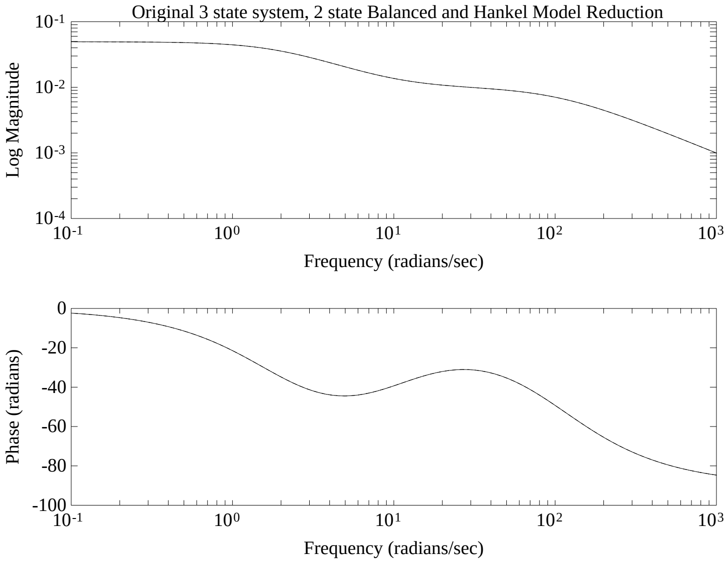
If the d option is set, the third output argument of hankmr is the vector siganti, which contains the Hankel singular values of the system, sysanti~. In this case

$$\mathrm{hanksv}(k+1) \le \|\mathrm{sys} - \mathrm{sysred}\|_\infty \le \mathrm{hanksv}(k+1) + \mathrm{sum}(\mathrm{siganti})$$

Examples

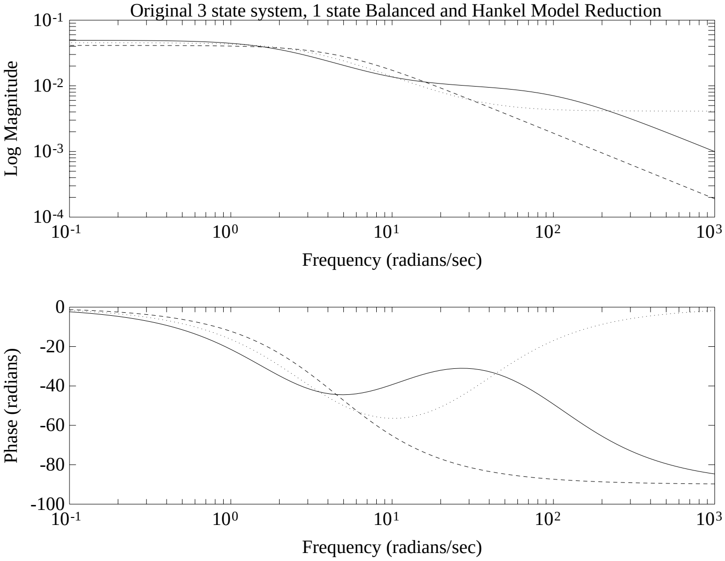
Given the system **sys** $= \dfrac{(s+10)(s+90)}{(s+2)(s+91)(s+10)}$, reduce the system to two and one states, respectively. First reduce the system to two states.

```
sys = zp2sys([-10 -90],[-2 -91 -100]);
w = logspace(-1,3,100);
sysg = frsp(sys,w);
[syssb,sv]=sysbal(sys);
sv
ans =
2.0613e-024.1136e-031.2663e-06
sys2s = strunc(syssb,2);
sys2sg = frsp(sys2s,w);
sys2h = hankmr(syssb,sv,2);
sys2hg = frsp(sys2h,w);
vplot('bode',sys_g,sys2sg,sys2hg)
tmp = 'Original 3 state system, 2 state Balanced ';
tmp1 = 'and Hankel Model Reduction';
title([tmp tmp1])
```



Notice that there is virtually no difference between the three systems. Now we will reduce the original system down to one state with `sysbal` and `hankmr`.

```
sys1s = strunc(syssb,1);
sys1sg = frsp(sys1s,w);
sys1h = hankmr(syssb,sv,1);
sys1hg = frsp(sys1h,w);
[syssb,sv] = sysbal(sys);
vplot('bode',sys_g,sys1sg,sys1hg)
tmp = 'Original 3 state system, 1 state Balanced '
tmp1 = 'and Hankel Model Reduction')
title([tmp tmp1])
```



The original three state system corresponds to the solid line, the one state balanced realization system corresponds to the dashed line, and the one state Hankel model reduced system corresponds to the dotted line. There is significant differences between the models and the two model reduction techniques. Depending on the model reduction objectives, the one state models may be inappropriate for use.

Reference Glover, K. “All optimal Hankel-norm approximations of linear multivariable systems and their error bounds,” *International Journal of Control*, vol. 39, pp. 1115–1193, 1984.

See Also `sdecomp`, `sfrwtbal`, `sfrwtbld`, `sresid`, `srelbal`, `sresid`

Purpose

Form linear interconnections of CONSTANT and SYSTEM matrices (or CONSTANT and VARYING matrices)

Syntax

```
sysic
```

Description

μ-Tools provides a simple linear system interconnection program called `sysic`. It forms linear interconnections of CONSTANT and SYSTEM matrices (or CONSTANT and VARYING matrices, though this can require a lot of memory), by writing the loop equations of the interconnection.

Using `sysic` involves setting up several variables in the MATLAB workspace, and then running the M-file `sysic`. The variables that are defined delineate the details of the interconnection.

Variable Descriptions

A list and description of the variables required by `sysic` follow.

systemnames. This variable is a character string, which contains the names of the matrices used in the interconnection. The names must be separated by spaces and/or tabs, and there should be no additional punctuation. Each named system must exist in the MATLAB workspace at the time the program `sysic` is run. The SYSTEM matrices names used within the `sysic` program are limited to 10 characters. This limitation is due to the MATLAB 19 character limitation on the workspace variable names. That is, a SYSTEM matrix named `andygaryjohnkeithroy` would be invalid.

inputvar. This variable is a character string, with names of the various external inputs that are present in the final interconnection. The input names are separated by semicolons, and the entire list of input names is enclosed in square brackets [ ]. Inputs can be multivariable signals; for instance a windgust input, with three directions (x, y, and z) that can be specified by using `windgust{3}`. This means that there is a three variable input to the interconnection called `windgust`. Alternatively, this could be specified as three separate, scalar inputs, say `wingustx`, `windgusty`, and `windgustz`. The order that the input names appear in the variable `inputvar` is the order that the inputs will be placed in the interconnection.

outputvar. This variable is a character string, describing the outputs of the interconnection, which *must be linear combinations of the subsystem outputs and the external inputs.* Semicolons are used to separate the channels of the output variables. Between semicolons, signals can be added and subtracted, and multiplied by scalars. For multivariable subsystems, arguments within parentheses specify which subsystem outputs are to be used and in what order. For instance plant `(2:5,8,1,9:11)` specifies outputs `2,3,4,5,8,1,9,10,11` from the system plant. If no arguments are specified with a system, then it is assumed that all outputs are being used, and in the order they appear in that system.

input_to_sys. Each subsystem named in the variable `systemnames` must have a variable set to define the inputs to the subsystem. If the system name is `controller`, then the variable that must be set should be called `input_to_controller`. It is specified in the same manner that the variable `outputvar` is set, with inputs consisting of linear combinations of subsystem outputs and external inputs. Separate channels are separated by semicolons, and the order of the inputs in the variable should match the order of the inputs in the system itself.

sysoutname. This character string variable is optional. If it exists in the MATLAB workspace when `sysic` is run, then the interconnection that is created by running `sysic` is placed in a MATLAB variable whose name is given by the string in sysoutname. If this variable does not exist in the workspace, then the interconnection is automatically placed in the variable `ic_ms`.

cleanupsysic. After running `sysic`, all of the above variables, which describe the interconnection, are left in the workspace. These will be automatically cleared if the optional variable `cleanupsysic` is set to the character string yes. The default value of the variable is `'no'` which does not result in any of the `sysic` descriptions you defined to be cleared. The MATLAB matrices listed in the variable `systemnames` are never automatically cleared.

Running sysic

If the variables `systemnames`, `inputvar`, and `outputvar` are set, and for each name `name_i` appearing in systemnames, the variable `input_to_name_i` is set, then the interconnection is created by running the M-file `sysic`. Depending on the existence/nonexistence of the variable `sysoutname`, the resulting interconnection is stored in your specified MATLAB variable, or the default MATLAB variable ic_ms.

Within sysic, a series of error-checking routines monitor the consistency and availability of system matrices and their inputs. These routines provide a basic level of error detection to aid you in debugging. The input/output dimensions of the final interconnection are defined by inputvar and outputvar variables.

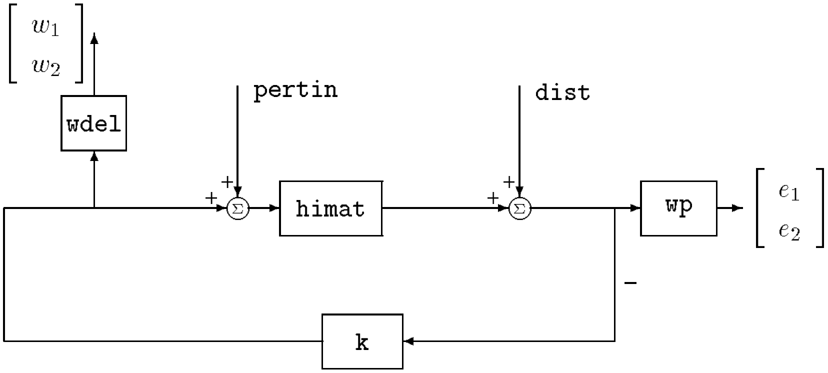
The syntax of sysic is limited, and for the most part is restricted to what is shown here. Some additional features are illustrated in the more complicated demonstration problems. Note that you must keep track of input/output variables defined for the final interconnection structure.

Examples

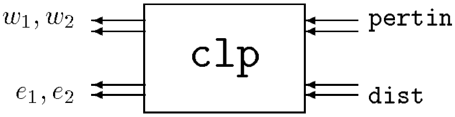
The HIMAT example provides another example of how to construct interconnection systems from block diagram descriptions. The interconnection diagram below corresponds to the HIMAT design example.

