MATRIXx™

Xmath™ Xμ Manual

The MATRIXx products and related items have been purchased from Wind River Systems, Inc. (formerly Integrated Systems, Inc.). These reformatted user materials may contain references to those entities. Any trademark or copyright notices to those entities are no longer valid and any references to those entities as the licensor to the MATRIXx products and related items should now be considered as referring to National Instruments Corporation.

National Instruments did not acquire RealSim hardware (AC-1000, AC-104, PCI Pro) and does not plan to further develop or support RealSim software.

NI is directing users who wish to continue to use RealSim software and hardware to third parties. The list of NI Alliance Members (third parties) that can provide RealSim support and the parts list for RealSim hardware are available in our online KnowledgeBase. You can access the KnowledgeBase at `www.ni.com/support`.

NI plans to make it easy for customers to target NI software and hardware, including LabVIEW real-time and PXI, with MATRIXx in the future. For information regarding NI real-time products, please visit `www.ni.com/realtime` or contact us at `matrixx@ni.com`.

NATIONAL INSTRUMENTS™

April 2004 Edition
Part Number 370760B-01

Worldwide Technical Support and Product Information

ni.com

National Instruments Corporate Headquarters

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

Worldwide Offices

Australia 1800 300 800, Austria 43 0 662 45 79 90 0, Belgium 32 0 2 757 00 20, Brazil 55 11 3262 3599,
Canada (Calgary) 403 274 9391, Canada (Ottawa) 613 233 5949, Canada (Québec) 450 510 3055,
Canada (Toronto) 905 785 0085, Canada (Vancouver) 514 685 7530, China 86 21 6555 7838,
Czech Republic 420 224 235 774, Denmark 45 45 76 26 00, Finland 385 0 9 725 725 11, France 33 0 1 48 14 24 24,
Germany 49 0 89 741 31 30, Greece 30 2 10 42 96 427, India 91 80 51190000, Israel 972 0 3 6393737,
Italy 39 02 413091, Japan 81 3 5472 2970, Korea 82 02 3451 3400, Malaysia 603 9131 0918, Mexico 001 800 010 0793,
Netherlands 31 0 348 433 466, New Zealand 0800 553 322, Norway 47 0 66 90 76 60, Poland 48 22 3390150,
Portugal 351 210 311 210, Russia 7 095 783 68 51, Singapore 65 6226 5886, Slovenia 386 3 425 4200,
South Africa 27 0 11 805 8197, Spain 34 91 640 0085, Sweden 46 0 8 587 895 00, Switzerland 41 56 200 51 51,
Taiwan 886 2 2528 7227, Thailand 662 992 7519, United Kingdom 44 0 1635 523545

For further support information, refer to the *Technical Support Resources and Professional Services* appendix. To comment on the documentation, send email to techpubs@ni.com.

Important Information

Warranty

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

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

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

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

Copyright

Trademarks

Patents

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

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

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

Contents

Chapter 1

Introduction

Xμ is a suite of Xmath functions for the modeling, analysis and synthesis of linear robust control systems. Robust control theory has developed rapidly during the last decade to the point where a useful set of computational tools can be used to solve a wide range of control problems. This theory has already been applied to a wide range of practical problems.

This manual describes the Xμ functions and presents a demonstration of their application. The underlying theory is outlined here and further theoretical details can be found in the many references provided.

It is assumed that the reader is familiar with the use of Xmath; the Xmath Basics manual and the on-line demos are a good way of getting started with Xmath. A good knowledge of control theory and application is also assumed. The more that is known about robust control theory the better as the details are not all covered here.

1.1 Notation

Several font types or capitalization styles are used to distinguish between data objects. The following table lists the various meanings.

Notation	Meaning
PDM	Xmath parameter dependent matrix data object
DYNAMIC SYSTEM	Xmath dynamic system data object

Code examples and function names are set in typewriter font to distinguish them from narrative text.

1.2 Manual Outline

Chapter 2 outlines the applicable robust control theory. Perturbation models and linear fractional transformations form the basis of the modeling framework. The discussion is aimed at an introductory level and not all of the subtleties are covered. The theory continues with an overview of the H_∞ design technique. Again the reader is referred elsewhere for detail of the theory. The robust control methodology covered here is based on the analysis of systems with perturbations. This is covered in some detail as such an understanding is required for effective use of this software. Repeated analysis can be used to improve upon the synthesis; this takes us from the standard H_∞ design method to the more sophisticated μ-synthesis techniques.

The translation between the theoretical concepts and the use of the software is made in Chapter 3. The means of performing typical robust control calculations are discussed in some detail. This chapter also serves to introduce the Xμ functions. The discussion is extended to include some of the relevant Xmath functions. A prior reading of Chapter 2 is helpful for putting this material in context.

The best means of getting an idea of the use of the software is to study completed design examples, given in Chapter 4. These currently includes a design study for an aerospace application. Typical modeling, analysis, synthesis, and simulation studies are illustrated. These studies can be used as initial templates for the user's application.

Chapter 6 is a function reference guide containing a formal description of each function. This is similar to that given via the on-line help capability. Functions are listed in relevant groupings at the start of the chapter. This gives an overview of some of the software capabilities.

1.3 How to avoid really reading this Manual

The layout of the manual proceeds from introduction to background to syntax detail to application descriptions. This may be tediously theoretical for some. If you are one of those that considers reading the manual as the option of last resort[1] then go directly to the applications (Chapter 4). If you have no prior Xmath experience then skimming through Chapter 3 is essential. After running the demos and getting a feel for what the software can do look briefly through the theory section.

[1] And it seems that you are now exercising that option

Chapter 2

Overview of the Underlying Theory

2.1 Introduction

The material covered here is taken from a variety of sources. The basic approach is described by Doyle [1, 2], and further elaborated upon by Packard [3]. Summaries have also appeared in work by Smith [4] and others.

Motivating background can be found in the early paper by Doyle and Stein [5]. An overview of the robust control approach, particularly for process control systems, is given by Morari and Zafiriou [6]. The reader can also find a description of the $\mathcal{H}_\infty/\mu$ synthesis robust control approach in [7].

There are a number of descriptions of this approach to practical problems. In the last few years a significant number of these have been described in the proceedings of the American Control Conference (ACC) and the IEEE Control and Decision Conference (CDC). Only some of the early illustrative examples are cited here.

Application of μ synthesis to a shuttle control subsystem is given by Doyle *et al.* [8]. Examples of flexible structure control are described by Balas and coworkers [9, 10, 11, 12] and Smith, Fanson and Chu [13, 14]. There have also been

several studies involving process control applications, particularly high purity distillation columns. These are detailed by Skogestad and Morari in [15, 16, 17, 18]

Section 2.2 introduces robust control perturbation models and linear fractional transformations. Weighted $\mathcal{H}_\infty$ design is covered in Section 2.3. The analysis of closed loop systems with the structured singular value (μ) is overviewed in Section 2.4. Section 2.5 discusses μ synthesis and the D-K iteration. Model reduction is often used to reduce the controller order prior to implementation and this is covered in Section 2.6.

2.1.1 Notation

We will use some fairly standard notation and this is given here for reference.

$\mathcal{R}$	set of real numbers
$\mathcal{C}$	set of complex numbers
$\mathcal{R}^n$	set of real valued vectors of dimension $n \times 1$
$\mathcal{C}^n$	set of complex valued vectors of dimension $n \times 1$
$\mathcal{R}^{n\times m}$	set of real valued matrices of dimension $n \times m$
$\mathcal{C}^{n\times m}$	set of complex valued matrices of dimension $n \times m$
I_n	identity matrix of dimension $n \times n$
0	matrix (or vector or scalar) of zeros of appropriate dimension

The following apply to a matrix, $M \in \mathcal{C}^{n\times m}$.

M^T	transpose of M
M^*	complex conjugate transpose of M
$\lvert M\rvert$	absolute value of each element of M (also applies if M is a vector or scalar)
$\mathrm{Re}\{M\}$	real part of M
$\mathrm{Im}\{M\}$	imaginary part of M
$\dim(M)$	dimensions of M
$\sigma_{\max}(M)$	maximum singular value of M
$\sigma_{\min}(M)$	minimum singular value of M
M_{ij}	element of M in row i, column j. (also used for the i,j partition of a previously defined partition of M)
$\lambda_i(M)$	an eigenvalue of M
$\rho(M)$	spectral radius ($\max_i \lvert\lambda_i(M)\rvert$)
$\lVert M\rVert$	norm of M (see section 2.1.2 for more details)

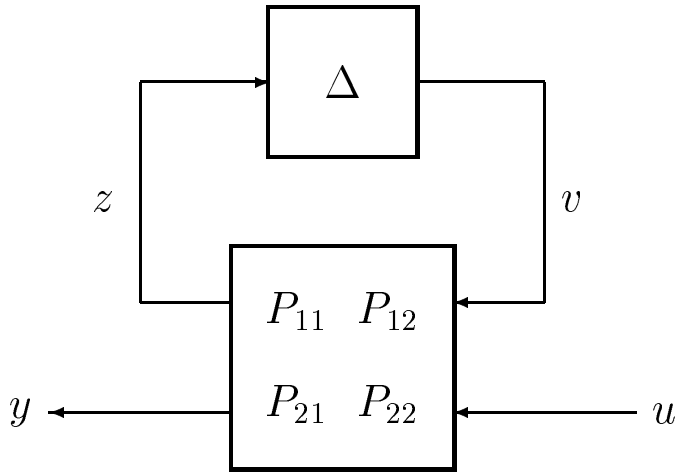


Figure 2.1: The generic robust control model structure

Trace(M) trace of M ($\sum_{i=1}^{n} M_{ii}$)

Block diagrams will be used to represent interconnections of systems. Consider the example feedback interconnection shown in Fig. 2.1. Notice that P has been partitioned into four parts. This diagram represents the equations,

$$\begin{aligned} z &= P_{11}v + P_{12}u \\ y &= P_{21}v + P_{22}u \\ v &= \Delta z. \end{aligned}$$

This type of diagram (and the associated equations) will be used whenever the objects P, z, y, etc., are well defined and compatible. For example P could be a matrix and z, y, etc., would be vectors. If P represented a dynamic system then z, y, etc., would be signals and

$$y = P_{21}v + P_{22}u,$$

is interpreted to mean that the signal y is the sum of the response of system P_{21} to input signal v and system P_{22} to input signal u. In general, we will not be specific about the representation of the system P. If we do need to be more specific about P, then $P(s)$ is the Laplace representation and $p(t)$ is the impulse response.

Note that Figure 2.1 is drawn from right to left. We use this form of diagram because it more closely represents the order in which the systems are written in the corresponding mathematical equations. We will later see that the particular block diagram shown in Figure 2.1 is used as a generic description of a robust control system.

In the case where we are considering a state-space representation, the following notation is also used. Given $P(s)$, with state-space representation,

$$\begin{aligned} sx(s) &= Ax(s) + Bu(s) \\ y(s) &= Cx(s) + Du(s), \end{aligned}$$

we associate this description with the notation,

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

The motivation for this notation comes from the example presented in Section 2.2.4. We will also use this notation to for state-space representation of discrete time systems (where s in the above is replaced by z). The usage will be clear from the context of the discussion.

2.1.2 An Introduction to Norms

A norm is simply a measure of the size of a vector, matrix, signal, or system. We will define and concentrate on particular norms for each of these entities. This gives us a formal way of assessing whether or not the size of a signal is large or small enough. It allows us to quantify the performance of a system in terms of the size of the input and output signals.

Unless stated otherwise, when talking of the size of a vector, we will be using the

Euclidean norm. Given,

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix},$$

the Euclidean (or 2-norm) of x, denoted by $\|x\|$, is defined by,

$$\|x\| = \left(\sum_{i=1}^{n} |x_i| \right)^{1/2}.$$

Many other norms are also options; more detail on the easily calculated norms can be found in the on-line help for the `norm` function. The term spatial-norm is often applied when we are looking at norms over the components of a vector.