Given that there are four SYSTEM matrices, named himat, wdel, wp, and k, in the MATLAB workspace, each with two inputs, and two outputs. The following 10 lines form the sysic commands to make the interconnection structure shown below, which is placed in the variable clp. You can execute these at the command line (as shown) or type them into an M-file. (Note that to run this example you must create the variables himat, wdel, wp and k.)

```
systemnames = ' himat wdel wp k ';
inputvar = '[ pertin{2};dis{2}]';
outputvar = '[ wdel ;wp ]';
input_to_himat = '[ k + pertin ]';
input_to_wp = '[ dist + himat ]';
input_to_wdel = '[ k ]';
input_to_k = '[ -dist - himat ]';
sysoutname = 'clp';
cleanupsysic = 'yes';
sysic;
```


$\begin{bmatrix} w_1 \\ w_2 \end{bmatrix}$
wdel
pertin
dist
+
+
+
+
himat
wp
$\begin{bmatrix} e_1 \\ e_2 \end{bmatrix}$
–
k

The final interconnection structure is located in clp with two sets of inputs, `pertin` and `dist`, and two sets of outputs *w* and *e*, corresponding to the perturbation and error outputs.



See Also

`abv`, `madd`, `daug`, `mmult`, `sbs`, `sel`, `starp`

Purpose

Transmission zeros of a SYSTEM matrix

Syntax

```
veczeros = szeros(sys,epp)
```

Description

szeros calculates the transmission zeros of the input SYSTEM matrix, sys. The output veczeros contains the vector of transmission zeros.

epp is an optional input argument which is used to test the closeness of the generalized eigenvalues of the randomly perturbed matrices. Its default value is the machine epsilon. Occasionally zeros at infinity are displayed as very large values due to numerical accuracy problems.

For a square SYSTEM matrix, `[A B; C D]`, the generalized eigenvalue test consists of finding the roots of

$$\det\left(\begin{bmatrix} \mathbf{A}-\lambda I & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}\right) = 0$$

Algorithm

For a square system, the transmission zeros are found via the generalized eigenvalue problem described above. To solve for the transmission zeros of a nonsquare SYSTEM matrix, additional random rows or columns are augmented to the SYSTEM matrix to make it square and the corresponding zeros are found. This is done twice, and the unchanged generalized eigenvalues, where the difference between the eigenvalues is less than epp, are considered to be the transmission zeros of the SYSTEM matrix.

Reference

Laub, A.J., and B.C. Moore, “Calculation of transmission zeros using QZ techniques,” *Automatica*, vol. 14, pp. 557–563, 1978.

See Also

spoles

Purpose

Compute the time response of linear system

Syntax

```
y = trsp(sys,u,tfinal,int,x0)
y = dtrsp(dsys,u,T,tfinal,x0)
[output,y,u] = sdtrsp(sys,k,input,T,tfinal,int,x0,z0)
```

Description

trsp computes the time response of the continuous-time system, sys, with the input, u. The input, u, is a VARYING matrix, which contains the input signal vector at certain points in time. The input can be irregularly spaced in the independent variable or a constant, in which case it is assumed to occur at t = 0.

The final time, tfinal, is an optional argument. If omitted, it defaults to the maximum time in u. The time response is calculated as though the input is a constant value between the points specified in u. If tfinal is greater than the largest independent variable in u, the input is held at the last value in u.

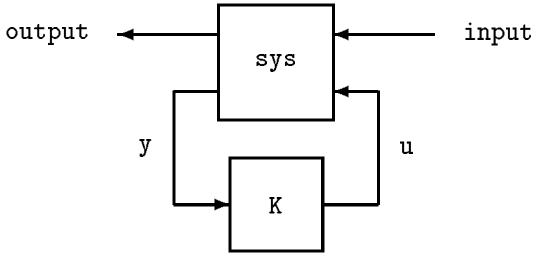
For continuous-time evaluation (trsp), you can optionally specify an integration time with the variable int. If this is omitted, or is equal to zero, an appropriate value is calculated and displayed. The calculated integration time depends on the minimum spacing in the input and the fastest dynamics in sys. int will also be the independent variable step size in the regularly spaced output, y. If a coarser output is adequate, it can be obtained with the function vdcmate.

Initial conditions can optionally be specified with the argument, x0. This specifies the state vector at the first time point in the input vector. If x0 is omitted, or is a zero scalar, then it is assumed to be a zero vector.

trsp interpolates the input with a zero-order hold of step size equal to int, discretizes the output at this same step size, and calculates the response from the initial time to tfinal in steps of int.

dtrsp calculates the response for a discrete-time system, dsys. The time (for the independent variable) between discrete indices is T. If the input is not regularly spaced at intervals of time T, it is interpolated. tfinal and x0 behave in the same manner as for trsp.

sdtrsp calculates a sampled-data time response for a closed-loop system with a continuous generalized plant (sys) and a discrete controller (K). The interconnection is illustrated below.

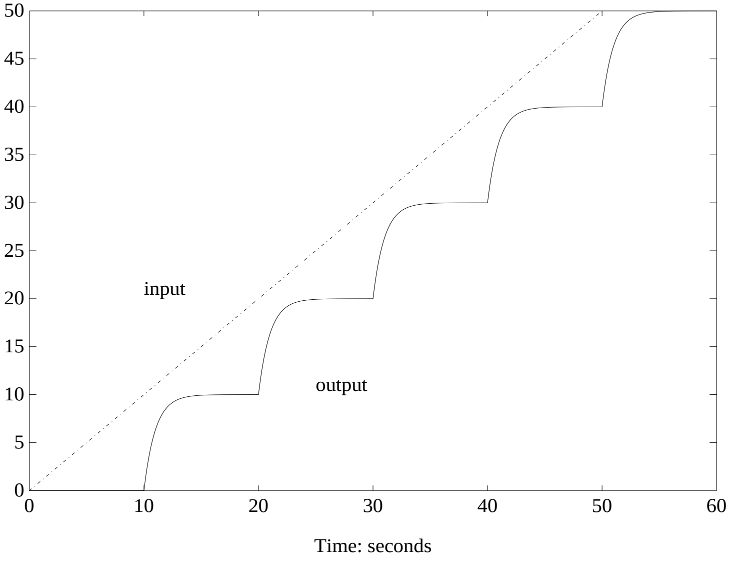


The signals, output, y, and u are calculated. T is the sampled-data controller sample time, and you have the same input and tfinal options as the trsp function. Similarly, an integration step size, int, can optionally be specified for the continous part of the simulation. Initial conditions can be specified for sys (x0) and K (z0).

Examples

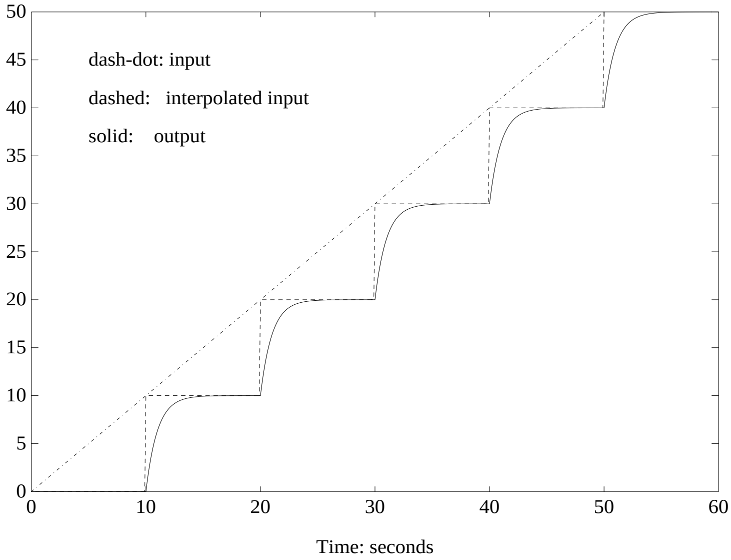
A simple SISO system illustrates the use of trsp. This example shows the consequences of the input being assumed to be constant between time points.

```
sys = pck(-1,1,1);
minfo(sys)
system:1 states1 outputs1 inputs
u = vpck([0:10:50]',[0:10:50]');
y = trsp(sys,u,60);
integration step size: 0.1
interpolating input vector (zero order hold)
minfo(y)
varying:601 pts1 rows1 cols
vplot(u,'-.',y,'-')
xlabel('time: seconds')
text(10,20,'input')
text(25,10,'output')
```



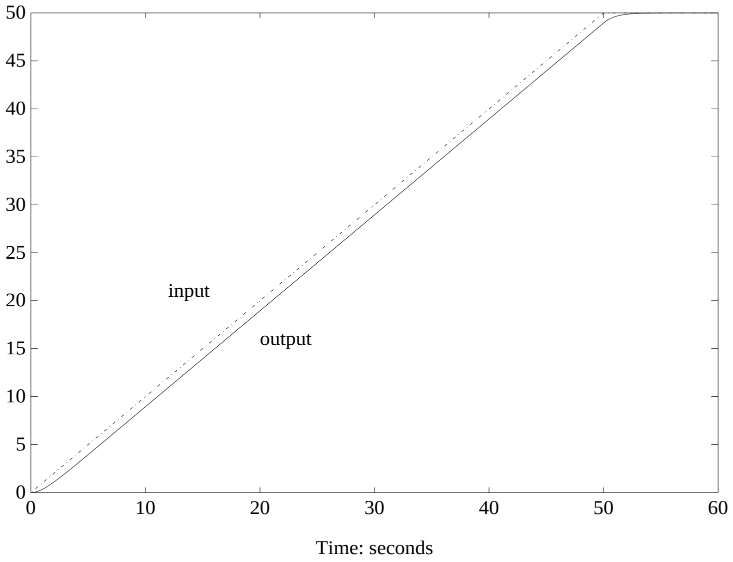
At first glance the output does not seem to be consistent with the plotted input. Remember that `trsp` assumes that the input is held constant between specified values. The `vplot` and `plot` commands display a linear interpolation between points. This can be more clearly seen by displaying the input signal interpolated to at least as small a step size as the default integration step (here 0.1 seconds).

```
vplot(u,'-.',vinterp(u,0.1),'--',y,'-')
xlabel('time: seconds')
text(5,44,'dash-dot: input')
text(5,40,'dashed: interpolated input')
text(5,36,'solid: output')
```



The staircase nature of the input is now evident. If you really want to have a ramp input, the function `vinterp` also provides linear interpolation. A linearly interpolated input is used in the following example.

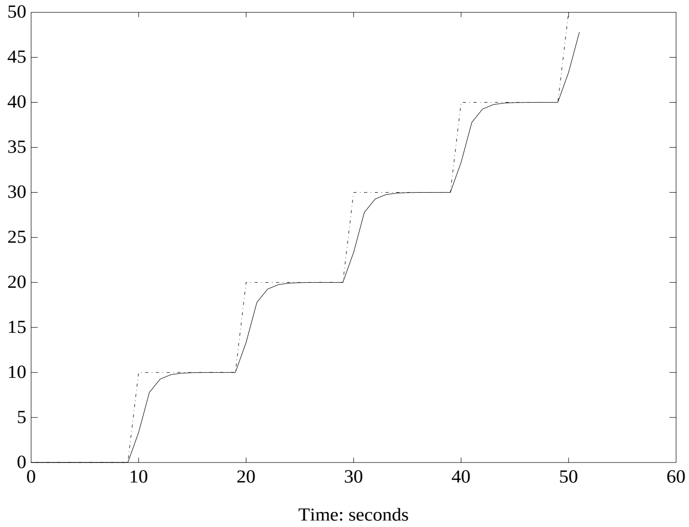
```
uramp = vinterp(u,0.1,60,1);
minfo(uramp)
varying:601 pts1 rows1 cols
yramp = trsp(sys,uramp);
integration step size: 0.1
vplot(uramp,'-.',yramp,'-')
xlabel('time: seconds')
text(20,15,'output')
text(12,20,'input')
```



Note that because the input is regularly spaced, with spacing less than or equal to the default integration time, the input is not interpolated by `trsp`. Since no final time was specified in the `trsp` argument list, and 60 seconds was specified to `vinterp` as the final time, this becomes the last time in the input vector uramp.

To illustrate the use of `dtrsp`, a bilinear transformation generates a digital system. The sample time is chosen as 1 second. The output is plotted against a 1 second interpolation of the input.

```
T = 1;
dsys = tustin(sys,T);
ydig = dtrsp(dsys,u,T);
vplot(ydig,'-',vinterp(u,1),'-.')
xlabel('time: seconds')
```



To illustrate the use of sdtrsp, consider the application of a discrete controller to a double integrator. A continuous plant and a discrete controller are created. A sample and hold equivalent of the plant is formed and the discrete closed-loop system is calculated. Simulating this with dtrsp gives the system response at the sample points. sdtrsp is then used to calculate the intersample behavior.

```
P = nd2sys(1,[1,0,0]);
T = 1.0/20;
C=pck([-1.5 T/4; -2/T -.5],[ .5 2;1/T 1/T], ...
   [-1/T^2 -1.5/T], [1/T^2 0]);
```

The closed-loop digital system is now set up.

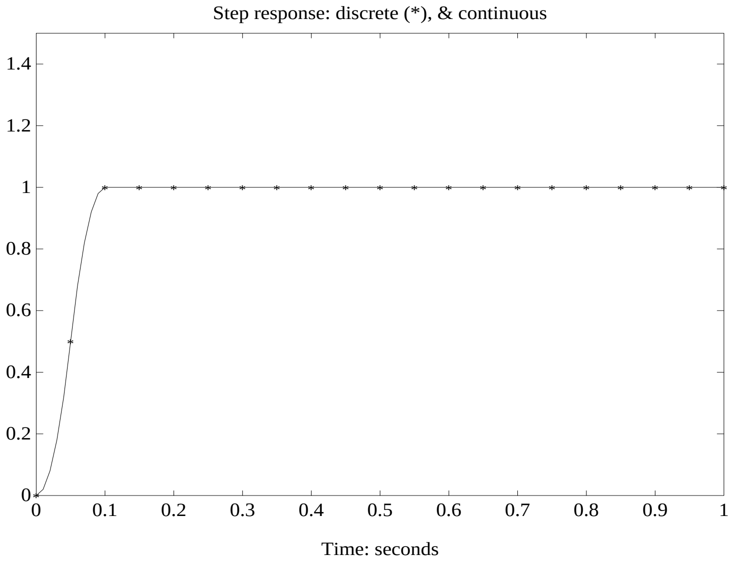
```
Pd = samhld(P,T);
systemnames = 'Pd C';
inputvar = '[ref]';
outputvar = '[Pd]';
input_to_Pd = '[C]';
input_to_C = '[ref ; Pd]';
sysoutname = 'dclp';
cleanupsysic = 'yes';
sysic;
```

dtrsp is used to simulate the digital step response.

```
ustep = step_tr(0,1,T,20*T);
y = dtrsp(dclp,ustep,T);
```

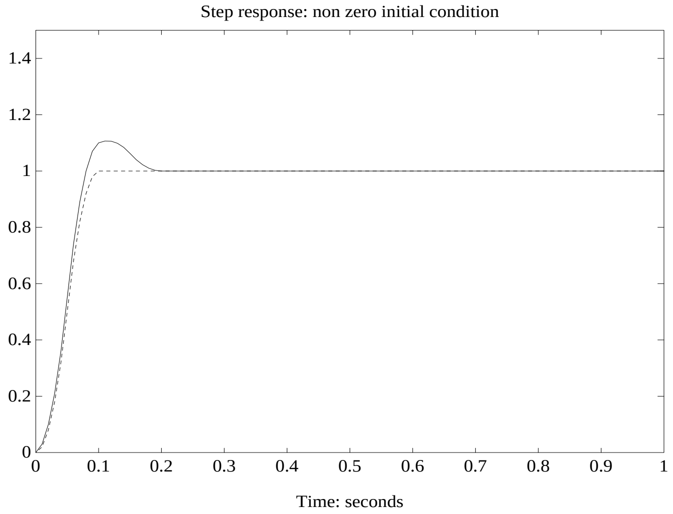
The continuous interconnection is set up and the sampled data response is calculated with sdtrsp.

```
M = mmult([0,1;1,0;0,1],daug(1,P));
y1 = sdtrsp(M,C,ustep,T);
vplot(y,'*',y1,'-')
axis([0,1,0,1.5])
xlabel('Time: seconds')
title('Step response: discrete (*), &continuous')
```



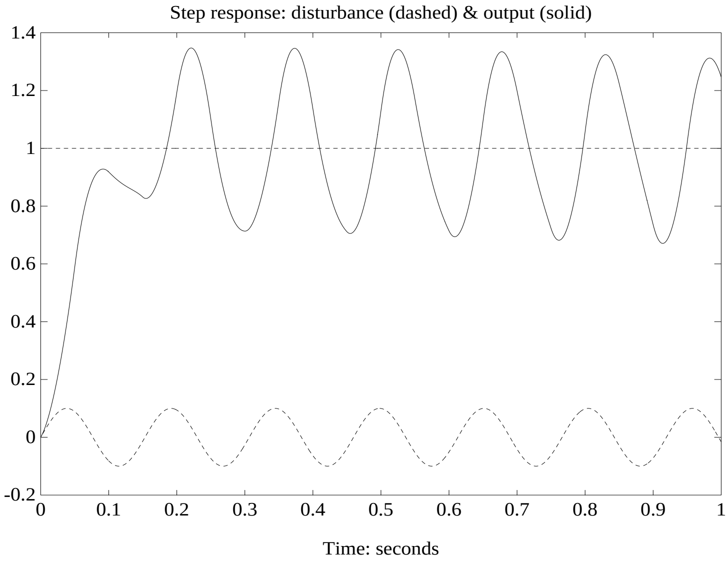
Now we look at the effect of a nonzero initial condition in the continuous system. Note how examining the system at only the sample points will underestimate the amplitude of the overshoot.

```
y2 = sdtrsp(M,C,vpck(1,0),T,1,0,[1;0]);
vplot(y1,'--',y2,'-')
axis([0,1,0,1.5])
xlabel('Time: seconds')
title('Step response: non zero initial condition')
```



Finally, we will examine the effect of a sinusoidal disturbance at the continuous plant output. This controller has not been designed to reject such a disturbance and the system does not contain anti-aliasing filters. Simulating the effect of anti-aliasing filters is easily accomplished by including them in the continuous interconnection structure.

```
M2 = mmult([0,1,1;1,0,0;0,1,1],daug(1,1,P));
dist = sin_tr(41,0.1,0.001,1);
[dat,datptr,t] = vunpck(dist);
input = abv(vpck(ones(t),t),dist);
[y3,meas,act] = sdtrsp(M2,C,input,T);
vplot(y3,'-',input,'--')
xlabel('Time: seconds')
title('Step response: disturbance (dashed) & output (solid)')
```



Algorithm

`trsp` first calculates an integration time (or uses the specified integration time) to determine the sample time at which to discretize the continuous-time system. The integration time is taken to be the inverse of 10 times the fastest mode of the input system. The input vector is interpolated at each sample time via a zero-order hold, and then a sample-hold of the input continous system is performed. Finally the time response of the system is performed via a `for` loop at each integration time step. `dtrsp` is provided a discrete time system and a sample time. `dtrsp` first interpolates the input vector via a zero-order hold and then determines the time response via a `for` loop at each sample time.

Caution Systems with fast dynamics lead to very small integration times. This is both time consuming and requires a significant amount of storage. We recommend you residualize the fastest modes of the system, which does not affect the time response. This can be done with the μ-Tools command `sresid`.

See Also

`cos_tr`, `siggen`, `sin_tr`, `step_tr`, `sysbal`, `vdcmate`, `vinterp`

Purpose

Create a discrete-time version of a continuous-time SYSTEM matrix using a bilinear or prewarped `tustin` transformation

Syntax

```
dsys = tustin(csys,T,prewarpf)
```

Description

The packed continuous SYSTEM matrix, `csys`, is converted into a discrete-time SYSTEM matrix, `dsys`, using a bilinear transformation with prewarping. The argument `T` is the sample time, in seconds. `prewarpf` is the prewarp frequency in rads/sec. `prewarpf` is an optional argument, and if omitted, or equal to zero, a bilinear transformation is performed instead.

The resulting discrete system, `dsys`, has the same transfer function at the continuous system, `csys`, at the prewarp frequency. Choosing a prewarp frequency close to the crossover frequency is often appropriate for a control system. Choosing a prewarp frequency too close to the Nyquist frequency $(1/2T)$ can result in severe distortion at the lower frequencies. In the extreme, if prewarp is greater than or equal to π/T, the discrete system can be unstable.

Note that the transfer function is preserved at zero frequency with a bilinear transformation, hence having the input variable `prewarpf` equal to zero to indicate a bilinear transformation is therefore consistent.