Now consider a vector valued signal,

$$x(t) = \begin{bmatrix} x_1(t) \\ \vdots \\ x_n(t) \end{bmatrix}.$$

As well as the issue of the spatial norm, we now have the issue of a time norm. In the theory given here, we concentrate on the 2-norm in the time domain. In otherwords,

$$\|x_i(t)\| = \left(\int_{-\infty}^{\infty} |x_i(t)|^2 \, dt \right)^{1/2}.$$

This is simply the energy of the signal. This norm is sometimes denoted by a subscript of two, i.e. $\|x_i(t)\|_2$. Parseval's relationship means that we can also express this norm in the Laplace domain as follows,

$$\|x_i(s)\| = \left(\frac{1}{2\pi} \int_{-\infty}^{\infty} |x_i(j\omega)|^2 \, d\omega \right)^{1/2}.$$

For persistent signals, where the above norm is unbounded, we can define a power norm,

$$\|x_i(t)\| = \left(\lim_{T\to\infty} \frac{1}{2T} \int_{-T}^{T} |x_i(t)|^2 \, dt \right)^{1/2}. \tag{2.1}$$

The above norms have been defined in terms of a single component, $x_i(t)$, of a vector valued signal, $x(t)$. The choice of spatial norm determines how we combine these components to calculate $\|x(t)\|$. We can mix and match the spatial and time parts of the norm of a signal. In practice it usually turns out that the choice of the time norm is more important in terms of system analysis. Unless stated otherwise, $\|x(t)\|$ implies the Euclidean norm spatially and the 2-norm in the time direction.

Certain signal spaces can be defined in terms of their norms. For example, the set of signals $x(t)$, with $\|x(t)\|_2$ finite is denoted by $\mathcal{L}_2$. The formal definition is,

$$\mathcal{L}_2 = \left\{ x(t) \,\middle|\, \|x(t)\| < \infty \right\}.$$

A similar approach can be taken in the discrete-time domain. Consider a sequence, $\{x(k)\}_{k=0}^{\infty}$, with 2-norm given by,

$$\|x(k)\|_2 = \left(\sum_{k=0}^{\infty} |x(k)|^2 \right)^{1/2}.$$

A lower case notation is used to indicate the discrete-time domain. All signals with finite 2-norm are therefore,

$$l_2 = \left\{ x(k), k = 0, \ldots, \infty \,\middle|\, \|x(k)\|_2 < \infty \right\}.$$

We can essentially split the space $\mathcal{L}_2$ into two pieces, $\mathcal{H}_2$ and $\mathcal{H}_2^{\perp}$. $\mathcal{H}_2$ is the set of elements of $\mathcal{L}_2$ which are analytic in the right-half plane. This can be thought of as those which have their poles strictly in the left half plane; i.e. all stable signals. Similarly, $\mathcal{H}_2^{\perp}$ are all signal with their poles in the left half plane; all strictly unstable

signals. Strictly speaking, signals in $\mathcal{H}_2$ or $\mathcal{H}_2^\perp$ are not defined on the $\jmath\omega$ axis. However we usually consider them to be by taking a limit as we approach the axis.

A slightly more specialized set is $\mathcal{RL}_2$, the set of real rational functions in $\mathcal{L}_2$. These are strictly proper functions with no poles on the imaginary axis. Similarly we can consider $\mathcal{RH}_2$ as strictly proper stable functions and $\mathcal{RH}_2^\perp$ as strictly proper functions with no poles in $\mathrm{Re}(s)< 0$. The distinction between $\mathcal{RL}_2$ and $\mathcal{L}_2$ is of little consequence for the sorts of analysis we will do here.

The concept of a unit ball will also come up in the following sections. This is simply the set of all signals (or vectors, matrices or systems) with norm less than or equal to one. The unit ball of $\mathcal{L}_2$, denoted by $\mathbf{B}\mathcal{L}_2$, is therefore defined as,

$$\mathbf{B}\mathcal{L}_2 = \left\{ x(t) \;\middle|\; \|x(t)\|_2 < 1 \right\}.$$

Now let's move onto norms of matrices and systems. As expected the norm of a matrix gives a measure of its size. We will again emphasize only the norms which we will consider in the following sections. Consider defining a norm in terms of the maximum gain of a matrix or system. This is what is known as an induced norm. Consider a matrix, M, and vectors, u and y, where

$$y = Mu.$$

Define, $\|M\|$, by

$$\|M\| = \max_{u,\|u\|<\infty} \frac{\|y\|}{\|u\|}.$$

Because M is obviously linear this is equivalent to,

$$\|M\| = \max_{u,\|u\|=1} \|y\|.$$

The properties of $\|M\|$ will depend on how we define the norms for the vectors u and y. If we choose our usual default of the Euclidean norm then $\|M\|$ is given by,

$$\|M\| = \sigma_{\max}(M),$$

where $\sigma_{\max}$ denotes the maximum singular value. Not all matrix norms are induced from vector norms. The Froebenius norm (square root of the sum of the squares of all matrix elements) is one such example.

Now consider the case where $P(s)$ is a dynamic system and we define an induced norm from $\mathcal{L}_2$ to $\mathcal{L}_2$ as follows. In this case, $y(s)$ is the output of $P(s)u(s)$ and

$$\|P(s)\| = \max_{u(s)\in\mathcal{L}_2} \frac{\|y(s)\|_2}{\|u(s)\|_2}.$$

Again, for a linear system, this is equivalent to,

$$\|P(s)\| = \max_{u(s)\in\mathbf{B}\mathcal{L}_2} \|y(s)\|_2 .$$

This norm is called the ∞-norm, usually denoted by $\|P(s)\|_\infty$. In the single-input, single-output case, this is equivalent to,

$$\|P(s)\|_\infty = \text{ess} \sup_\omega |P(\jmath\omega)|.$$

This formal definition uses the term ess sup, meaning essential supremum. The "essential" part means that we drop all isolated points from consideration. We will always be considering continuous systems so this technical point makes no difference to us here. The "supremum" is conceptually the same as a maximum. The difference is that the supremum also includes the case where we need to use a limiting series to approach the value of interest. The same is true of the terms "infimum" (abbreviated to "inf") and "minimum." For practical purposes, the reader can think instead in terms of maximum and minimum.

Actually we could restrict $u(s) \in \mathcal{H}_2$ in the above and the answer would be the same. In other words, we can look over all stable input signals $u(s)$ and measure the 2-norm of the output signal, $y(s)$. The subscript, ∞, comes from the fact that we are looking for the supremum of the function on the $\jmath\omega$ axis. Mathematicians sometimes refer to this norm as the "induced 2-norm." Beware of the possible confusion when reading some of the mathematical literature on this topic.

If we were using the power norm above (Equation 2.1) for the input and output norms, the induced norm is still $\|P(s)\|_\infty$.

The set of all systems with bounded ∞-norm is denoted by $\mathcal{L}_\infty$. We can again split this into stable and unstable parts. $\mathcal{H}_\infty$ denotes the stable part; those systems with $|P(s)|$ finite for all $\mathrm{Re}(s) > 0$. This is where the name "$\mathcal{H}_\infty$ control theory" originates, and we often call this norm the $\mathcal{H}_\infty$-norm. Again we can restrict ourselves to real rational functions, so $\mathcal{RL}_\infty$ is the set of proper transfer functions with no poles on the $\jmath\omega$ axis. Similary, $\mathcal{RH}_\infty$ is the set of proper, stable transfer functions.

Again, we are free to choose a spatial norm for the input and output signals $u(s)$ and $y(s)$. In keeping with our above choices we will choose the Euclidean norm. So if $P(s)$ is a MIMO system, then,

$$\|P(s)\|_\infty = \sup_\omega \sigma_{\max}[P(\jmath\omega)].$$

There is another choice of system norm that will arise in the following sections. This is the $\mathcal{H}_2$-norm for systems, defined as,

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

where $P(\jmath\omega)^*$ denotes the conjugate transpose of $P(\jmath\omega)$ and the trace of a matrix is the sum of its diagonal elements. This norm will come up when we are considering linear quadratic Gaussian (LQG) problems.

2.2 Modeling Uncertain Systems

2.2.1 Perturbation Models for Robust Control

A simple example will be used to illustrate the idea of a perturbation model. We are interested in describing a system by a set of models, rather than just a nominal model. Our uncertainty about the physical system will be represented in an unknown component of the model. This unknown component is a perturbation, Δ, about which we make as few assumptions as possible; maximum size, linearity, time-invariance, etc..

Every different perturbation, Δ, gives a slightly different system model. The complete

robust control model is therefore a set description and we hope that some members of this set capture some of the uncertain or unmodeled aspects of our physical system.

For example, consider the "uncertain" model illustrated in Figure 2.2. This picture is equivalent to the input-output relationship,

$$y = [\ (I + \Delta W_m) P_{nom}\ ]\, u. \tag{2.2}$$

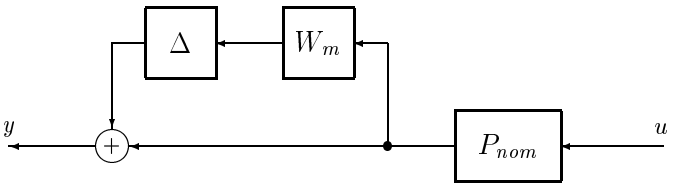


Figure 2.2: Generic output multiplicative perturbation model

In this figure, Δ, W_m and P_{nom} are dynamic systems. The most general form for the theory can be stated with these blocks as elements of $\mathcal{H}_\infty$. For the purposes of calculation we will be dealing with Xmath Dynamic Systems, and in keeping with this we will tend to restrict the theoretical discussion to $\mathcal{RH}_\infty$, stable, proper real rational transfer function matrices.

The only thing that we know about the perturbation, Δ, is that $\|\Delta\|_\infty \leq 1$. Each Δ, with $\|\Delta\|_\infty \leq 1$ gives a different transfer function between u and y. The set of all possible transfer functions, generated in this manner, is called $\mathcal{P}$. More formally,

$$\mathcal{P} = \left\{ (I + \Delta W_m) P_{nom} \;\middle|\; \|\Delta\|_\infty \leq 1 \right\}. \tag{2.3}$$

Now we are looking at a set of possible transfer functions,

$$y(s) = P(s)u(s),$$

where $P(s) \in \mathcal{P}$.

Equation 2.2 represents what is known as a multiplicative output perturbation structure. This is perhaps one of the easiest to look at initially as $W(s)$ can be viewed

as specifying a maximum percentage error between P_{nom} and every other element of $\mathcal{P}$. The system $P_{nom}(s)$ is the element of $\mathcal{P}$ that comes from $\Delta = 0$ and is called the nominal system. In otherwords, for $\Delta = 0$, the input-output relationship is $y(s) = P_{nom}(s)\,u(s)$. As Δ deviates from zero (but remains bounded in size), the nominal system is multiplied by $(I + \Delta W_m(s))$. $W_m(s)$ is a frequency weighting function which allows us the specify the maximum effect of the perturbation for each frequency. Including $W_m(s)$ allows us to model $\mathcal{P}$ with Δ being bounded by one. Any normalization of Δ is simply included in $W_m(s)$.

We often assume that Δ is also linear and time-invariant. This means that $\Delta(\jmath\omega)$ is simply an unknown, complex valued matrix at each frequency, ω. If $\|\Delta\|_\infty \leq 1$, then, at each frequency, $\sigma_{\max}(\Delta(\jmath\omega)) \leq 1$. Section 2.2.3 gives a further discussion on the pros and cons of considering Δ to be linear, time-invariant.

Now consider an example of this approach from a Nyquist point of view. A simple first order SISO system with multiplicative output uncertainty is modeled as

$$y(s) = \Big[(I + W_m(s)\Delta)P_{nom}(s)\Big]\,u(s),$$

where

$$P_{nom}(s) = \frac{1 + 0.05s}{1 + s} \quad \text{and} \quad W_m(s) = \frac{0.1 + 0.2s}{1 + 0.05s}.$$

Figure 2.3 illustrates the set of systems generated by a linear time-invariant Δ, $\|\Delta\|_\infty \leq 1$.

At each frequency, ω, the transfer function of every element of $\mathcal{P}$, lies within a circle, centered at $P_{nom}(\jmath\omega)$, of radius $|P_{nom}(\jmath\omega)W_m(\jmath\omega)|$. Note that for certain frequencies the disks enclose the origin. This allows us to consider perturbed systems that are non-minimum phase even though the nominal system is not.

It is worth pointing out that $\mathcal{P}$ is still a model; in this case a set of regions in the Nyquist plane. This is model set is now able to describe a larger set of system behaviors than a single nominal model. There is still an inevitable mismatch between any model (robust control model set or otherwise) and the behaviors of a physical system.