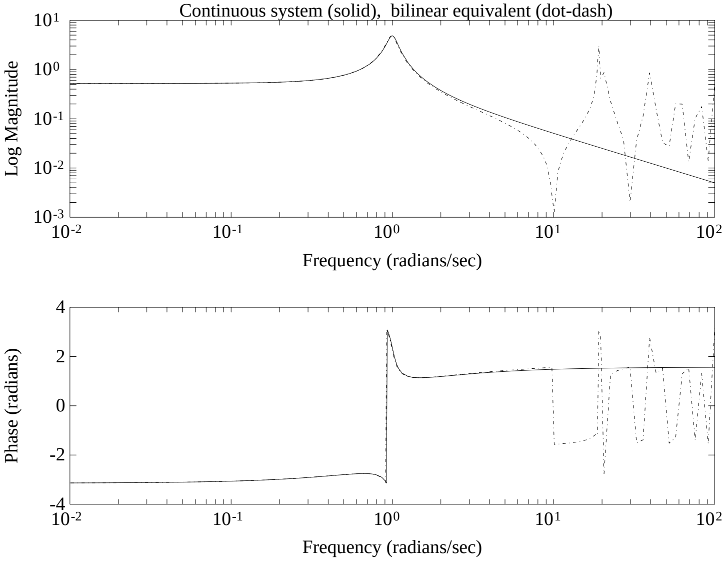
Examples

Create a second-order system with a resonance at 1 rad/sec.

```
a = [-.1,1;-1,-0.05];
b = [1;1]; c = [-0.5,0]
sys = pck(a,b,c);
minfo(sys)
system:2states1 outpus1 inputs
omega = logspace(-2,2,100);
omega2 =[ [0.05:0.1:1.5] [1.6:.5:20] [0.9:0.01:1.1] ];
omega = sort([omega omega2]);
sys_g = frsp(sys,omega);
```

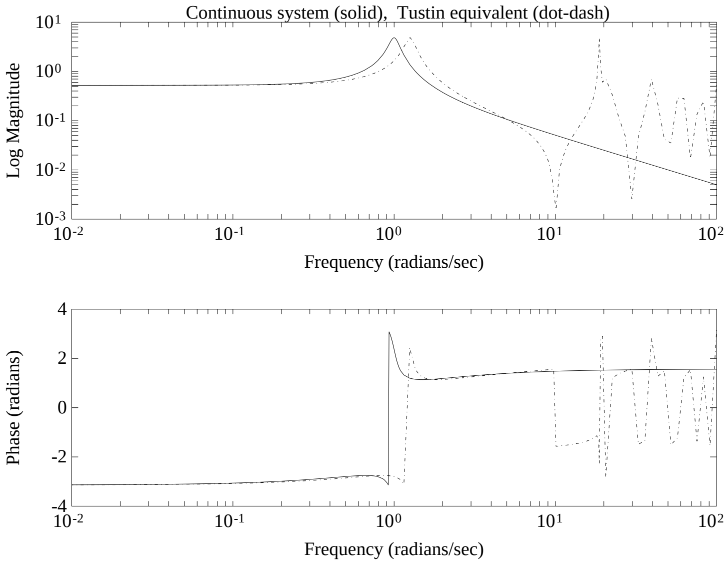
Choose a sample frequency of 20 rads/sec and discretize the system with a bilinear transformation.

```
T = 2*pi/20;
dsys = tustin(sys,T);
dsys_g = frsp(dsys,omega,T);
vplot('bode',sys_g,dsys_g,'-.');
title('Continuous system (solid), bilinear equivalent
(dot-dash)')
```



The bilinear approximation is accurate up to about 2 rads/sec. This example shows the effect of choosing a higher prewarping frequency, specifically 5 rads/sec.

```
prewarpf = 5;
dsys2 = tustin(sys,T,prewarpf);
dsys2g = frsp(dsys2,omega,T);
vplot('bode',sys_g,dsys2_g,'-.');
title('Continuous system (solid), tustin equivalent (dot-dash)')
```



Note the distortion in the frequency of the lightly damped peak. At 5 rads/sec both the continuous and discrete systems have the same transfer function.

```
sys_5 = vunpck(frsp(sys,5));
dsys2_5 = vunpck(frsp(dsys2,5,T));
err = abs(dsys2_5 - sys_5);
fprintf('error at %g rad/sec is : %g ',prewarpf,err);
error at 5 rad/sec is : 1.155158e-17
```

This example highlights the distortion possible. If the frequency of the resonance had been critical to the design, a prewarp frequency of 1 rad/sec would have been more appropriate.

As an alternative, you can generate a filter/controller design using a warped frequency scale in the continuous domain. Then the transformation to the discrete domain would result in the correct transfer function at the frequencies of interest.

Algorithm

The prewarped `tustin` transformation is based on the equation:

$$\texttt{csys} = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right], \quad \texttt{using the prewarped Tustin transform}$$

$$\texttt{dsys} = \left[\begin{array}{c|c} A_{disc} & B_{disc} \\ \hline C_{disc} & D_{disc} \end{array}\right],$$

where

$$\alpha = \frac{\texttt{prewrapf}}{\texttt{tan(T}_{*}\texttt{prewarpf/2)}}$$

$$A_{disc} = (I + \tfrac{1}{\alpha}\mathbf{A}) * (I - \tfrac{1}{\alpha}\mathbf{A})^{-1}$$

$$B_{disc} = (\tfrac{2}{\alpha})^{\frac{1}{2}} * (I - \tfrac{1}{\alpha}\mathbf{A})^{-1} * \mathbf{B}$$

$$C_{disc} = (\tfrac{2}{\alpha})^{\frac{1}{2}} * \mathbf{C} * (I - \tfrac{1}{\alpha}\mathbf{A})^{-1}$$

$$D_{disc} = D + \tfrac{1}{\alpha} * \mathbf{C} * (\ I - \tfrac{1}{\alpha}\mathbf{A}^{-1}) * \mathbf{B}$$

Reference

Oppenheim, A.V., and R.W. Schafer, *Digital Signal Processing*, Prentice Hall, New Jersey, 1975.

See Also

`dtrsp`, `frsp`, `samhld`, `tustin`

unum, xnum, ynum

Purpose

unum returns the SYSTEM matrix input dimension

xnum returns the SYSTEM matrix state dimension

ynum returns the SYSTEM matrix output dimension

Syntax

```
numinputs = unum(sys)
numstates = xnum(sys)
numoutputs = ynum(mat)
```

Description

unum returns the input (column) dimension of SYSTEM, CONSTANT, and VARYING matrices.

xnum returns the state dimension of SYSTEM matrices.

ynum returns the output (row) dimension of SYSTEM, CONSTANT, and VARYING matrices.

Examples

```
sys = sysrand(3,4,5);
xnum(sys)
ans =
   3
unum(sys)
ans =
   5
ynum(sys)
ans =
   4
mat = crand(17,9);
xnum(mat)
ans =
   0
unum(mat)
ans =
   9
ynum(mat)
ans =
   17
```

See Also

find, minfo, xtract, xtracti

vabs, vimag, vreal, vfloor, vceil

Purpose

Perform element-by-element operations on CONSTANT and VARYING matrices

Syntax

```
matout = vabs(matin)
matout = vimag(matin)
matout = vreal(matin)
matout = vfloor(matin)
matout = vceil(matin)
```

Description

This set of commands allows standard element-by-element operations on CONSTANT/VARYING matrices. These commands are identical to abs, imag, real, floor, and ceil but also work on VARYING matrices.

vabs	element-by-element absolute value of a VARYING matrix
vimag	element-by-element imaginary part of a VARYING matrix
vreal	element-by-element real part of a VARYING matrix
vfloor	element-by-element floor of a VARYING matrix
vceil	element-by-element ceiling of a VARYING matrix

A general element-by-element command, vebe, allows all standard arithmetic MATLAB element commands that have only one input argument.

Examples

Construct a complex VARYING matrix and find the magnitude of the entries and the real parts.

```
see(matin)
1 row                       2 columns
iv = .2
   0.2190 - 0.4379i0.6789 - 1.3577i
iv = .7
   0.0470 - 0.0941i0.6793 - 1.3586i
```

```
see(vabs(matin))
1 row      2 columns
iv = .2
0iv = .7
   0.10521.5190
see(vreal(matin))
1 row2 columns
iv = .2
   0.21900.6789
iv = .7
   0.04700.6793
```

Use `pkvnorm` to find the maximum element magnitude of the VARYING matrix.

```
pkvnorm(matin,inf)
1 row      2 columns
ans =
   1.5190
```

This agrees with the maximum magnitude of `matin(1,1)` associated with the second independent variable.

See Also

`vdet`, `vdiag`, `vebe`, `veval`

vdet, vdiag, vexpm, vrcond

Purpose

Calculate determinant, diagonal, matrix exponential and estimate of condition number of a CONSTANT or VARYING matrix

Syntax

```
out = vdet(matin)
out = vdiag(matin)
out = vexpm(matin)
out = vrcond(matin)
```

Description

These commands operate on square, CONSTANT and VARYING matrices and they are identical to the MATLAB commands `det`, `diag`, `exp`, and `rcond` on CONSTANT matrices.

`vdet` of a square, VARYING matrix, returns `matout`, which is a VARYING 1 × 1 matrix, containing the value of the determinant of `matin` at each independent variable value.

`vdiag` of a square, VARYING matrix, returns `matout`, which is a VARYING matrix of size `min(size(matin))` × 1, containing the diagonal elements of `matin` at each independent variable.

`vexpm` of a square, VARYING matrix, returns `matout`, which is a VARYING matrix of the same size as `matin`, containing the matrix exponential of `matin`. The MATLAB command `expm`, which is called, uses a Pade expansion after scaling `matin` to calculate the exponential.

`vrcond` of a square, VARYING matrix, returns `matout`, which is an estimate of the condition number of a matrix.

Examples

`vdet` and `vrcond` work similarly to their MATLAB counterparts, `det` and `rcond`, but on square VARYING matrices as shown below.

```
see(matin)
2 rows                    2 columns
iv = 2.3
   0.04750.3282
   0.73610.6326
iv = 5.6
   0.75640.3653
   0.99100.2470
matout = vdet(matin);
```

```
see(matout)
1 row                      1 column
iv = 2.3
   -0.2116
iv = 5.6
   -0.1752
see(vrcond(matin))
1 row                      1 column
iv = 2.3
   0.1907
iv = 5.6
   0.0848
```

See Also

`det`, `diag`, `expm`, `rcond`

Purpose

Perform element-by-element operations on CONSTANT and VARYING matrices

Syntax

```
out = vebe('oper',matin)
```

Description

The vebe function allows any single argument MATLAB element-by-element arithmetic command to operate on a VARYING matrix. The first input argument, oper, is the character string defining the MATLAB element-by-element command and matin is the VARYING matrix on which the command is applied. vebe calls the MATLAB eval command to execute the string command. Some standard MATLAB comands compatible with vebe are sin, abs, real, imag, and gamma.

Examples

In this example of vebe, the real part of a matrix is found along with gamma of each matrix element.

```
see(matin)
3 rows 3 columns
iv = 4.2

1.0000 + 2.0000i   1.0000 + 2.0000i   1.0000 + 2.0000i
2.0000 + 4.0000i   2.0000 + 4.0000i   2.0000 + 4.0000i
3.0000 + 6.0000i   3.0000 + 6.0000i   3.0000 + 6.0000i

iv = 11.01

4.0000 +  8.0000i   4.0000 +  8.0000i   4.0000 +  8.0000i
5.0000 + 10.0000i   5.0000 + 10.0000i   5.0000 + 10.0000i
6.0000 + 12.0000i   6.0000 + 12.0000i   6.0000 + 12.0000i

matout = vebe('real',matin);
see(matout)
3 rows 3 columns
```

```
iv = 4.2
1  1  1
2  2  2
3  3  3

iv = 11.01
4  4  4
5  5  5
6  6  6
see (vebe ('gamma ', matin))
3 rows 3 columns

iv = 4.2
1  1  1
1  1  1
2  2  2

iv = 11.01
  6    6    6
 24   24   24
120  120  120
```

See Also eval, vabs, vceil, veval, vfloor, vimag, vreal

Purpose

Calculate eigenvalues and eigenvectors of CONSTANT and VARYING matrices

Syntax

```
evals = veig(matin)
[evecs,evals] = veig(matin)
[evecs,evals] = veig(matin,'nonbalance')
[evecs,evals] = veig(mat1,mat2)
```

Description

veig is identical to MATLAB's command eig, but works on VARYING matrices. veig solves for the eigenvalues (evals) and optionally the eigenvectors (evecs) of the input matrix matin. veig works only on square CONSTANT or VARYING matrices. Depending on the input arguments, the following operations are performed:

evals = veig(matin) returns a VARYING vector evals containing the eigenvalues of the VARYING matrix matin for each independent variable.

[evecs,evals] = veig(matin) returns the VARYING diagonal matrix evals and a square VARYING matrix evecs whose columns are the corresponding eigenvectors for each independent variable.

[evecs,evals] = veig(matin,'nobalance') is the same as the above command without performing a preliminary balancing step. Balancing a matrix, which has very small entries due to round off error, can lead to incorrect eigenvectors.

[evecs,evals] = veig(mat1,mat2) returns a VARYING diagonal matrix of generalized eigenvalues evals and the corresponding values of generalized right eigenvectors evecs associated with each independent variable. mat1 and mat2 can be CONSTANT or VARYING matrices. If either is CONSTANT, then that same matrix is used in the generalized eigenvalue solution for each independent variable. If they are both VARYING matrices, then they must have the same independent variables.

Examples

Create a 2 × 2 random VARYING matrix and find its eigenvalues.

```
see(matin)
2 rows  2 columns

iv = 0.1
0.9304   0.5269
0.8462   0.0920
iv = 0.4
0.6539   0.7012
0.4160   0.9103
evals = veig(matin);
see(evals)
2 rows1 column

iv = 0.1
  1.2996
 -0.2772

iv = 0.4
 0.2270
 1.3372
```

Using the same matrix and creating another two by two VARYING matrix, solve the generalized eigenvalue problem with these two matrices.

```
mat1 = matin;
mat2 = vpck([4*eye(2);3*eye(2)],[.1 .4]);
[evecs,evals] = veig(mat1,mat2);
see(evecs)
2 rows  2 columns

iv = 0.1
 0.8190   -0.3999
 0.5738    0.9166

iv = 0.4
 0.8542   0.7162
-0.5200   0.6979
see(evals)
```

```
2 rows   2 columns

iv = 0.1
 0.3249   0
 0       -0.0693
iv = 0.4
 0.0757   0
 0        0.4457
```

Algorithm

veig calls the MATLAB eig command.

See Also

eig, indvcmp, svd, vsvd, vpoly, vroots

veval

Purpose

Evaluate general functions of CONSTANT, SYSTEM, and VARYING matrices

Syntax

```
[out1,out2,out3,...]= veval('oper',in1,in2,in3,...)
```

Description

The `veval` function evaluates the command `oper` on the input matrices. `veval` works like feval but on collections of VARYING, CONSTANT, and SYSTEM matrices. `'oper'` is a character string with the name of a MATLAB function (user written, or MATLAB supplied). The function is applied to each input argument at the independent variable's values. Any CONSTANT or SYSTEM matrix is held at its value while the sweep through the independent variable is performed. `veval` is currently limited to 10 output arguments, and 13 input arguments. These are both easily changeable. `veval` can be used to generate and manipulate VARYING, SYSTEM matrices or VARYING matrices whose elements are themselves VARYING matrices. Arbitrary nesting of VARYING matrices is possible.

The `veval` function is very useful for rapid prototyping of customized commands employing VARYING matrices.

Examples

To show the flexibility of `veval`, two random SYSTEM matrices are constructed. The poles of each SYSTEM are determined with the spoles command.

```
sys1 = sysrand(2,1,1);
sys2 = sysrand(2,1,1);
spoles(sysl)
ans =
 0.1577
 0.7405
spoles(sys2)
ans =
 0.6273
-0.5661
```

These two SYSTEM matrices are combined to form a VARYING matrix, `vsys`. The `veval` command is used to find poles of the VARYING matrix, which consists of the two SYSTEM matrices. A SYSTEM matrix is associated with each independent variable.

```
vsys = vpck([sysl;sys2],[1 2]);
vsyspoles = veval('spoles',vsys);
see(vsyspoles)
2 rows1 column

iv = 1
 0.1577
 0.7405

iv = 2
 0.6273
-0.5661
```

See Also

`eval`, `feval`, `vebe`

vfft, vifft, vspect

Purpose

Calculate FFTs, inverse FFTs, and perform spectral analysis on VARYING matrices

Syntax

```
yfreq = vfft(ytime,n)
ytime = vifft(yfreq)
P = vspect(x,m,noverlap,'window')
P = vspect(x,y,m,noverlap,'window')
```

Description

vfft implements the MATLAB fft command on VARYING matrix structures. A one-dimensional FFT of length n is performed on each element of the VARYING matrix ytime. It is assumed that the independent variable is in units of seconds. The independent variables are regularly spaced — only the first interval is used to determine the frequency scale. yfreq is returned with the independent variable, frequency, in radians/second.

vifft performs the inverse FFT. This is done with the MATLAB command ifft(yfreq) for each element of the VARYING matrix.

vspect is the VARYING matrix structure equivalent of the Signal Processing Toolbox command, spectrum. For algorithmic details, see the spectrum command. Note that vspect gives you the option of specifying a window for the data. For example, using the string hamming as the fifth argument generates a window with the command window = hamming(n);. hamming is an M-file in the Signal Processing Toolbox. You can use custom windows by specifying the name as the window argument.

In the case of the spectrum of a single signal, the command

```
P = vspect(x,m,noverlap,'window');
```

will return a VARYING matrix, P, containing the power spectral density of x. Note that x, and therefore P, can be a matrix of signals. In the case of the spectrum, and cross-spectrum, of two signals, the command

P = vspect(x,y,m,noverlap,'window');

will return a VARYING matrix, P, with the following five columns.

```
[Pxx Pyy Pxy Txy Cxy]
```

These are given below.

Pxx	Power spectral density of x
Pyy	Power spectral density of y
Pxy	Cross spectral density
Txy	Complex transfer function between x and y
Cxy	Coherence function between x and y

The signal x, must be scalar (i.e., a one row, one column, VARYING matrix). y can be a vector signal. The row dimension of p is the same as that of y. vspect can do single-input, multiple-output (SIMO) identification. This is illustrated in the following example. Refer also to the example in the *Tutorial* chapter.

vfft, vifft, and vspect have not been optimized for speed. The appropriate row and column data is extracted from the VARYING matrices with the μ-Tools commands, sel and xtract. sbs and abv are used to create the final output.

Examples

A single-input two-output system is generated as an identification example. This example is only a simple illustration of some of the frequency domain techniques available.

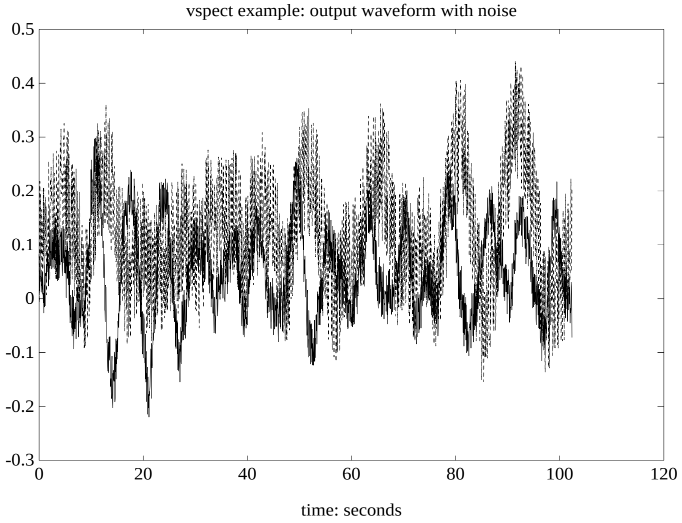
```
a1 = [-.1,1;-1,-0.05];
b1 = [1;1]; c1 = [-0.5,0];
sys1 = pck(a1,b1,c1);
a2 = [-.1,0.5;-0.5,-0.1];
b2 = [1;1]; c2 = [-0.5,0];
sys2 = pck(a2,b2,c2);
sys = abv(sys1,sys2);
minfo(sys)
system:4 states2 outputs1 inputs
```

Now a random input signal is created.

```
t = [0:0.05:102.35];
u = siggen('0.5-rand(size(t))',t);
minfo(u)
varying:2048 pts1 rows1 cols
```

The signal u is the input to the system. siggen is used to generate some random noise on the output signal, y.

```
y =
madd(trsp(sys,u),siggen('[0.01*rand(size(t));0.025*rand(size(t))
]',t));
integration step size: 0.05
vplot(y)
title('vspect example: output waveform with noise')
xlabel('time: seconds')
```



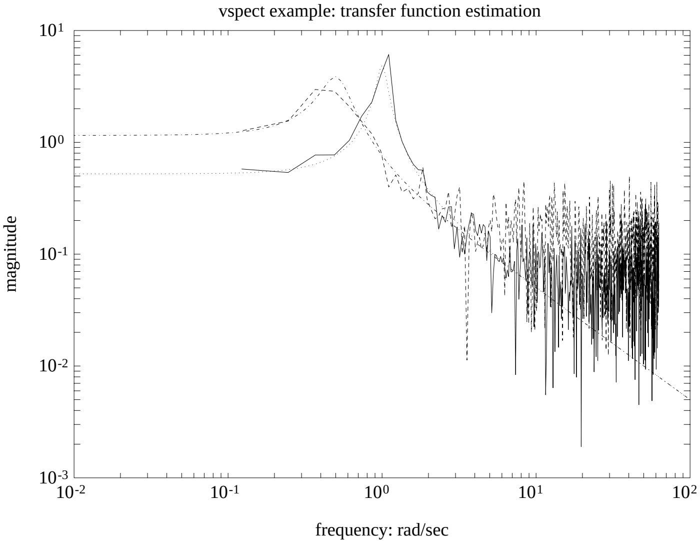
The vspect command specifies a 1024 point window, with 512 points of overlap. A Hamming window is applied to the data.

```
P = vspect(u,y,1024,512,'hamming');
3 hamming windows in averaging calculation
```

Column 4 in P contains the complex transfer function estimate. Its magnitude is compared to the actual system transfer function.

```
omega = logspace(-2,2,100);
```

```
omega2 =[ [0.05:0.1:1.5] [1.6:.5:20] [0.9:0.01:1.1] ];
omega = sort([omega omega2]);
sys_g = frsp(sys,omega);
vplot('liv,lm',sel(P,1:2,4),sys_g);
Warning: Data includes a number that is negative or zero.
The LOG of this results in NaN or Infinity and is not shown on
plot.
title('vspect example: transfer function estimation ')
ylabel('magnitude')
xlabel('frequency: rad/sec')
```



Algorithm

`vfft`, `vifft`, and `vspectrum` call the MATLAB commands `fft` and `ifft`.