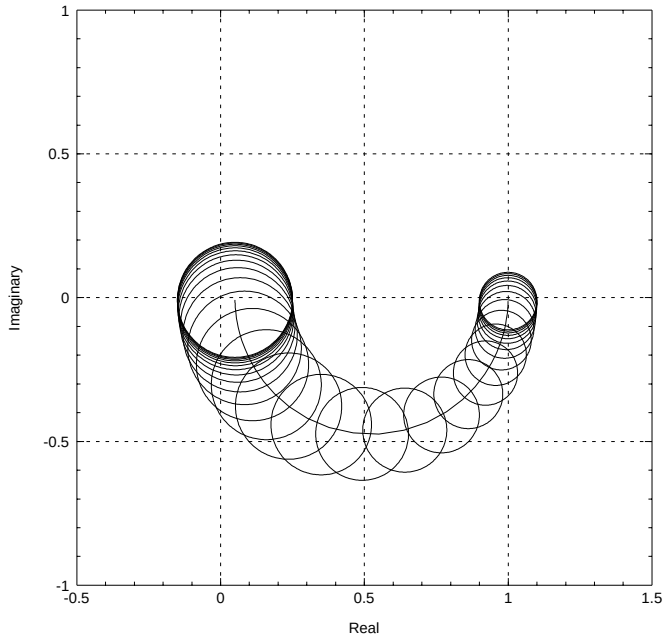


Figure 2.3: Nyquist diagram of the set of systems, $\mathcal{P}$

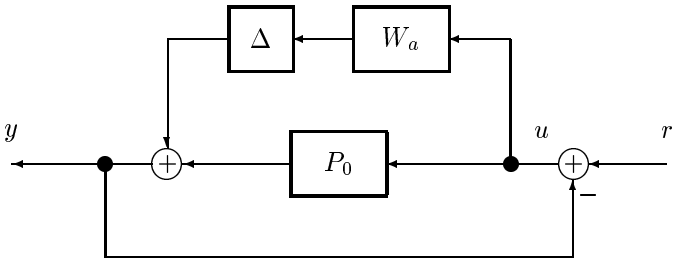


Figure 2.4: Unity gain negative feedback for the example system, $P_0 + \Delta W_a$

2.2.2 Linear Fractional Transformations

A model is considered to be an interconnection of lumped components and perturbation blocks. In this discussion we will denote the input to the model by u, which can be a vector valued signal representing input signals such as control inputs, disturbances, and noise. The outputs signal, denoted in this discussion by y, are also vector valued and can represent system outputs and other variables of interest.

In order to treat large systems of interconnected components, it is necessary to use a model formulation that is general enough to handle interconnections of systems. To illustrate this point consider an affine model description:

$$y = (P_0 + \Delta W_a)u, \quad \|\Delta\|_\infty \leq 1, \tag{2.4}$$

where u is the input and y is the output. Δ again represents an unknown but bounded perturbation. This form of perturbed model is known as an additive perturbation description. While such a description could be applied to a large class of linear systems, it is not general enough to describe the interconnection of models. More specifically, an interconnection of affine models is not necessarily affine. To see this, consider unity gain positive feedback around the above system. This is illustrated in Figure 2.4.

The new input-output transfer function is

$$y = (P_0 + \Delta W_a)[I + (P_0 + \Delta W_a)]^{-1}\, r. \tag{2.5}$$

It is not possible to represent the new system with an affine model. Note that stability questions arise from the consideration of the invertibility of $[I + (P_0 + \Delta W_a)]$.

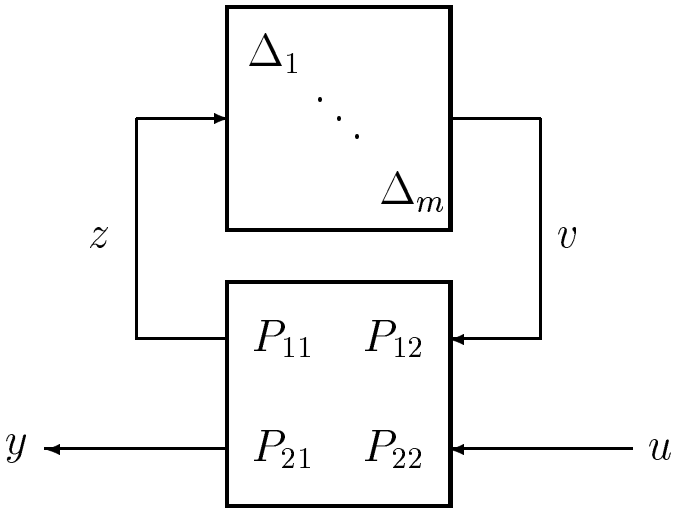


Figure 2.5: Generic LFT model structure including perturbations,Δ

A generic model structure, referred to as a linear fractional transformation (LFT), overcomes the difficulties outlined above. The LFT model is equivalent to the relationship,

$$y = \left[P_{21}\Delta(I - P_{11}\Delta)^{-1}P_{12} + P_{22}\right] u, \tag{2.6}$$

where the Δ is the norm bounded perturbation. Figure 2.5 shows a block diagram equivalent to the system described by Equation 2.6. Because this form of interconnection is widely used, we will give it a specific notation. Equation 2.6 is abbreviated to,

$$y = F_u(P, \Delta)u.$$

The subscript, u, indicates that the Δ is closed in the upper loop. We will also use $F_l(.,.)$ when the lower loop is closed.

In this figure, the signals, u, y, z and v can all be vector valued, meaning that the partitioned parts of P, (P_{11}, etc.) can themselves be matrices of transfer functions.

To make this clear we will look at the perturbed system example, given in Equation 2.4,

in an LFT format. The open-loop system is described by,

$$y = F_u(P_{olp}, \Delta)u,$$

where

$$P_{olp} = \begin{bmatrix} 0 & W_a \\ I & P_0 \end{bmatrix}.$$

The unity gain, negative feedback configuration, illustrated in Figure 2.4 (and given in Equation 2.5) can be described by,

$$y = F_u(G_{clp}, \Delta)r,$$

where

$$G_{clp} = \begin{bmatrix} -W_a(I+P_0)^{-1} & W_a(I+P_0)^{-1} \\ (I+P_0)^{-1} & P_0(I+P_0)^{-1} \end{bmatrix}$$

Figure 2.5 also shows the perturbation, Δ as block structured. In otherwords,

$$\Delta = \text{diag}(\Delta_1, \ldots, \Delta_m). \tag{2.7}$$

This allows us to consider different perturbation blocks in a complex interconnected system. If we interconnect two systems, each with a Δ perturbation, then the result can always be expressed as an LFT with a single, structured perturbation. This is a very general formulation as we can always rearrange the inputs and outputs of P to make Δ block diagonal.

The distinction between perturbations and noise in the model can be seen from both Equation 2.6 and Figure 2.5. Additive noise will enter the model as a component of u. The Δ block represents the unknown but bounded perturbations. It is possible that for some Δ, $(I - P_{11}\Delta)$ is not invertible. This type of model can describe nominally stable systems which can be destabilized by perturbations. Attributing unmodeled effects purely to additive noise will not have this characteristic.

The issue of the invertibility of $(I - P_{11}\Delta)$ is fundamental to the study of the stability of a system under perturbations. We will return to this question in much more detail in Section 2.4. It forms the basis of the μ analysis approach.

Note that Equation 2.7 indicates that we have m blocks, Δ_i, in our model. For notational purposes we will assume that each of these blocks is square. This is actually without loss of generality as in all of the analysis we will do here we can square up P by adding rows or columns of zeros. This squaring up will not affect any of the analysis results. **The software actually deals with the non-square Δ case; we must specify the input and output dimensions of each block**.

The block structure is a m-tuple of integers, $(k_1, \ldots, k_m)$, giving the dimensions of each Δ_i block. It is convenient to define a set, denoted here by $\mathbf{\Delta}$, with the appropriate block structure representing all possible Δ blocks, consistent with that described above. By this it is meant that each member of the set of $\mathbf{\Delta}$ be of the appropriate type (complex matrices, real matrices, or operators, for example) and have the appropriate dimensions. In Figure 2.5 the elements P_{11} and P_{12} are not shown partitioned with respect to the Δ_i. For consistency the sum of the column dimensions of the Δ_i must equal the row dimension of P_{11}. Now define $\mathbf{\Delta}$ as

$$\mathbf{\Delta} = \left\{ \text{diag } (\Delta_1, \ldots, \Delta_m) \;\middle|\; \dim(\Delta_i) = k_i \times k_i \right\}.$$

It is assumed that each Δ_i is norm bounded. Scaling P allows the assumption that the norm bound is one. If the input to Δ_i is z_i and the output is v_i, then

$$\|v_i\| \;=\; \|\Delta_i z_i\| \;\leq\; \|z_i\|.$$

It will be convenient to denote the unit ball of $\mathbf{\Delta}$, the subset of $\mathbf{\Delta}$ norm bounded by one, by $\mathbf{B\Delta}$. More formally

$$\mathbf{B\Delta} = \left\{ \Delta \in \mathbf{\Delta} \;\middle|\; \|\Delta\| \leq 1 \right\}.$$

Putting all of this together gives the following abbreviated representation of the perturbed model,

$$y = F_u(P, \Delta)u, \qquad \Delta \in \mathbf{B\Delta}. \tag{2.8}$$

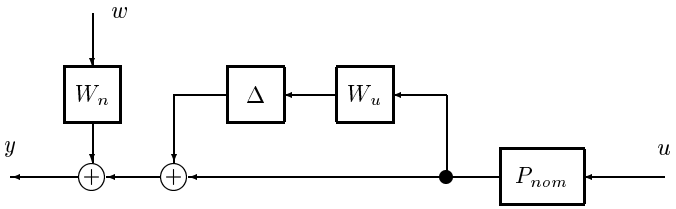


Figure 2.6: Example model: multiplicative output perturbation with weighted output noise

References to a robust control model will imply a description of the form given in Equation 2.8.

As a example, consider one of the most common perturbation model descriptions, illustrated in Figure 2.6. This model represents a perturbed system with bounded noise at the output.

The example model is given by,

$$y = W_n w + (I + \Delta W_u) P_{nom} u.$$

The system W_n is a frequency dependent weight on the noise signal, w. This allows us to use a normalized representation for w. In other words the model includes the assumption that $\|w\|_\infty \leq 1$. Similarly, we assume that $\|\Delta\|_\infty \leq 1$ and W_u is a frequency dependent weight which specifies the contribution of the perturbation at each frequency. In a typical model W_n will be small (assuming that the noise is small compared to the nominal output) and W_u will increase at high frequencies (to capture the likely case that we know less about the model at higher frequencies). The LFT representation of this model is,

$$y = F_u(P, \Delta) \begin{bmatrix} w \\ u \end{bmatrix},$$

where

$$P = \begin{bmatrix} 0 & 0 & W_u P_{nom} \\ I & W_n & P_{nom} \end{bmatrix}$$

Robust control models are therefore set descriptions. In the analysis of such models it is also assumed that the unknown inputs belong to some bounded set. Several choices of set for the unknown signals can be made, leading to different mathematical problems for the analysis. Unfortunately not all of them are tractable. The following section discusses the assumptions typically applied to the robust control models.

2.2.3 Assumptions on P, Δ, and the unknown signals

It will be assumed that the elements of P are either real-rational transfer function matrices or complex valued matrices. The second case arises in the frequency by frequency analysis of systems.

In modeling a system, P_{22} defines the nominal model. Input/output effects not described by the nominal model can be attributed to either unknown signals which are components of the model input (w in the previous example), or the perturbation Δ. Unmodeled effects which can destabilize a system should be accounted for in Δ. The Δ can loosely be considered as accounting for the following. This list is by no means definitive and is only included to illustrate some of the physical effects better suited to description with Δ.

- Unmodeled dynamics. Certain dynamics may be difficult to identify and there comes a point when further identification does not yield significant design performance improvement.
- Known dynamics which have been bounded and included in Δ to simplify the model. As the controller complexity depends on the order of the nominal model a designer may not wish to explicitly include all of the known dynamics.
- Parameter variations in a differential equation model. For example linearization constants which can vary over operating ranges.
- Nonlinear or inconsistent effects. At some point a linear model will no longer account for the residual differences between the behaviors of the model and the physical system.

Several assumptions on Δ are possible. In the most general case Δ is a bounded operator. Alternatively Δ can be considered as a linear time varying multiplier. This assumption can be used to capture nonlinear effects which shift energy between frequencies. Analysis and synthesis are possible with this assumption; Doyle and

Packard [19] discuss the implications of this assumption on robust control theory and we briefly touch upon this in Section 2.4.6. The most common assumption is that Δ is an unknown, norm-bounded, linear time-invariant system.

Systems often do not fall neatly into one of the usual choices of Δ discussed above. Consider a nonlinear system linearized about an operating point. If a range of operation is desired then the linearization constants can be considered to lie within an interval. The model will have a Δ block representing the variation in the linearization constants. If this is considered to be a fixed function of frequency then the model can be considered to be applicable for small changes about any operating point in the range. The precise meaning of small will depend on the effect of the other Δ blocks in the problem.

If the Δ block is assumed to be time-varying then arbitrary variation is allowed in the operating point. However this variation is now arbitrarily fast, and the model set now contains elements which will not realistically correspond to any observed behavior in the physical system.

The robust control synthesis theory gives controllers designed to minimize the maximum error over all possible elements in the model set. Including non-physically motivated signals or conditions can lead to a conservative design as it may be these signals or conditions that determine the worst case error and consequently the controller. Therefore the designer wants a model which describes all physical behaviors of the system but does not include any extraneous elements.

The designer must select the assumptions on P and Δ. An inevitable tradeoff arises between the ideal assumptions given the physical considerations of the system, and those for which good synthesis techniques exist.

The most commonly used assumption is that Δ is a linear time invariant system. This allows us to consider the interconnection, $F_u(P, \Delta)$, from a frequency domain point of view. At each frequency Δ can be taken as an unknown complex valued matrix of norm less than or equal to one. This leads to analyses (covered in Section 2.4) involving the complex structured singular value. The following section discusses more complicated block structures and their use in modeling uncertain systems.

2.2.4 Additional Perturbation Structures

Equation 2.7 introduced a perturbation structure, $\mathbf{\Delta}$ containing m perturbation blocks, Δ_i. This form of perturbation is applicable to a wide range of models for uncertain

systems. We will now look at other possible perturbation structures. For more detail on these structures (in the complex case) refer to Packard and Doyle [20].

Consider a blocks which are of the form scalar $\times$ identity, where the scalar is unknown. In the following we will include q of these blocks in $\mathbf{\Delta}$. The definition of $\mathbf{\Delta}$ is therefore modified to be,

$$\mathbf{\Delta} = \left\{ \mathrm{diag}(\delta_1 I_1, \ldots, \delta_q I_q, \Delta_1, \ldots, \Delta_m) \;\middle|\; \dim(I_j) = l_j \times l_j, \dim(\Delta_i) = k_i \times k_i \right\}. \quad (2.9)$$

The block structure now contains the dimension of the q scalar $\times$ identity blocks and the m full blocks. The block structure is therefore, $(l_1, \ldots, l_q, k_1, \ldots, k_m)$. If $\dim(\Delta) = n \times n$, then these dimensions must be consistent. In otherwords,

$$\sum_{j=1}^{q} l_j + \sum_{i=1}^{m} k_i = n.$$

Note that this block structure collapses to the previously defined structure (Equation 2.7) when $q = 0$.

The most obvious application of a repeated scalar block structure occurs when we know that perturbations occurring in several places in a system are identical (or perhaps just correlated). For example, dynamic models of aircraft often have the altitude (or dynamic pressure) occuring in several places in the model. Naturally the same value should be used in each place and if we model the altitude as an LFT parameter then the repeated scalar $\times$ identity approach is the most appropriate.

This structure also allows us to express uncertain state-space models as LFTs. To illustrate this consider the following discrete time system.

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k) + Du(k). \end{aligned}$$

This digital system has transfer function,

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

$$
\begin{aligned}
&= Cz^{-1}(I - z^{-1}A)^{-1}B + D \\
&= F_u(P_{ss}, z^{-1}I),
\end{aligned}
$$

where P_{ss} is the real valued matrix,

$$
P_{ss} = \begin{bmatrix} A & B \\ C & D \end{bmatrix},
$$

and the scalar $\times$ identity, $z^{-1}I$, has dimension equal to the state dimension of $P(z)$. This is now in the form of an LFT model with a single scalar $\times$ identity element in the upper loop.

One possible use of this is suggested by the following. Define,

$$
\mathbf{\Delta} = \left\{ \delta I_{nx} \,\middle|\, \delta \in \mathcal{C} \right\},
$$

where nx is the state dimension. The set of models,

$$
F_u(P_{ss}, \Delta), \quad \Delta \in \mathbf{B\Delta},
$$

is equivalent to $P(z)$, $|z| \geq 1$. This hints at using this formulation for a stability analysis of $P(z)$. This is investigated further in Section 2.4.6.

In the analyses discussed in Section 2.4 we will concentrate on the assumption that Δ is complex valued at each frequency. For some models we may wish to restrict Δ further. The most obvious restriction is that some (or all) of the Δ blocks are real valued. This is applicable to the modeling of systems with uncertain, real-valued, parameters. Such models can arise from mathematical system models with unknown parameters.

Consider, for example a very simplified model of the combustion characteristics of an automotive engine. This is a simplified version of the model given by Hamburg and Shulman [21]. The system input to be considered is the air/fuel ratio at the carburettor. The output is equivalent to the air/fuel ratio after combustion. This is measured by an oxygen sensor in the exhaust. Naturally, this model is a strong function of the engine speed, v (rpm). We model the relationship as,

$$
y = \mathrm{e}^{-T_d s} \left(\frac{0.9}{1 + T_c s} + \frac{0.1}{1 + s} \right) u,
$$

where the transport delay, T_d, and the combustion lag, T_c, are approximately,

$$T_d = \frac{252}{v} \quad \text{and} \quad T_c = \frac{202}{v}.$$

For the purposes of our example we want to design an air/fuel ratio controller that works for all engine speeds in the range 2,000 to 6,000 rpm. We will use a first order Padé approximation for the delay and express the T_d and T_c relationships in an LFT form with a normalized speed deviation, δ_v.

The dominant combustion lag can be expressed as an LFT on $1/T_c$ as follows,

$$\frac{0.9}{1+T_c s} \quad = \quad F_u(P_{tc}, T_c^{-1}),$$

where

$$P_{tc} = \begin{bmatrix} \frac{-1}{s} & 1 \\ \frac{0.9}{s} & 0 \end{bmatrix}.$$

Note that T_c^{-1} is easily modeled in terms of δ_v,

$$\frac{1}{T_c} = \frac{4000 + 2000\delta_v}{202}, \quad \delta_v \in \mathcal{R},\ |\delta_v| \leq 1.$$

The Padé approximation (with delay time, T_d) is given by

$$\mathrm{e}^{-T_d s} \approx F_u\left(P_{delay}, T_d^{-1}\right),$$

where,

$$P_{delay} = \begin{bmatrix} \frac{-2}{s} & 1 \\ \frac{4}{s} & -1 \end{bmatrix}.$$

Putting all the pieces together gives an engine model in the following fractional form.

$$P(s) = F_u(P_{mod}, \Delta),$$

where

$$P_{mod} = \begin{bmatrix} \frac{-15.87}{s} & \frac{7.14(s-19.8)}{s^2} & \frac{141.5(1+1.006s)}{s(s+1)} \\ 0 & \frac{-9.9}{s} & 9.9 \\ \frac{-27.75}{s} & \frac{-0.9(s-15.8)(s-19.8)}{s^3} & \frac{-17.82(s-15.8)(1+1.006s)}{s(s+1)} \end{bmatrix},$$

and $\Delta \in \mathbf{B\Delta}$, with the structure defined as,

$$\mathbf{\Delta} = \left\{ \delta_v I_2 \,\middle|\, \delta_v \in \mathcal{R} \right\}.$$

To capture the effects of unmodeled high frequency dynamics we will also include an output multiplicative perturbation. If the output multiplicative weight is $W_m(s)$ then the complete open-loop model is,

$$P(s) = F_u(P_{mod}, \Delta), \tag{2.10}$$

where,

$$P_{mod} = \begin{bmatrix} \frac{-15.87}{s} & \frac{7.14(s-19.8)}{s^2} & 0 & \frac{141.5(1+1.006s)}{s(s+1)} \\ 0 & \frac{-9.9}{s} & 0 & 9.9 \\ \frac{-27.75}{s} & \frac{-0.9(s-15.8)(s-19.8)}{s^3} & 0 & \frac{-17.82(s-15.8)(1+1.006s)}{s(s+1)} \\ \frac{-27.75}{s} & \frac{-0.9(s-15.8)(s-19.8)}{s^3} & W_m(s) & \frac{-17.82(s-15.8)(1+1.006s)}{s(s+1)} \end{bmatrix}$$

and $\Delta \in \mathbf{B\Delta}$, with the structure defined as,

$$\mathbf{\Delta} = \left\{ \mathrm{diag}(\delta_v I_2, \Delta_1) \,\middle|\, \delta_v \in \mathcal{R},\ \Delta_1 \in \mathcal{C} \right\}.$$

Note that this is an LFT with a repeated real-valued parameter, δ_v ($|\delta_v| \leq 1$), and a complex perturbation, Δ_1 ($|\Delta_1| \leq 1$).

Note that as $\mathcal{R} \subset \mathcal{C}$, assuming that $\Delta \in \mathcal{C}^{n \times n}$, always covers the case where some of the Δ_i (or δ_j) are more appropriately modeled as real-valued. However this may be potentially conservative as the $\Delta \in \mathcal{C}^{n \times n}$, allows many more systems in the model set. In this case it would be somewhat better to consider combining the effects of δ_v and Δ_1 into a single complex valued Δ with an appropriate weight.

In principle, if we have additional information about the system (some $\delta_j \in \mathcal{R}$, for example) then we should use this information. Performing analyses with real valued perturbations is currently at the forefront of the structured singular value research. We will return to this issue in more detail when we cover the analysis methods (Section 2.4).

2.2.5 Obtaining Robust Control Models for Physical Systems

Obtaining a model of the above form is where the real engineering comes in. A designer must model and identify the physical system to arrive at such a model. This is usually an iterative process whereby designs are performed and then tested on the system. In this way the designer often obtains a feeling for the adequacy of the model.

The best way of studying the modeling problem is to look at the documented experiences of others applying these approaches; particularly in similar applications. The citations given in the Section 2.1 will be useful in this regard. There are also approaches addressing the problem of obtaining LFT models from descriptions with variable state-space matrix coefficients [22, 23]. In the area of SISO process control, Laughlin *et al.* [24] describe the relationship between uncertainties in time constant, delay, and gain, and a suitable Δ weighting. Models of this form are often applicable to process control.

There is little formal theory addressing the robust control modeling problem although this is an area of increasing interest. A recent workshop proceedings volume on the subject is an excellent reference for those interested in this area [25]. Other references can be found in the review article by Gevers [26].

An area of work, known as identification in $\mathcal{H}_\infty$, looks at experimental identification techniques which minimize the worst case $\mathcal{H}_\infty$ error between the physical system and the model. The following works address this issue: [27, 28, 29, 30, 31, 32, 33, 34, 35].

Applying the more standard, probabilistically based, identification techniques to uncertain systems is also receiving attention. Relevant work in this area is described in: [36, 37, 38, 39]

Model validation is the experimental testing of a given robust control model. This can be useful is assessing model quality. This work is covered in the following: [4, 40, 41, 42, 43, 44, 45, 46]. An experimental example is described by Smith [47].

The problems of identifying model parameters in an uncertain model is discussed further in [48, 49, 50]. A nonlinear ad-hoc approach for obtaining suitable multiplicative perturbation models for certain classes of systems is given in [51].

Several researchers are also formalizing the interplay between identification and design in iterative approaches. In practical situations the designer usually ends up with ad-hoc identification/design iterations. The work in this area is described in [52, 53, 54, 55, 56].

On reading the above works, one will get the impression that this area is the most poorly developed of the current robust control theory. In obtaining these models engineering judgement is of paramount importance. The users of this software are encouraged to document their experiences and bring this work to the authors' attention.

2.3 $\mathcal{H}_\infty$ and $\mathcal{H}_2$ Design Methodologies

The generic synthesis configuration is illustrated in LFT form in Figure 2.7. Here $P(s)$ is referred to as the interconnection structure. The objective is to design $K(s)$ such that the closed loop interconnection is stable and the resulting transfer function from w to e (denoted by $G(s)$),

$$\begin{aligned} e &= F_l[P(s), K(s)]w, \\ &= G(s)w, \end{aligned}$$

satisfies a norm objective.

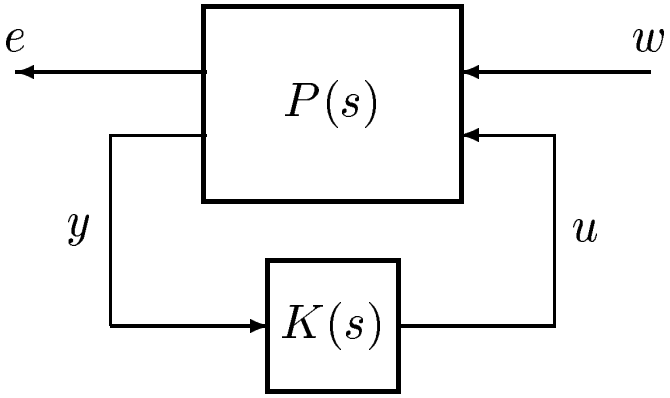


Figure 2.7: LFT configuration for controller synthesis, $G(s) = F_l[P(s), K(s)]$

Note that the interconnection structure, $P(s)$, given here, differs from that discussed in the previous section. Here we set up $P(s)$ so that the input, w, is the unknown signals entering our system. Typical examples would be sensor noise, plant disturbances or tracking commands. The output, e, represent signals that we would like to make small. In an engineering application these could include actuator signals and tracking errors.