Reference

Ljung, L., *System Identification: Theory for the User*, Prentice Hall, New Jersey, 1987.

Oppenheim, A.V., and R.W. Schafer, *Digital Signal Processing*, Prentice Hall, New Jersey, 1975.

See Also

`fft`, `ifft`, `spectrum`

vfind

Purpose

Unary find function across independent variable

Syntax

```
[iv_value,iv_index] = vfind(condition,mat)
```

Description

vfind is a unary find function that searches across independent variable values. The condition to be tested can be any valid MATLAB conditional statement, using the string mat to identify the matrix, and iv as the independent variable's value. Both the values and indices of the applicable independent variables are returned.

Examples

Suppose that matin is a VARYING matrix. In order to find those entries for which the product of the norm of the matrix, and the independent variable is greater than 2, use vfind as follows.

```
[iv_value,iv_index] = vfind('iv*norm(mat)>2',matin);
matpropv = xtract(matin,iv_value); % extract by value
matpropi = xtracti(matin,iv_index); % extract by index
pkvnorm(msub(matpropv,matpropi)) % compare - both are the same
```

See Also

find, xtract, xtracti

Purpose

Interpolate or decimate VARYING matrices

Syntax

```
vout = vinterp(vin,stepsize,finaliv,order)
vout = vinterp(vin,varymat,order)
vout = vdcmate(vin,spacing)
```

Description

In the first form, `vinterp` produces a regularly spaced interpolated version of the input VARYING matrix. The input arguments are

`stepsize`	independent variable stepsize
`finaliv`	end value for independent variable (Optional: the default is the final independent variable in the input)
`order`	type of interpolation (optional, default = 0) 0 zero-order hold 1 linear interpolation

The end value for the independent variable may or may not be in the actual output. This is consistent with the usual MATLAB treatment of regularly spaced vectors. For example, consider

```
iv = [1:2:6];
disp(iv)
   1   3   5
```

Note that the value of 6 does not appear in the vector.

In the second form, `vinterp` produces a VARYING matrix `vout` that is an interpolated version of `vin`. The independent variables of `vout` are the same as the independent variables of `varymat`. The input arguments are

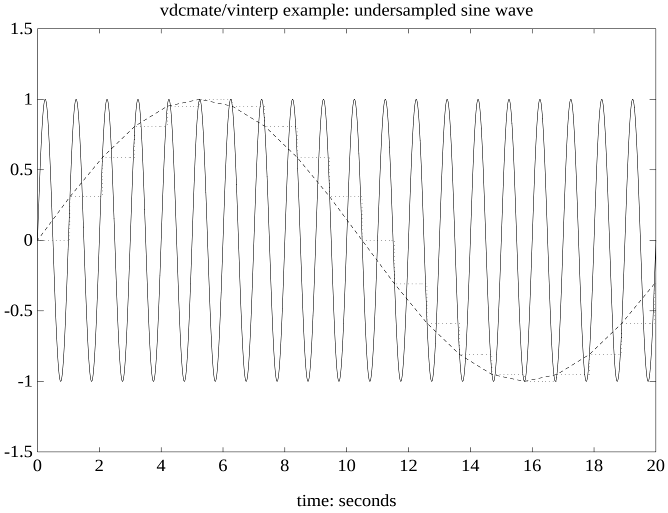
`varymat`	VARYING matrix with desired independent variables
`order`	type of interpolation (optional, default = 0) 0 zero-order hold 1 linear interpolation

`vdcmate` decimates the VARYING matrix `vin`, whose independent variable must be linearly spaced and in ascending order. If `spacing` has a value of *n*, then the output contains only the matrices corresponding to every *n*th independent variable of the input. If no spacing is specified, the default is 10.

Examples

siggen creates a sinewave. This is effectively sampled by vdcmate and then interpolated by vinterp. Note that the default interpolation is a zero-order hold, giving a stair-step output, yi. If a linearly interpolated output were specified, it would look identical to yd since the MATLAB plot command displays a linear interpolation.

```
timebase = [0:0.005:20];
y = siggen('sin(2*pi*t)',timebase);
minfo(y)
varying:4001 pts1 rows1 cols
yd = vdcmate(y,210);
minfo(yd)
varying:20 pts1 rows1 cols
yi = vinterp(yd,0.005,20,0);
minfo(yi)
varying:4001 pts1 rows1 cols
axis([0,20,-1.5,1.5])
vplot(y,yd,yi)
title('vdcmate/vinterp example: undersampled sine wave')
xlabel('time: seconds')
```



See Also

dtrsp, sort, sortiv, tackon, trsp

vldiv, vpinv, vrdiv

Purpose

Determine left division, pseudo-inverse and right division of a CONSTANT or VARYING matrix

Syntax

```
out = vldiv(mat1,mat2)
out = vpinv(mat,tol)
out = vrdiv(mat1,mat2)
```

Description

These commands operate on VARYING matrices and they are identical to the MATLAB commands \, pinv, and / on CONSTANT matrices.

`vldiv` of a VARYING matrix returns `out`, which is a VARYING matrix, containing the value of the left division `(mat1(i)mat2(i))` at each independent variable value. `vldiv` is identical to the MATLAB command \ for CONSTANT matrices.

`vpinv` of a VARYING matrix returns `out`, which is the pseudo-inverse of `mat` at each independent variable. `out` is of the same dimension as `vcjt(out)`, and satisfies `mat = mmult(mat,out,mat)`. `tol` is used within the `svd` routine to determine zero singular values. The default value of `tol` is 1e–12. `vpinv` is identical to the MATLAB command `pinv` for CONSTANT matrices.

`vrdiv` of a VARYING matrix returns `out`, which is a VARYING matrix, containing the value of the right division `(mat1(i)/mat2(i))` at each independent variable value. `vrdiv` is identical to the MATLAB command / for CONSTANT matrices.

See Also

`\`, `/`, `pinv`, `vinv`, `vsvd`

vpck, vunpck, var2con

Purpose

Pack and unpack a VARYING matrix and convert from a VARYING matrix to a CONSTANT

Syntax

```
matout = vpck(matin,indv)
[varydata,rowpoint,indv,err] = vunpck(mat)
[matout,ivval] = var2con(mat,desiv)
```

Description

The data structure for a VARYING matrix consists of the sampled matrix values stacked one upon each other, and the particular independent variable values. `vpck` places the stacked data from the input variable, `matin`, and the vector, `indv`, which represents the independent variable values, into a new matrix, `matout`, with the correct structure and data structure of a VARYING matrix.

The command `vunpck` performs the inverse operation; unpacking a VARYING matrix into stacked data `varydata`, row pointers `rowpoint`, a vector of independent variables `indv`, and an error flag `err`. The value of `rowpoint(i)` points to the row of data that corresponds to the first row of the *i*th value of `matin`. `indv` is a column vector with the independent variable values. The error flag is normally 0 but it is set to 1 if the input matrix is a SYSTEM.

`var2con` converts VARYING matrices to CONSTANT matrices. If there is one input argument, `mat`, and it is a VARYING matrix, then the output `matout` is the CONSTANT matrix in `mat` associated with the independent variable's first value. The optional second output argument is this independent variable's value. If two input arguments are used, then the first is a VARYING matrix, and the second is a desired independent variable's value. The command finds the matrix in `mat` whose independent variable's value is closest to `desiv`, and returns this matrix as a CONSTANT matrix.

Examples

Construct a VARYING matrix from a CONSTANT matrix and a vector of independent variables.

```
disp(matin)
   1   1   1
   2   2   2
   3   3   3
   4   4   4
   5   5   5
   6   6   6
time = [.1 2.3 5.6]';
disp(time)
   1.0000e-01
   2.3000e+00
   5.6000e+00
matout = vpck(matin,time);
see(matout)
2 rows                 3 columns

iv = 0.1
   1   1   1
   2   2   2

iv = 2.3
   3   3   3
   4   4   4

iv = 5.6
   5   5   5
   6   6   6
```

See Also

pck, unpck, xtract, xtracti

vplot

Purpose

Plot multiple VARYING matrices on the same graph

Syntax

```
vplot('plot_type',vmat1,vmat2,...)
vplot('plot_type',vmat1,'linetype1',...)
vplot('bode_l',top_axis_limits,bottom_axis_limits,vmat1,vmat2,...)
```

Description

The vplot command calls the standard MATLAB plot command for plotting. The optional plot_type argument specifies the type of graph, and selects between the various logarithmic or linear graph types. The plot_type specification choices are

iv,d	matrix (decimal) vs. independent variable
iv,m	magnitude vs. independent variable
iv,lm	log(magnitude) vs. independent variable
iv,p	phase vs. independent variable
liv,d	matrix vs. log(independent variable)
liv,m	magnitude vs. log(independent variable)
liv,lm	log(magnitude) vs. log(independent variable)
liv,p	phase vs. log(independent variable)
ri	real vs. imaginary (parametrized by independent variable)
nyq	real vs. imaginary (parametrized by independent variable)
nic	Nichols chart
bode	Bode magnitude and phase plots
bode_g	Bode magnitude and phase plots with grids
bode_l	Bode magnitude and phase plots with axis limits
bode_gl	Bode magnitude and phase plots with grids and axis limits

If no plot_type specification is given the default is 'iv,d'.

The bode_l and bode_gl plot_type specifications require that the second and third arguments are the desired axis limits for the top and bottom plots. These are simply the 1×4 vectors to be used as arguments for the axis command.

The remaining arguments of vplot take the same form as the MATLAB plot command. Line types (for example,'+', 'g-.', or '*r') can be optionally specified after any VARYING matrix argument.

There is a subtle distinction between CONSTANT and VARYING matrices with only one independent variable. A CONSTANT is treated as such across all independent variables, and consequently shows up as a line on any graph with the independent variable as an axis. A VARYING matrix with only one independent variable will always show up as a point. You may need to specify one of the more obvious point types in order to see it (e.g., '+', 'x', etc.).

Examples

Two SISO second-order systems are created, and their frequency responses are calculated for each over different frequency ranges.