The signal y is the measurement available to the controller, $K(s)$. In any realistic problem, some weighted component of w would be added to y to model sensor noise. The output of the controller, u, is our actuation input to the system. Again, a reasonable engineering problem would include a weighted u signal as a component of the penalty output, e.

The interconnection structure, $P(s)$, also contains any frequency weightings on the signals e and w. Weightings on components of e are used to determine the relative importance of the various error signals. Weight functions on w indicate the relative expected size of the unknown inputs.

Xμ provides functions to calculate the controllers minimizing either the $\mathcal{H}_2$ or $\mathcal{H}_\infty$ norm of $G(s)$. We will cover both of these approaches in the context of the design problem illustrated in Figure 2.7.

Note that neither of these design approaches takes advantage of any information about structured perturbations occuring within the model. The following discussion can be considered as applying to a nominal design problem. Section 2.5 uses D-K iteration to

extend these approaches to the case where $P(s)$ is replaced by $F_u(P(s), \Delta)$, $\Delta \in \mathbf{B}\mathbf{\Delta}$.

2.3.1 $\mathcal{H}_\infty$ Design Overview

Again, recall from Section 2.1.2, the $\mathcal{H}_\infty$ is norm of $G(s)$ is,

$$\|G(s)\|_\infty = \sup_\omega \sigma_{\max}[G(\jmath\omega)].$$

The $\mathcal{H}_\infty$ norm is the induced $\mathcal{L}_2$ to $\mathcal{L}_2$ norm. Therefore minimizing the $\mathcal{H}_\infty$ norm of $G(s)$ will have the effect of minimizing the worst-case energy of e over all bounded energy inputs at w.

Consider $\gamma(K)$ to be the closed loop $\mathcal{H}_\infty$ norm achieved for a particular controller K. In other words,

$$\gamma(K) = \|F_l(P, K)\|_\infty.$$

There is a choice of controller, K, which minimizes $\gamma(K)$. This is often referred to as the optimal value of γ and is denoted by γ_{opt}. Furthermore, there is no stabilizing controller which satisfies,

$$\|G(s)\|_\infty < \gamma_{opt}.$$

In a particular design problem, γ_{opt} is not known *a priori*. Therefore the functions calculating the $\mathcal{H}_\infty$ controller use some form of optimization to obtain a value of γ close to γ_{opt}.

The first approaches to the solution of this problem were described by Doyle [1]. The book by Francis [57] gives a good overview of the early version of this theory. A significant breakthrough was achieved with the development of state-space calculation techniques for the problem. These are discussed in the paper colloquially known as DGKF [58]. The algorithmic details are actually given by Glover and Doyle [59].

2.3.2 Assumptions for the $\mathcal{H}_\infty$ Design Problem

There are several assumptions required in order to achieve a well-posed design problem. The DGKF paper gives a state-space solution to the $\mathcal{H}_\infty$ design problem and we will use a similar notation here.

Consider the open loop state-space representation of $P(s)$, partitioned according to the signals shown in Figure 2.7,

$$P(s) = \left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{array}\right]. \tag{2.11}$$

We will assume that $P(s)$ is a minimal representation. The following assumptions are required for a well-posed problem.

(i) (A, B_2) is stabilizable and (C_2, A) is detectable;

(ii) D_{12} and D_{21} are full rank;

(iii) The matrix,

$$\left[\begin{array}{cc} A - \jmath\omega I & B_2 \\ C_1 & D_{12} \end{array}\right],$$

has full column rank for all $\omega \in \mathcal{R}$;

(iv) The matrix,

$$\left[\begin{array}{cc} A - \jmath\omega I & B_1 \\ C_2 & D_{21} \end{array}\right],$$

has full row rank for all $\omega \in \mathcal{R}$.

Item (i) is required so that input-output stability is equivalent to internal stability. If it is not satisfied then there are unstable modes which cannot be stabilized by any $K(s)$.

Items (ii) and (iii) mean that, at every frequency, there is no component of the output signal, e, that cannot be influenced by the controller. Similarly, items (ii) and (iv) mean

that the effect of all disturbances, w, at every frequency, can be measured by the controller. If either of these conditions are not met then the problem could be ill-posed.

It is possible to violate these conditions by using pure integrators as design weights. While this could still give a meaningful design problem, solution via the state-space $\mathcal{H}_\infty$ approach requires that an approximation be used for the integrator weight. If item (*iii*) or (*iv*) is violated at $\omega = 0$, then the integrator should be replaced with very low frequency pole.

2.3.3 A Brief Review of the Algebraic Riccati Equation

Solution of the $\mathcal{H}_\infty$ design problem requires the solution of coupled Algebraic Riccati Equations (AREs). This is illustrated in more detail in the next section. Here we give a very brief review of the Riccati equation and the most common solution techniques. Some knowledge of this area is helpful because the design software displays variables related to the Riccati solutions and the user has the option of adjusting several software tolerances relating to these solutions. The notation used here comes from DGKF [58].

The matrix equation,

$$A^T X + XA + XRX - Q = 0,$$

is an ARE. Given A, R and Q (with R and Q symmetric), we are interested in finding a symmetric positive definite solution, X. In other words, $X = X^T \geq 0$. With this ARE we associate a Hamiltonian matrix, denoted by H,

$$H = \begin{bmatrix} A & R \\ Q & -A^T \end{bmatrix}.$$

If $\dim(A) = n \times n$, then $\dim(H) = 2n \times 2n$. Assume that H has no $\jmath\omega$ axis eigenvalues. The structure of H means that it has n stable ($\mathrm{Re}\{s\} < 0$) and n unstable ($\mathrm{Re}\{s\} > 0$) eigenvalues.

Now consider finding a basis for the stable eigenvalues. Stacking the basis vectors together will give a $2n \times n$ matrix,

$$\begin{bmatrix} X_1 \\ X_2 \end{bmatrix}.$$

We have partitioned the matrix into two $n \times n$ blocks, X_1 and X_2. If X_1 is invertible, then

$$X = X_2 X_1^{-1},$$

is the unique, stabilizing solution to the ARE. The ability to form X doesn't depend on the particular choice of X_1 and X_2.

Given a Hamiltonian, H, we say that $H \in \mathrm{dom(Ric)}$ if H has no $\jmath\omega$ axis eigenvalues and the associated X_1 matrix is invertible. Therefore, if $H \in \mathrm{dom(Ric)}$, we can obtain a unique stabilizing solution, X. This mapping, from H to X, is often written as the function, $X = \mathrm{Ric}(H)$.

To give an idea of the application of the ARE consider the following lemma (taken from DGKF).

Lemma 1 *Suppose* $H \in \mathrm{dom(Ric)}$ *and* $X = \mathrm{Ric}(H)$. *Then:*

a) X *is symmetric;*

b) X *satisfies the ARE,*

$$A^T X + XA + XRX - Q = 0;$$

c) $A + RX$ *is stable.*

This is of course the well know result relating AREs to the solution of stabilizing state feedback controllers.

AREs can also be used in calculating the $\mathcal{H}_\infty$-norm of a state-space system. The approach outlined here is actually that used in the software for the calculation of $\|P(s)\|_\infty$. Consider a stable system,

$$P(s) = \left[\begin{array}{c|c} A & B \\ \hline C & 0 \end{array}\right].$$

Choose $\gamma > 0$ and form the following Hamiltonian matrix,

$$H = \begin{bmatrix} A & \gamma^{-2}BB^T \\ -C^TC & -A^T \end{bmatrix}.$$

The following lemma gives a means of checking whether or not $\|P(s)\|_\infty < \gamma$. A proof of this lemma is given in DGKF although it is based on the work of Anderson [60], Willems [61] and Boyd *et al.* [62].

Lemma 2 *The following conditions are equivalent:*

a) $\|P(s)\|_\infty < \gamma$*;*

b) H *has no eigenvalues on the* $\jmath\omega$ *axis;*

c) $H \in \mathrm{dom}(\mathrm{Ric})$*;*

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

As the above illustrates, AREs play a role in both stabilization and $\mathcal{H}_\infty$-norm calculations for state-space systems. Before giving more detail on the $\mathcal{H}_\infty$ design problem (Section 2.3.4), we will discuss some of the issues that arise in the practical calculation of ARE solutions.

We can summarize an ARE solution method as follows:

(*i*) Form the Hamiltonian, H.

(*ii*) Check that H has no $\jmath\omega$ axis eigenvalues.

(*iii*) Find a basis for the stable subspace of H.

(*iv*) Check that X_1 is invertible.

(*v*) Form $X = X_2X_1^{-1}$.

The first issue to note is that it is difficult to numerically determine whether or not H has $\jmath\omega$ axis eigenvalues. A numerical calculation of the eigenvalues is unlikely to give

any with a zero real part. In practice we must use a tolerance to determine what is considered as a zero real part.

Finding a basis for the stable subspace of H involves either an eigenvalue or Schur decomposition. Numerical errors will be introduced at this stage. In most cases using an eigenvalue decomposition is faster and less accurate than using a Schur decomposition. Similarly, forming $X = X_2 X_1^{-1}$ will also introduce numerical errors. The Schur solution approach, developed by Laub *et al.* [63, 64, 65], is currently the best numerical approach to solving the ARE and is used in the software as the default method. An overview of invariant subspace methods for ARE solution is given by Laub [66]. Accurate solution of the ARE is still very much an active area of research.

2.3.4 Solving the $\mathcal{H}_\infty$ Design Problem for a Special Case

We will now look at the $\mathcal{H}_\infty$ design problem for a simplifying set of assumptions. The general problem (with assumptions given in Section 2.3.2) can be transformed into the simplified one given here via scalings and other transformations. The simplified problem illustrates the nature of the solution procedure and is actually the problem studied in DGKF. The formulae for the general problem are given in Glover and Doyle [59]. The software solves the general problem.

Consider the following assumptions, with reference to the system in Equation 2.11:

(i) (A,B_1) stabilizable and (C_1,A) detectable;

(ii) (A,B_2) stabilizable and (C_2,A) detectable;

(iii) $D_{12}^T [\, C_1 \;\; D_{12} \,] = [\, 0 \;\; I \,]$;

(iv) $$\begin{bmatrix} B_1 \\ D_{21} D_{21}^T \end{bmatrix} = \begin{bmatrix} 0 \\ I \end{bmatrix};$$

(v) $D_{11} = D_{22} = 0$.

Assumption (i) is included in DGKF for technical reasons. The formulae are still correct if it is violated. Note that, with these assumptions,

$$e = C_1 x + D_{12} u,$$

and the components, $C_1 x$ and $D_{12} u$ are orthogonal. D_{12} is also assumed to be normalized. This essentially means that there is no cross-weighting between the state and input penalties. Assumption (iv) is the dual of this; the input and unknown input (disturbance and noise) affect the measurement, y, orthogonally, with the weight on the unknown input being unity.

To solve the $\mathcal{H}_\infty$ design problem we define two Hamiltonian matrices,

$$H_\infty = \begin{bmatrix} A & \gamma^{-2} B_1 B_1^T - B_2 B_2^T \\ -C_1^T C_1 & -A^T \end{bmatrix},$$

and

$$J_\infty = \begin{bmatrix} A^T & \gamma^{-2} C_1^T C_1 - C_2^T C_2 \\ -B_1 B_1^T & -A \end{bmatrix}.$$

The following theorem gives the solution to the problem.

Theorem 3 *There exists a stabilizing controller satisfying $\|G(s)\|_\infty < \gamma$ if and only if the following three conditions are satisfied:*

a) $H_\infty \in \mathrm{dom}(\mathrm{Ric})$ *and* $X_\infty = \mathrm{Ric}(H_\infty) \geq 0$.

b) $J_\infty \in \mathrm{dom}(\mathrm{Ric})$ *and* $Y_\infty = \mathrm{Ric}(J_\infty) \geq 0$.