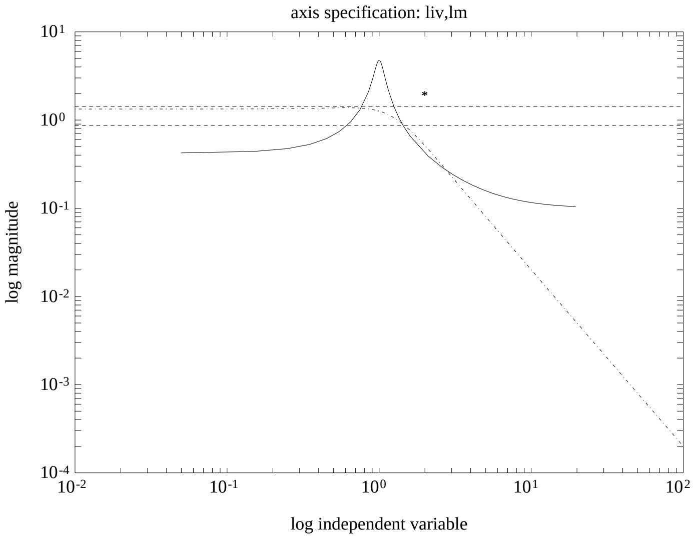
```
a1 = [-1,1;-1,-0.5];
b1 = [0;2]; c1 = [1,0]; d1 = 0;
sys1 = pck(a1,b1,c1,d1);
minfo(sys1)
system:2 states1 outputs1 inputs
a2 = [-.1,1;-1,-0.05];
b2 = [1;1]; c2 = [-0.5,0]; d2 = 0.1;
sys2 = pck(a2,b2,c2,d2);
minfo(sys2)
system:2 states1 outputs1 inputs
omega = logspace(-2,2,100);
sys1_g = frsp(sys1,omega);
omega2 = [ [0.05:0.1:1.5] [1.6:.5:20] [0.9:0.01:1.1] ];
omega2 = sort(omega2);
sys2_g2 = frsp(sys2,omega2);
```

A VARYING matrix with a single independent variable is also created. Note the distinction between this and the CONSTANT matrix in the subsequent plots.

```
rspot = vpck(sqrt(2)-sqrt(2)*i,2);
minfo(rspot)
varying:1 pts1 rows1 cols
```

The following plot uses the 'liv,lm' plot_type specification. Note that the CONSTANT matrix is seen over all values of the independent variable. This is only true because it is displayed as a line type. If it were displayed as a point, then one would see points only on each of the side axes. The single valued VARYING matrix (rspot) is shown only at the appropriate independent variable value.

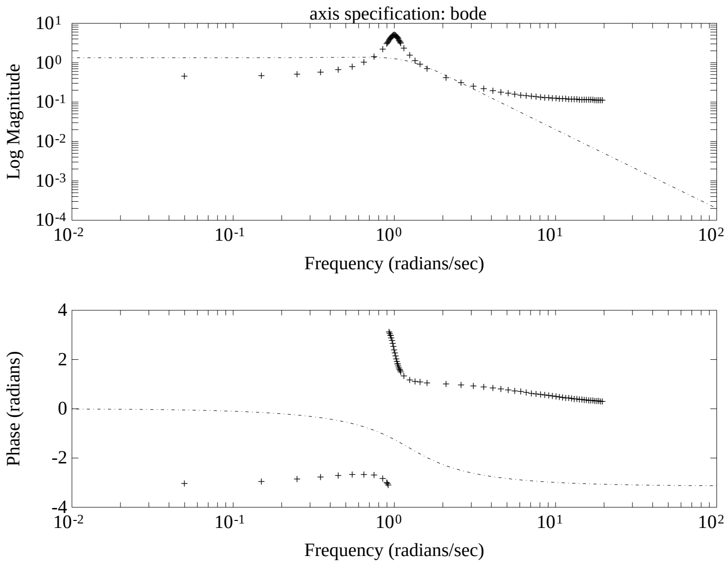
```
vplot('liv,lm',sys1_g,'b-.',[1+i;0.5-0.707*i],'g--',...
rspot,'r*',sys2_g2);
xlabel('log independent variable')
ylabel('log magnitude')
title('plot_type specification: liv,lm')
```



You can customize vplot to select the type of axis uses for log magnitude and phase plots. The default is to plot the log magnitude on a base 10 scale and plot phase in radians. It is a simple modification to select a dB scale and phase in degrees. Documentation of the modification is provided in the M-file vplot. You can copy the command vplot to a private directory (for example, matlab/toolboxes/mu_cmds on UNIX systems) and make the appropriate modifications.

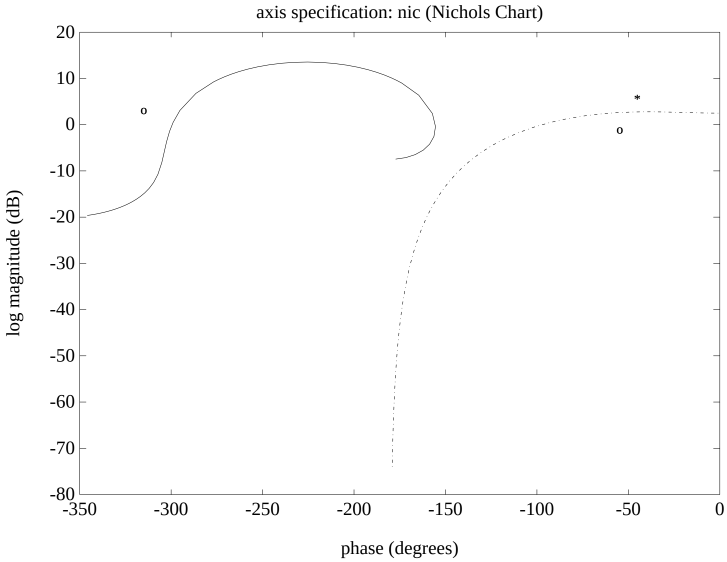
Several control design plot functions are also provided. These are bode, nic, and nyq, for Bode, Nichols, and Nyquist, respectively. The following three plots demonstrate each of these commands.

```
vplot('bode',sys1_g,'b',sys2_g2,'g+');
title('plot_type specification: bode')
```



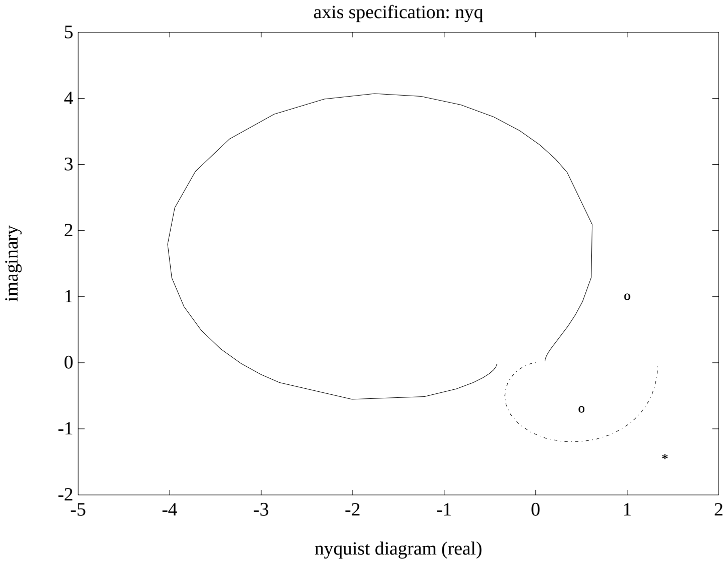
The log magnitude and phase axes are labeled automatically. You can change these labels. Documentation for doing this is in the Help facility for vplot.

```
vplot('nic',sys1_g,'b-.',[1+i;0.5-0.707*i],'go',rspot,...
rspot,'r*',sys2_g2);
title('plot_type specification: nic')
xlabel('phase (degrees)')
ylabel('log magnitude (dB)')
title('plot_type specification: nic (Nichols Chart)')
```



The default axis scale selection for the Nichols plot is dB versus phase in degrees. This corresponds to the usual choice for this plot and can be different from the axis scale selection for bode, liv, lm, liv, p, etc. Again you can change this if required.

```
vplot('nyq',sys1_g,'b-.',[1+i;0.5-0.707*i],'go',rspot,...
xlabel('nyquist diagram (real)')
vplot('liv,lm',sys1_g,'b-.',[1+i;0.5-0.707*i],'g--',...
rspot,'r*',sys2_g2);
ylabel('imaginary')
title('plot_type specification: nyq')
```



See Also `plot`

vpoly, vroots

Purpose

Find coefficients and roots of a characteristic polynomial from a CONSTANT or VARYING matrix

Syntax

```
matout = vpoly(matin)
vecout = vpoly(vecin)
vecout = vroots(vecin)
```

Description

`vpoly` forms an $n + 1$ element VARYING row vector whose elements form the coefficients of the characteristic polynomial, $det(sI$ – `matin(i)`), if `matin` is an $n \times n$ VARYING matrix. The coefficients are ordered in descending powers of s. If the input is a column vector `vecin` containing the roots of a polynomial, `vpoly(vecin)` returns a VARYING row vector whose elements are the coefficients of the corresponding characteristic polynomial.

`vroots` returns as a VARYING column vector `vecout` whose elements are the roots of the polynomial at each independent variable, if `vecin` is a VARYING row vector containing the coefficients of a polynomial. `vpoly` and `vroots` are identical to the MATLAB `poly` and `roots` commands, but also work on VARYING matrices.

Examples

Given a 3 × 3 VARYING matrix, find the characteristic polynomial and its roots. Compare this to finding the eigenvalues of the input matrix via `veig`.

```
see(matin)
3 rows      3 columns
iv = 0.1
1   2   3
4   5   6
7   8   9
iv = 0.4
10   11   12
13   14   15
16   17   18
matout = vpoly(matin);
see(matout)
1 row      4 columns
iv = 0.1
l.0000e+00    -1.5000e+01    -1.8000e+01    -1.4483e-14
iv = 0.4
l.0000e+00    -4.2000e+01    -1.8000e+01     1.2818e-14
```

```
vecout = vroots(matout);
see(vecout)
3 rows       1 column
iv = 0.1
 1.6117e+01
-1.1168e+00
-8.0463e-16

iv = 0.4
 4.2424e+01
-4.2429e-01
7.1212e-16
evals = veig(matin);
see(evals)
3 rows        1 column

iv = 0.1
 1.6117e+01
-1.1168e+00
-8.0463e-16
iv = 0.4
 4.2424e+01
-4.2429e-01
 7.1212e-16
```

Algorithm

vpoly and vroots call the MATLAB poly and roots commands.