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

When these conditions are satisfied, one such controller is,

$$K_\infty(s) = \left[\begin{array}{c|c} \hat{A}_\infty & -Z_\infty L_\infty \\ \hline F_\infty & 0 \end{array}\right],$$

where,

$$F_\infty \;\; = \;\; -B_2^T X_\infty$$

$$
\begin{aligned}
L_\infty &= -Y_\infty C_2^T \\
Z_\infty &= (I - \gamma^{-2} Y_\infty X_\infty)^{-1} \\
\hat{A}_\infty &= A + \gamma^{-2} B_1 B_1^T X_\infty + B_2 F_\infty + Z_\infty L_\infty C_2.
\end{aligned}
$$

Actually, the above formulation can be used to parametrize all stabilizing controllers which satisfy, $\|G(s)\|_\infty < \gamma$. This can be expressed as an LFT. All such controllers are given by,

$$K_\infty = F_l(M_\infty, Q),$$

where,

$$
M_\infty = \left[\begin{array}{c|cc}
\hat{A}_\infty & -Z_\infty L_\infty & Z_\infty B_2 \\
\hline
F_\infty & 0 & I \\
-C_2 & I & 0
\end{array}\right],
$$

and Q satisfies: $Q \in \mathcal{RH}_\infty$, $\|Q\|_\infty < \gamma$. Note that if $Q = 0$ we get back the controller given in Theorem 3. This controller is referred to as the central controller and it is the controller calculated by the software.

Note also that the controller given above satisfies $\|G\|_\infty < \gamma$. It is not necessarily the controller that minimizes $\|G\|_\infty$ and is therefore referred to as a suboptimal $\mathcal{H}_\infty$ controller. In practice this is not a problem, and may even be an advantage. The optimal $\mathcal{H}_\infty$ controller has properties which may not be desirable from an implementation point of view. One typical property is that the high frequency gain is often large. Suboptimal central controllers seem to be less likely to exhibit this characteristic.

2.3.5 Further Notes on the $\mathcal{H}_\infty$ Design Algorithm

Now that we have covered the problem solution we can look at the areas that might give potential numerical problems. The above results give a means of calculating a controller (if one exists) for a specified γ value. As we mentioned earlier, the smallest such γ is referred to as γ_{opt}. An iterative algorithm is used to find γ close to γ_{opt} and calculate the resulting controller. The algorithm can be stated conceptually as follows:

a) Choose $\gamma \geq \gamma_{\text{opt}}$

b) Form H_∞ and J_∞

c) Check that $H_\infty \in \text{dom(Ric)}$ and $J_\infty \in \text{dom(Ric)}$.

d) Calculate $X_\infty = \text{Ric}(H_\infty)$ and $Y_\infty = \text{Ric}(J_\infty)$

e) Check that $X_\infty \geq 0$ and $Y_\infty \geq 0$

f) Check that $\rho(X_\infty Y_\infty) < \gamma^2$

g) Reduce γ and go to step b).

The value of γ can be reduced until one of the checks at steps c), e) or f) fails. In this case, $\gamma < \gamma_{\text{opt}}$ and we use the X_∞ and Y_∞ of the lowest successful γ calculation to generate the controller. In the Xμ software a bisection search over γ is used to find a γ close to γ_{opt}. If step a) is not satisfied, the routine exits immediately and tells the user to select a higher initial choice for γ.

As part of the check that $H_\infty \in \text{dom(Ric)}$, (and $J_\infty \in \text{dom(Ric)}$) the real part of the eigenvalues is calculated. The software uses a tolerance to determine whether or not to consider these zero. The default tolerance works well in most cases; the user can adjust it if necessary.

In practice determining that X_∞ (and Y_∞) is positive definite involves checking that,

$$\min_i \text{Re}\{\lambda_i(X_\infty)\} \geq -\epsilon.$$

Again, ϵ is a preset tolerance which can be adjusted by the user if necessary.

The third check is that,

$$\rho(\gamma^{-2} X_\infty Y_\infty) < 1.$$

Fortunately this is a relatively well conditioned test.

The software displays the critical variables relating to each of these tests. The minimum real part of the eigenvalues of H_∞ (and J_∞) is displayed. Similarly the minimum

eigenvalue of X_∞ (and Y_∞) is displayed. The ultimate test of the software is to form the closed loop system and check both its stability and norm. We strongly suggest that the user always perform this step.

The numerical issues discussed above are very unlikely to arise in low order systems. Experience has shown that systems with very lightly damped modes are more susceptible to numerical problems than those with more heavily damped modes. However, it has been found to be possible, with the software provided, to design controllers using 60th order interconnection structures with very lightly damped modes.

2.3.6 $\mathcal{H}_2$ Design Overview

Recall from Section 2.1.2 that the $\mathcal{H}_2$ norm of a frequency domain transfer function, $G(s)$, is

$$\|G(s)\|_2 = \left(\frac{1}{2\pi} \int_{-\infty}^{\infty} \text{Trace } [G(\jmath\omega)^* G(\jmath\omega)] \, d\omega \right)^{1/2}.$$

Several characterizations of this norm are possible in terms of input/output signals. For example, if the unknown signals are of bounded energy, $\|G\|_2$ gives the worst case magnitude of the outputs e. Alternatively, if impulses are applied to the inputs of $G(s)$, $\|G(s)\|_2$ gives the energy of the outputs e. $\mathcal{H}_2$ synthesis involves finding the controller which minimizes the $\mathcal{H}_2$ norm of the closed loop system. This is the same the well studied Linear Quadratic Gaussian problem.

2.3.7 Details of the $\mathcal{H}_2$ Design Procedure

The $\mathcal{H}_2$ design procedure is best explained by contrasting it with the $\mathcal{H}_\infty$ procedure explained in the previous sections. There are several differences, the most obvious being that the $\mathcal{H}_2$ design problem always has a unique, minimizing, solution. The other difference is that (in addition to the four conditions given in Section 2.3.2) D_{11} is required to be zero, even in the general case. If this condition is violated no controller will give a finite $\mathcal{H}_2$ norm for the closed loop system as it will not roll off as the frequency goes to ∞.

We present the $\mathcal{H}_2$ solution in an LFT framework rather than the more well known LQG

framework. We again assume the simplifying assumptions used in Section 2.3.4 The $\mathcal{H}_2$ design solution is obtained (at least conceptually) from the $\mathcal{H}_\infty$ design procedure by setting $\gamma = \infty$ and using the resulting central controller. It is interesting to compare the $\mathcal{H}_\infty$ solution, given above, and the $\mathcal{H}_2$ solution given below.

Define two Hamiltonians, H_2 and J_2, by,

$$H_2 = \begin{bmatrix} A & -B_2 B_2^T \\ -C_1^T C_1 & -A^T \end{bmatrix},$$

and

$$J_2 = \begin{bmatrix} A^T & -C_2^T C_2 \\ -B_1 B_1^T & -A \end{bmatrix}.$$

The sign definiteness of the off-diagonal blocks guarantees that $H_2 \in \mathrm{dom}(\mathrm{Ric})$, $J_2 \in \mathrm{dom}(\mathrm{Ric})$ and $X_2 = \mathrm{Ric}(H_2) \geq 0$ and $Y_2 = \mathrm{Ric}(J_2) \geq 0$. The following theorem gives the required result.

Theorem 4 *The unique $\mathcal{H}_2$ optimal controller is given by,*

$$K_2(s) = \left[\begin{array}{c|c} \hat{A}_2 & -L_2 \\ \hline F_2 & 0 \end{array}\right],$$

where,

$$\begin{aligned} F_2 &= -B_2^T X_2 \\ L_2 &= -Y_2 C_2^T \\ \hat{A}_2 &= A + B_2 F_2 + L_2 C_2. \end{aligned}$$

We commented above that the controller, K_2, is (conceptually) obtained by choosing $\gamma = \infty$ in the $\mathcal{H}_\infty$ design procedure. This does not mean that $\|G\|_\infty = \infty$; it simply means that we can make no *a priori* prediction about $\|G\|_\infty$ for this controller. K_2 minimizes $\|G\|_2$ and yields a finite $\|G\|_\infty$. As such, it is often useful for determining an

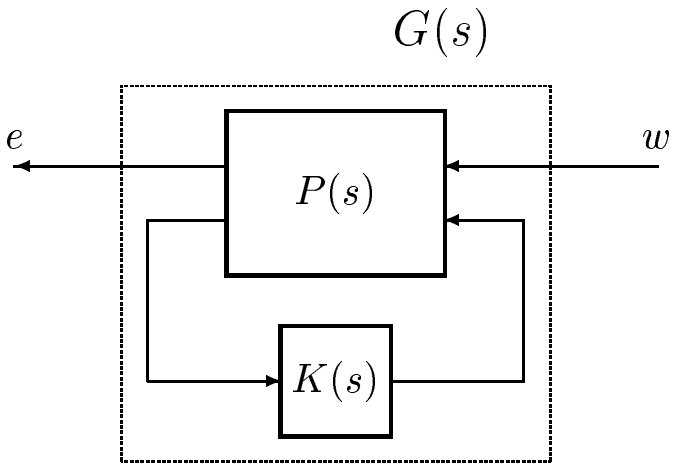


Figure 2.8: Closed loop system, $G(s)$, for performance analysis

initial choice of γ for the $\mathcal{H}_\infty$ design procedure. We will see later (Section 2.5) that it can also be used to initialize the D-K iteration procedure when an open-loop $\mathcal{H}_\infty$ design is poorly conditioned.

2.4 μ Analysis

2.4.1 Measures of Performance

Section 2.3 presented design performance objectives in terms of the norm ($\mathcal{H}_2$ or $\mathcal{H}_\infty$) of a closed loop system. We will now expand on this idea of performance. Consider the closed loop system illustrated in Figure 2.8. The interconnection structure, $P(s)$, is specified such that w represents unknown inputs; typically reference commands, disturbances and noise. The outputs, e, represent signals that we would like to be small. "Small" means in the sense of a selected norm. These signals might include actuator effort, and tracking error. As Figure 2.8 suggests, this analysis is typically applied after calculating a controller.

The inputs w are described only as members of a set. The performance question is then: *For all w in this set, are all possible outputs e also in some set?* The following set

descriptions are considered, where $\mathbf{B}$ again denotes the unit ball.

$$\text{Power :} \qquad \mathbf{BP} = \left\{ w \;\middle|\; \lim_{T\to\infty} \frac{1}{2T} \int_{-T}^{T} |\, w(t) \,|^2 \, dt \leq 1 \right\} \tag{2.12}$$

$$\text{Energy :} \qquad \mathbf{B}\mathcal{L}_2 = \left\{ w \;\middle|\; \|w\|_2^2 = \int_{-\infty}^{\infty} |\, w(t) \,|^2 \, dt \leq 1 \right\} \tag{2.13}$$

$$\text{Magnitude :} \quad \mathbf{B}\mathcal{L}_\infty = \left\{ w \;\middle|\; \|w\|_\infty = \text{ess} \sup_t |w(t)| \leq 1 \right\} \tag{2.14}$$

These norms are defined for scalar signals for clarity. The choice of w and e as the above sets defines the performance criterion. The performance can be considered as a test on the corresponding induced norm of the system. More formally,

Lemma 5 (Nominal Performance)

For all w in the input set, e is in the output set

if and only if $\quad \|G(s)\| \leq 1.$

Only certain combinations of input and output sets lead to meaningful induced norms. The $\mathcal{H}_\infty/\mu$ approach is based on the cases $w,\, e \in \mathbf{BP}$ and $w,\, e \in \mathbf{B}\mathcal{L}_2$. As we noted in Section 2.1.2, both of these cases lead to the following induced norm.

$$\|G(s)\|_\infty = \sup_\omega \sigma_{\max} \left[G(\jmath\omega) \right].$$

The choice of other input and output sets can lead to meaningful norms with engineering significance. For example $w,\, e \in \mathbf{B}\mathcal{L}_\infty$ is arguably a more suitable choice for some problems and leads to $\|G\|_1$ as a performance measure where

$$\|G\|_1 = \int_0^\infty |g(\tau)| \, d\tau.$$

and $g(\tau)$ is the convolution kernel (impulse response) of $G(s)$. For a discussion on the other possible selections of input and output sets, and the mathematical advantages of

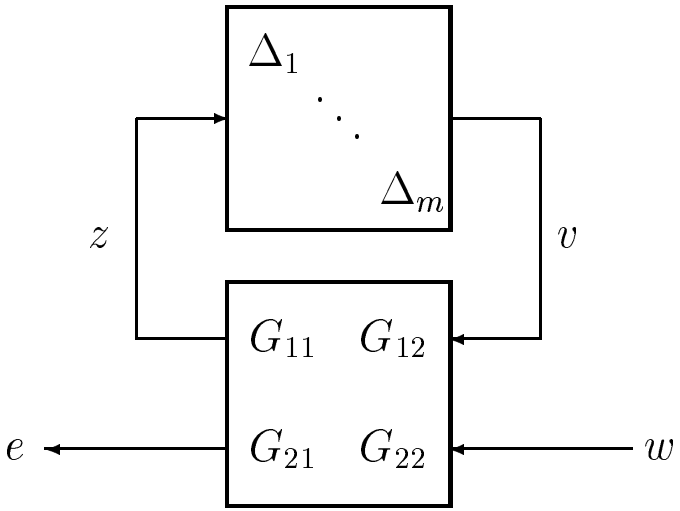


Figure 2.9: Perturbed closed loop system for stability analysis

the induced norms, the reader is referred to Doyle [2]. The major advantage of choosing **BP** or $\mathbf{B}\mathcal{L}_2$ is that the test for the performance can be considered in terms of the same norm as stability. This has significant advantages when we are considering performance and stability in the presence of perturbations, Δ.

2.4.2 Robust Stability and μ

Now we will consider the stability of a closed loop system under perturbations. In Figure 2.9, $G(s)$, is a perturbation model of a closed loop system. In the following robust stability and robust performance analyses we will assume that Δ is linear and time-invariant.

We will also assume that the interconnection structure $G(s)$ consists of stable transfer function matrices, where stability is taken to mean that the system has no poles in the closed right half plane. In practice this amounts to assuming that $G_{22}(s)$ (the nominal closed loop system model) is stable as the other elements, $G_{11}(s)$, $G_{12}(s)$, and $G_{21}(s)$, are weighting functions and can be chosen to be stable. The nominal closed loop system, $G_{22}(s)$, often arises from a standard design procedure ($\mathcal{H}_2$, $\mathcal{H}_\infty$, or loopshaping for example) and will be stable.

Consider the case where the model has only one full Δ block ($m = 1$ and $q = 0$ in Equation 2.9). This is often referred to as unstructured, and the well known result (refer to Zames [67] and Doyle and Stein [5]) is given in the following lemma.

Lemma 6 (Robust Stability, Unstructured)

$F_u(G(s), \Delta)$ is stable for all Δ, $\|\Delta\|_\infty \leq 1$,

if and only if $\|G_{11}(s)\|_\infty < 1$.

A generalization of the above is required in order to handle $F_u(G(s), \Delta)$ models with more than one full Δ block. The positive real valued function μ can be defined on a complex valued matrix M, by

$$\det(I - M\Delta) \neq 0 \quad \text{for all } \Delta \in \mathbf{B\Delta}, \qquad \text{if and only if} \quad \mu(M) < 1.$$

Note that μ scales linearly. In other words, for all $\alpha \in \mathcal{R}$,

$$\mu(\alpha M) = |\alpha|\mu(M).$$

In practice the test is normalized to one with the scaling being absorbed into the interconnection structure. An alternative definition of μ is the following.

$$\mu(M) \;=\; \begin{cases} 0 \text{ if no } \Delta \in \mathbf{\Delta} \text{ solves } \det(I + M\Delta) = 0 \\ \text{otherwise} \\ \left[\min\limits_{\Delta \in \mathbf{\Delta}} \left\{ \beta \;\middle|\; \exists \Delta, \|\Delta\| \leq \beta, \text{ such that } \det(I + M\Delta) = 0 \right\} \right]^{-1} \end{cases}$$

Note that μ is defined as the inverse of the size of the smallest destabilizing perturbation. This immediately gives the following lemma.

Lemma 7 (Robust Stability, Structured)

$F_u(G(s), \Delta)$ *stable for all* $\Delta \in \mathbf{B\Delta}$

if and only if $\quad \|\mu(G_{11}(s))\|_\infty < 1.$

where

$$\|\mu(G_{11}(s))\|_\infty = \sup_\omega \mu[G_{11}(\jmath\omega)].$$

The use of this notation masks the fact that μ is also a function of the perturbation structure, $\mathbf{\Delta}$. The above definition of μ applies to the more general block structure given in Section 2.2.4. We can even consider the some of the blocks to be real valued, rather than complex valued. The robust stability lemma is still valid; however the calculation of μ becomes significantly more difficult.

In applying the matrix definition of μ to a real-rational $G_{11}(s)$, it has been assumed that Δ is a complex constant at each frequency. This arises from the assumption that Δ is linear and time-invariant. Under this assumption we can examine the combination of system and perturbation independently at each frequency. The analysis then involves looking for the worst case frequency. If Δ is not time-invariant then the frequency by frequency analysis does not apply; Δ can be used to shift energy between frequencies and cause instability not predicted by the above analysis.

In practice this μ test is applied by selecting a frequency grid and at each frequency calculating $\mu(G_{11}(\jmath\omega))$. The choice of range and resolution for this grid is a matter of engineering judgement. If very lightly damped modes are present a fine grid may be required in the region of those modes.

2.4.3 Robust Performance

The obvious extension to the above is to consider performance in the presence of perturbations Δ. For e, $w \in \mathbf{BP}$ or $\mathbf{B}\mathcal{L}_2$ robust performance is a simple extension of robust stability.

Lemma 8 (Robust Performance)

$F_u(G(s), \Delta)$ *is stable and* $\|F_u(G(s), \Delta)\|_\infty \leq 1$ *for all* $\Delta \in \mathbf{B\Delta}$

if and only if $\quad \|\mu(G(s))\|_\infty < 1,$

where μ *is taken with respect to an augmented structure* $\widehat{\mathbf{\Delta}}$*,*

$$\widehat{\mathbf{\Delta}} = \left\{ \mathrm{diag}(\Delta, \hat{\Delta}) \;\middle|\; \Delta \in \mathbf{\Delta},\ \hat{\Delta} = \mathcal{C}^{\dim(w)\times\dim(e)} \right\}.$$

The additional perturbation block, $\hat{\Delta}$ can be thought of as a "performance block" appended to the Δ blocks used to model system uncertainty. This result is the major benefit of the choice of input and output signal norms; the norm test for performance is the same as that for stability. Robust performance is simply another μ test with one additional full block.

The frequency domain robustness approach, outlined above, assumes that the perturbations, Δ, are linear and time-invariant. This assumption is the most commonly applied. Section 2.4.6 will consider a robustness analysis from a state-space point of view. This form of analysis applies to norm bounded non-linear or time varying perturbations. We will first look more closely at the properties of μ, particularly as they relate to its calculation.

2.4.4 Properties of μ

The results presented here are due to Doyle [68]. Fan and Tits [69, 70] have done extensive work on algorithms for tightening the bounds on the calculation of μ. Packard [3] has also worked on improvement of the bounds and the extension of these results to the repeated block cases. The most comprehensive article on the complex singular value is that by Packard and Doyle [20]. More detail is contained in the technical report by Doyle *et al.* [71].

We will look at simple bounds on μ. The upper bound results are particularly important as they will form the basis of the design procedure provided in this software (*D-K* iteration).

Defining a block structure made up of one repeated scalar, ($\mathbf{\Delta} = \{\lambda I \mid \lambda \in \mathcal{C}\}$) makes the definition of μ the same as that of the spectral radius.

$$\mathbf{\Delta} = \left\{ \lambda I \;\middle|\; \lambda \in \mathcal{C} \right\} \quad \Rightarrow \quad \mu(M) = \rho(M).$$

For the other extreme consider a single full block ($\mathbf{\Delta} = \{\Delta \mid \Delta \in \mathcal{C}^{n\times n}\}$); the definition of μ is now the same as that for the maximum singular value,

$$\mathbf{\Delta} = \{\Delta \mid \Delta \in \mathcal{C}^{n\times n}\} \quad \Rightarrow \quad \mu(M) = \sigma_{\max}(M).$$

Observe that every possible block structure, $\mathbf{\Delta}$, contains $\{\lambda I \mid \lambda \in \mathcal{C}\}$ as a perturbation; and every possible block structure, $\mathbf{\Delta}$, is contained in $\mathcal{C}^{n\times n}$. These particular block structures are the boundary cases. This means that the resulting μ tests act as bounds on μ for any block structure, $\mathbf{\Delta}$. This gives the following bounds.

$$\rho(M) \quad \leq \quad \mu(M) \quad \leq \quad \sigma_{\max}(M).$$

The above bounds can be arbitrarily conservative but can be improved by using the following transformations. Define the set

$$\mathcal{D} = \left\{ \mathrm{diag}(D_1, \ldots, D_q, d_1 I_1, \ldots, d_m I_m,) \;\middle|\; D_j = D_j^* > 0, \right. \\ \left. \dim(I_i) = k_i, \;\; d_i \in \mathcal{R}, \;\; d_i > 0 \right\} \quad (2.15)$$

This is actually the set of invertible matrices that commute with all $\Delta \in \mathbf{\Delta}$. This allows us to say that for all $D \in \mathcal{D}$ and for all $\Delta \in \mathbf{\Delta}$,

$$D^{-1}\Delta D = \Delta.$$

Packard [3] shows that the restriction that d_i be positive real is without loss of generality. We can actually take one of these blocks to be one (or the identity).

Now define $\mathcal{Q}$ as the set of unitary matrices contained in $\mathbf{\Delta}$:

$$\mathcal{Q} = \left\{ Q \in \mathbf{\Delta} \;\middle|\; Q^*Q = I \right\}. \quad (2.16)$$

The sets $\mathcal{D}$ and $\mathcal{Q}$ can be used to tighten the bounds on μ in the following way (refer to Doyle [68]).

$$\max_{Q\in\mathcal{Q}} \rho(QM) \quad \leq \quad \mu(M) \quad \leq \quad \inf_{D\in\mathcal{D}} \sigma_{\max}(DMD^{-1}). \quad (2.17)$$

Actually, the lower bound is always equal to μ but the implied optimization has local maxima which are not global. For the upper bound Safonov and Doyle [72], have shown that finding the infimum is a convex problem and hence more easily solved. However the bound is equal to μ only in certain special cases. Here we use the infimum rather that the miminum because D may have a element which goes to zero as the maximum singular value decreases. So the limiting case (where an element of D is zero) is not a member of the set $\mathcal{D}$.

The cases where the upper bound is equal to μ are tabulated below.

	$q = 0$	$q = 1$	$q = 2$
$m = 0$		equal	less than or equal
$m = 1$	equal	equal	less than or equal
$m = 2$	equal	less than or equal	less than or equal
$m = 3$	equal	less than or equal	less than or equal
$m = 4$	less than or equal	less than or equal	less than or equal

Most practical applications of the μ theory involve models where $q = 0$. Here we see that we have equality with the upper bound for three or fewer blocks. Computational experience has yet to produce an example where the bound differs by more than 15 percent. In practically motivated problems the gap is usually much less.

2.4.5 The Main Loop Theorem

We will introduce a fundamental theorem in μ analysis: the main loop theorem. From the previous discussion you will see that there are several matrix properties that can be expressed as μ tests. The spectral radius and the maximum singular value are two such quantities. The main loop theorem gives a way of testing such properties for perturbed systems. The test is simply a larger μ problem. This is the theorem underlying the extension from robust stability to robust performance.

Consider a partitioned matrix,

$$M = \begin{bmatrix} M_{11} & M_{12} \\ M_{21} & M_{22} \end{bmatrix},$$

and two block structures, $\mathbf{\Delta}_1$ (compatible with M_{11}) and $\mathbf{\Delta}_2$ (compatible with M_{22}). There are two perturbed subsystems that we can study here: $F_u(M, \Delta_1)$, where Δ_1 is closed in a feedback loop around M_{11}; and $F_l(M, \Delta_2)$, where Δ_2 is closed in a feedback loop around M_{22}.

We have already seen that in the case of a dynamic system, the robust stability of $F_u(M, \Delta_1)$ is analyzed by checking that $\mu_1(M_{11}) < 1$. Here we have used μ_1 to indicate that we are considering it with respect to the block structure $\mathbf{\Delta}_1$. In the constant matrix case, we say that the LFT, $F_u(M, \Delta_1)$ is well posed for all $\Delta_1 \in \mathbf{B\Delta}_1$ if and only if $\mu_1(M_{11}) < 1$. This simply means that the inverse in the LFT equations is well defined for all $\Delta_1 \in \mathbf{B\Delta}_1$.

The well posedness discussion above applies equally well to $F_l(M, \Delta_2)$ and we will denote the μ test for M_{22} by $\mu_2(M_{22})$. However, instead of looking at μ_2 of M_{22}, we want to look at $\mu_2(F_u(M, \Delta_1))$. Note that $F_u(M, \Delta_1)$ has the same dimensions as M_{22} and in fact $F_u(M, \Delta_1) = M_{22}$ when $\Delta_1 = 0$. In otherwords, what happens when we apply the μ_2 test to the whole set of matrices generated by $F_u(M, \Delta_1)$.

To answer this question we need to introduce a larger block structure, denoted here simply by $\mathbf{\Delta}$. This is simply the diagonal combination of the previous two structures:

$$\mathbf{\Delta} = \mathrm{diag}(\mathbf{\Delta}_1, \mathbf{\Delta}_2).$$

Note that this has compatible dimensions with M itself and the associated μ test will be denoted by $\mu(M)$. Now we can answer the question about what happens to $\mu_2(F_u(M, \Delta_1))$ for all $\Delta_1 \in \mathbf{B\Delta}_1$.