See Also

eig, poly, roots, veig

vsvd, vrho, vschur

Purpose

Perform a singular value decomposition, spectral radius and Schur decomposition of a CONSTANT or VARYING matrix

Syntax

```
s = vsvd(matin)
[u,s,v] = vsvd(matin)
out = vrho(matin)
t = vschur(matin)
[u,t] = vschur(matin)
```

Description

vsvd performs a singular value decomposition on a VARYING matrix. It is identical to MATLAB's svd routine, and will work on CONSTANT matrices as well. If there is one output argument, the output is a VARYING matrix with the singular values of matin at each point. If there are three output arguments, [u,s,v], then u is a VARYING matrix with the left singular vectors, s is a VARYING matrix with the singular values, and v is a VARYING matrix with the right singular vectors.

vrho finds the spectral radius, max(abs(eig(xtracti(matin,i)))), at each independent variable of a VARYING matrix.

vschur computes the Schur form of a VARYING matrix for each independent variable. It is identical to the MATLAB schur command, but also works on VARYING matrices. Given two output arguments, vschur returns two VARYING matrices u and t. t corresponds to the Schur form matrix and u is a VARYING unitary matrix such that

```
matin = mmult(u,t,vcjt(u))
```

Examples

Construct a random VARYING matrix and find its singular values.

```
see(matin)
2 rows                 2 columns

iv = 0.1
   0.93040.5269
   0.84620.0920
iv = 0.4
   0.65390.7012
   0.41600.9103
[u,s,v] = vsvd(matin);
```

```
see(u)
2 rows      2 columns
iv = 0.1
0.7884      0.6152
0.6152     -0.7884

iv = 0.4
0.6909     -0.7229
0.7229      0.6909
see(s)

iv = 0.1
1.3400      0.2689

iv = 0.4
1.3681      0.2219
see(v)
2 rows      2 columns

iv = 0.1
0.9359     -0.3522
0.3522      0.9359

iv = 0.4
0.5501     -0.8351
0.8351      0.5501
```

Algorithm

vrho, vschur, and vsvd call the MATLAB commands svd, eig, and schur

See Also

eig, hess, pkvnorm, mu, qz, schur, svd, veig, vnorm

vzoom

Purpose

Freeze plot axes by clicking mouse twice in plot window

Syntax

```
vzoom('axis')
```

Description

vzoom uses the MATLAB functions ginput and axis to freeze the axes by clicking the mouse twice in the plot window that defines minimum and maximum values for *x* and *y*. The clicking may be done in any order.

The axis argument specifies the type of graph, and can select between the various logarithmic or linear graph types, just as in vplot. Unlike vplot, the axis argument is not optional. The axis specification choices are

'iv,d', 'iv,m','iv,p'	decimal, magnitude, or phase vs. independent variable
'liv,d', 'liv,m','liv,p'	decimal, magnitude, or phase vs.
'iv,lm'	log(magnitude) vs. independent variable
'liv,lm'	log(magnitude) vs. log(independent variable)
'ri', 'nyq'	real vs. imaginary (parametrized by independent variable)
'nic'	Nichols chart
'ss'	standard decimal
'll'	log-log
'ls'	semilogx
'sl'	semilogy

Note that the axis specification is the same as for vplot, with the addition of the last four possibilities. The function is not defined for 'bode'.

Examples

The command vzoom(’liv,m’) would be equivalent to

```
[x,y] = ginput(2);
axis([log10([min(x) max(x)]) min(y) max(y)]);
```

An example of the use of vzoom is

```
tf = frsp(nd2sys([ 1 .1],[.1 1]),logspace(-2,2,100));
vplot('nic',tf); vzoom('nic'); vplot('nic',tf); axis;
```

See Also

axis, ginput, vplot

Purpose

A graphical user interface for the MATLAB workspace

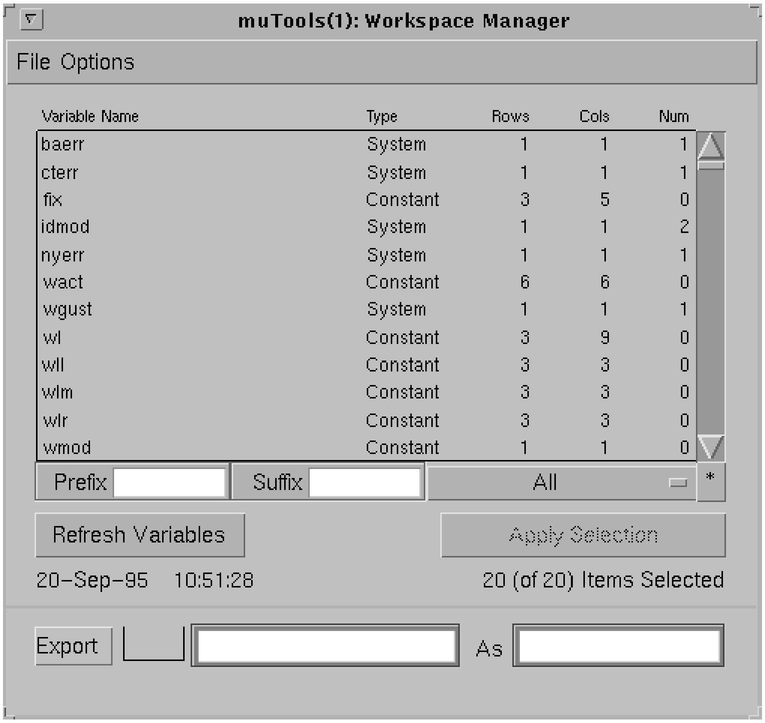
Syntax

```
wsgui
```

Description

`wsgui` is a graphical user interface (GUI) for the MATLAB workspace. It allows you to view, delete, and save variables in the workspace, drag these variables to other μ-Tools GUIs, `dkitgui` and `simgui`, drop boxes, and export variables from the μ-Tools GUI interfaces to the MATLAB workspace.

The `wsgui` Workspace Manager window appears as shown on the following page.



Variable Name	Type	Rows	Cols	Num
baerr	System	1	1	1
cterr	System	1	1	1
fix	Constant	3	5	0
idmod	System	1	1	2
nyerr	System	1	1	1
wact	Constant	6	6	0
wgust	System	1	1	1
wl	Constant	3	9	0
wll	Constant	3	3	0
wlm	Constant	3	3	0
wlr	Constant	3	3	0
wmod	Constant	1	1	0

20-Sep-95 10:51:28

20 (of 20) Items Selected

Each time **Refresh Variables** is pressed, the MATLAB command who is executed, and `minfo` is run to determine the variable type and dimension. This information is displayed in the main scrollable table. The date and time of the last refresh are displayed below the button.

The scrollable table can be moved up/down one page by pressing above/below the slider. Pressing the arrows at the end of the slider moves the table one line.

A filter is used to make viewing of a reduced number of selections easy. The `Prefix, Suffix` and matrix type filters are on the bottom of the scrollable table. The matrix type filter is a pop-up menu to the right of `Suffix`. The `Custom` filter, which is shown if * is pressed, allows you to create a more complicated selection criteria. Press the pushbutton marked with an *; this pushbutton is to the right of the pop-up menu, to switch to the custom filter. A detailed description of `wsgui` is provided in the "Workspace User Interface Tool: wsgui" section in Chapter 6.

Examples

An example of using `wsgui` is shown in the "Workspace User Interface Tool: wsgui" section in Chapter 6.

See Also

`clear`, `save`, `who`

wcperf

Purpose

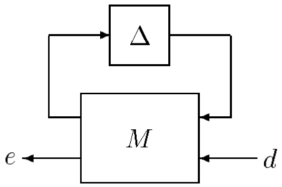
Computes upper and lower bounds for the worst-case gain of a linear system subjected to structured, bounded, LFT perturbations. Also computes worst-case structured perturbation of a specified H_∞

Syntax

```
[deltawc,lowbnd,uppbnd] = wcperf(Mg,uncblk,alpha,npts);
```

Description

The command `wcperf` is associated with the block diagram



where Δ has block structure as defined by Δ, which is described via the matrix `uncblk`. The worst-case performance curve, $f(\alpha)$, is defined as

$$f(\alpha) := \underline{\quad\max\quad}_{\Delta \in \$\Delta,\, max_{w}\sigma(\Delta(j\omega))\eth\alpha} \left\|F_U(M, \Delta)\right\|_\infty$$

Both lower and upper bounds for f are returned as VARYING matrices in `lowbnd` and `uppbnd`. Each VARYING matrix is guaranteed to have at least `npts` values of the independent variable α, spread uniformly between 0 and the stability limit.

The first output argument, `delta_wc`, is the "worst-case" perturbation from Δ with `norm` equal to the value of `alpha`. `delta_wc` has the block-diagonal structure associated with `uncblk`, and causes the LFT F_U (M,Δ_{wc}) to have `norm` equal to the value of `lowbnd` associated with the independent variable value α = alpha.

See Also

`dypert`, `mu`

Purpose Extract a specified portion of a VARYING matrix

Syntax

```
[matout,err] = xtract(mat,iv_low,iv_high)
[matout,err] = xtract(mat,ivdes)
[matout,err] = xtracti(mat,indvindex)
```

Description `xtract` extracts a portion of a VARYING matrix. The independent variable associated with a VARYING matrix monotonically increases (like frequency or time). In the first form, `xtract` is called with three input arguments: a VARYING matrix, a lower bound `iv_low`, and an upper bound `iv_high`. The matrix values associated with any independent variables between `iv_low` and `iv_high` are extracted, and returned as a VARYING matrix. In the second form, `xtract` is called with two arguments. The second argument is a vector of desired independent variable values. For each desired value, the matrix from `mat` with the closest independent variable value (in absolute value) is extracted.

`xtracti` extracts the value of the VARYING matrix at the specific indices `indvindex` of the independent variable. Hence `indvindex` should be an array of positive integers. The extracted matrix is returned as a VARYING matrix.

Examples Extract ranges of independent variable from a VARYING matrix.

```
see(mat)
2 rows2 columns

iv = 0.1
   2.9703e+00 - 2.9703e-0li5.9406e+00 - 5.9406e-0li
   3.9604e+00 - 3.9604e-0li7.9208e+00 - 7.9208e-0li

iv = 0.4
   2.5862e+00 - 1.0345e+00i5.1724e+00 - 2.0690e+00i
   3.4483e+00 - 1.3793e+00i6.8966e+00 - 2.7586e+00i

iv = 0.9
   1.6575e+00 - 1.4917e+00i3.3149e+00 - 2.9834e+00i
   2.2099e+00 - 1.9890e+00i4.4199e+00 - 3.9779e+00i
```

```
matl = xtract(mat,.3,.8);
see(matl)
2 rows2 columns

iv = 0.4
   2.5862e+00 - 1.0345e+00i5.1724e+00 - 2.0690e+00i
   3.4483e+00 - 1.3793e+00i6.8966e+00 - 2.7586e+00i

matl = xtracti(mat,2);
see(matl)
2 rows2 columns

iv = 0.4
   2.5862e+00 - 1.0345e+00i5.1724e+00 - 2.0690e+00i
   3.4483e+00 - 1.3793e+00i6.8966e+00 - 2.7586e+00i
```

See Also

`sel`, `var2con`, `vpck`, `vunpck`, `vfind`

Index