Theorem 9 (Main Loop Theorem)

$$\mu(M) < 1 \qquad \textit{if and only if} \qquad \begin{cases} \mu_1(M_{11}) < 1 \\ \textit{and} \\ \max\limits_{\Delta_1 \in \mathbf{B\Delta}_1} \mu_2[F_u(M, \Delta_1)] < 1 \end{cases}$$

This theorem underlies the fact that robust performance is a simple extension of robust stability. It has a much more significant role in developing connections between the μ theory and other theoretical aspects of control. The example in the following section is an illustration of this point.

2.4.6 State-space Robustness Analysis Tests

We will look at some more advanced state-space approaches to the analysis of robust performance. Most users of the software will concentrate on the more common frequency domain analysis methods covered in Section 2.4.3. The analysis tests given here can be implemented with the Xμ functions and the more advanced user can use these to study the robustness of systems with respect to time-varying and nonlinear perturbations.

To illustrate this approach, consider the state-space LFT model of a system introduced in Section 2.2.4. A digital system is described by,

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) \\ y(k) &= Cx(k) + Du(k). \end{aligned}$$

This digital system has transfer function,

$$P(z) = F_u(P_{ss}, z^{-1}I),$$

where P_{ss} is the real valued matrix,

$$P_{ss} = \begin{bmatrix} A & B \\ C & D \end{bmatrix},$$

and the scalar $\times$ identity, $z^{-1}I$, has dimension nx, equal to the state dimension of $P(z)$. This is now in the form of an LFT model with a single scalar $\times$ identity element in the upper loop.

Define,

$$\mathbf{\Delta}_1 = \left\{ \delta I_{nx} \,\middle|\, \delta \in \mathcal{C} \right\},$$

and note that with this definition,

$$\mu_1(A) = \rho(A).$$

Therefore $\mu_1(A) < 1$ is equivalent to our system being stable. Furthermore, the maximum modulus theorem for a stable system tells us that,

$$\begin{aligned} \|P(z)\|_\infty &= \sup_{|z|\geq 1} \sigma_{\max}(P(z)) \\ &= \sup_{|z^{-1}|\leq 1} \sigma_{\max}(F_u(P_{ss}, z^{-1}I) \\ &= \sup_{\Delta_1 \in \mathbf{B}\boldsymbol{\Delta_1}} \mu_2(F_u(P_{ss}, \Delta_1), \end{aligned}$$

if $\boldsymbol{\Delta}_2$ is defined as a single full block of dimensions matching the input-output dimensions of $P(z)$. The main loop theorem (Theorem 9) immediately suggests the following result.

Lemma 10

$$\mu(P_{ss}) < 1 \qquad \text{if and only if} \qquad \begin{cases} P(z) \text{ is stable} \\ \text{and} \\ \|P(z)\|_\infty < 1. \end{cases}$$

Note that this tests whether or not the ∞ norm is less than one. It doesn't actually calculate the ∞ norm. To do this we have to set up a scaled system and search over the scaling with makes $\mu(P_{ss}) = 1$.

To apply this to a robust performance problem, consider the configuration shown in Figure 2.10. This is a state-space representation of a perturbed system. It would typically model an uncertain closed-loop system where the performance objective is $\|e\| \leq \|w\|$, for all $w \in \mathbf{B}\mathcal{L}_2$ and all $\Delta \in \mathbf{B}\boldsymbol{\Delta}$.

The real-valued matrix, G_{ss}, is,

$$G_{ss} = \begin{bmatrix} A & B_1 & B_2 \\ C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{bmatrix}.$$

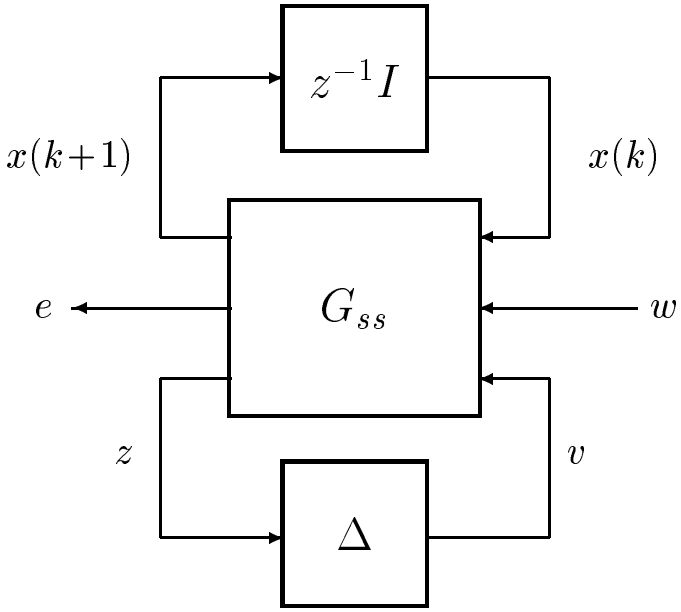


Figure 2.10: Perturbed system for state-space robustness tests

Note that the nominal system is given by,

$$G_{nom}(z) = F_u\left(\left[\begin{array}{cc} A & B_1 \\ C_1 & D_{11} \end{array}\right], z^{-1}I\right),$$

and the perturbed system is,

$$G(z) = F_u(F_l(G, \Delta), z^{-1}I).$$

We assume that Δ is an element of a unity norm bounded block structure, $\Delta \in \mathbf{B\Delta}$.

For the μ analysis we will define a block structure corresponding to G_{ss},

$$\mathbf{\Delta}_s = \left\{ \mathrm{diag}(\delta_1 I_{nx}, \Delta_2, \Delta) \;\middle|\; \delta_1 \in \mathcal{C},\ \Delta_2 \in \mathcal{C}^{\dim(w)\times\dim(e)},\ \Delta \in \mathbf{\Delta} \right\}.$$

Consider also a block structure corresponding to $F_u(G_{ss}, z^{-1}I)$,

$$\mathbf{\Delta}_p = \left\{ \mathrm{diag}(\Delta_2, \Delta) \;\middle|\; \Delta_2 \in \mathcal{C}^{\dim(w)\times\dim(e)},\ \Delta \in \mathbf{\Delta} \right\}.$$

This is identical to the $\mathbf{\Delta}_s$ structure except that the $\delta_1 I_{nx}$ block, corresponding to the state equation, is not present. The following theorem gives the equivalence between the standard frequency domain μ test and a state-space μ test for robust performance (first introduced by Doyle and Packard [23]. The notation μ_{Δ_s} will denote a μ test with respect to the structure $\mathbf{\Delta}_s$, and μ_{Δ_p} is a μ test with respect to the $\mathbf{\Delta}_p$ structure.

Theorem 11

The following conditions are equivalent.

i) $\mu_{\Delta_s}(G_{ss}) < 1$ *(state-space μ test);*

ii) $\rho(A) < 1$ *and* $\max_{\omega\in[0,2\pi]} \mu_{\Delta_p}(F_u(G_{ss}, \mathrm{e}^{\jmath\omega}I)) < 1$ *(frequency domain μ test);*

iii) There exists a constant $\beta \in [0,1]$ such that for each fixed $\Delta \in \mathbf{B}\mathbf{\Delta}$, $G(z)$ is stable and for zero initial state response, e satisfies $\|e\|_2 \leq \beta \|w\|_2$ (robust performance).

The frequency domain μ test is implemented by searching for the maximum value of μ over a user specified frequency grid. Theorem 11 shows that this is equivalent to a single, larger, μ test. There are subtle distinctions between the two tests. As we would expect, calculation of the larger μ test is more difficult. More importantly, the result does not scale. In the frequency domain test, if

$$\max_{\omega \in [0,2\pi]} \mu_{\Delta_p}[F_u(G_{ss}, \mathrm{e}^{j\omega} I_{nx})] = \beta,$$

where $\beta > 1$, then we are robust with respect to perturbations up to size $1/\beta$. In the state-space test, if $\mu_{\Delta_s}(G_{ss}) = \beta$, where $\beta > 1$, then we cannot draw any conclusions about the robust performance characteristics of the system. We must scale the inputs or outputs and repeat the calculation until the μ test gives a result less than one.

In practice we can only calculate upper and lower bounds for both of these μ tests. Although the state-space and frequency domain μ tests are equivalent, their upper bound tests have different meanings. We will see that this difference can be used to study the difference between linear time-invariant perturbations and linear time-varying (and some classes of non-linear) perturbations.

To clarify this issue, consider the D scales which correspond to $\mathbf{\Delta}_s$ and $\mathbf{\Delta}_p$;

$$\begin{aligned}
\mathcal{D}_s &= \left\{ \mathrm{diag}(D_1, d_2 I_2, D) \,\middle|\, D_1^T = D_1 > 0, \dim(D_1) = nx \times nx, \right. \\
&\qquad \left. d_2 > 0, \dim(I_2) = \dim(w) \times \dim(w),\ D \in \mathcal{D} \right\}, \\
\mathcal{D}_p &= \left\{ \mathrm{diag}(d_2 I_2, D) \,\middle|\, d_2 > 0, \dim(I_2) = \dim(w) \times \dim(w),\ D \in \mathcal{D} \right\}.
\end{aligned}$$

In the above $\mathcal{D}$ is the set of D-scales for the perturbation structure $\mathbf{\Delta}$, and, for notational simplicity, we have assumed that $\dim(w) = \dim(e)$. Now, the upper bound tests are:

i) State-space upper bound:

$$\inf_{D_s \in \mathcal{D}_s} \sigma_{\max}[D_s G_{ss} D_s^{-1}] < 1;$$

ii) Frequency domain, constant D, upper bound:

$$\inf_{D_p \in \mathcal{D}_p} \max_{\omega \in [0,2\pi]} \sigma_{\max}[D_p F_u(G_{ss}, \mathrm{e}^{\jmath\omega} I_{nx}) D_p^{-1}] < 1;$$

iii) Frequency domain upper bound:

$$\max_{\omega \in [0,2\pi]} \inf_{D_p \in \mathcal{D}_p} \sigma_{\max}[D_p F_u(G_{ss}, \mathrm{e}^{\jmath\omega} I_{nx}) D_p^{-1}] < 1.$$

In both the state-space and frequency domain, constant D, upper bound, a single D scale is selected to guarantee robust performance over all frequencies. These two tests (items *i*) and *ii*) above) are equivalent. In the frequency domain upper bound test (item *iii*)) a different D-scale is selected at each frequency.

The relationship between all of these tests is summarized by the following:

$\inf_{D_s \in \mathcal{D}_s} \sigma_{\max}[D_s G_{ss} D_s^{-1}] < 1$	State-space upper bound
$\Updownarrow$	
$\inf_{D_p \in \mathcal{D}_p} \max_{\omega \in [0,2\pi]} \sigma_{\max}[D_p F_u(G_{ss}, \mathrm{e}^{\jmath\omega} I_{nx}) D_p^{-1}] < 1$	Frequency domain, constant D, upper bound
$\Downarrow$	
$\max_{\omega \in [0,2\pi]} \inf_{D_p \in \mathcal{D}_p} \sigma_{\max}[D_p F_u(G_{ss}, \mathrm{e}^{\jmath\omega} I_{nx}) D_p^{-1}] < 1$	Frequency domain upper bound
$\Downarrow$	
$\max_{\omega \in [0,2\pi]} \mu_{\Delta_p}[F_u(G_{ss}, \mathrm{e}^{\jmath\omega} I_{nx})] < 1$	Frequency domain μ test
$\Updownarrow$	
$\mu_{\Delta_s}[G_{ss}] < 1$	State-space μ test

In the two cases where there are one way implications, there are real gaps. We have already seen that there is a gap between the frequency domain μ test and its upper bound for four or more full blocks. This is a computational issue.

The gap between the state-space (or constant D) upper bound and the frequency domain upper bound is more significant. In the state-space upper bound, a single D scale is selected. This gives robust performance for all Δ satisfying, $\|v\| \leq \|z\|$ for all $e \in \mathcal{L}_2$. This can be satisfied for linear time-varying perturbations or non-linear cone bounded perturbations. The formal result is given in the following theorem (given in Packard and Doyle [20]).

Theorem 12

If there exists $D_s \in \mathcal{D}_s$ such that

$$\sigma_{\max}[D_s G_{ss} D_s^{-1}] = \beta < 1,$$

then there exists constants, $c_1 \geq c_2 > 0$ such that for all perturbation sequences, $\{\Delta(k)\}_{k=0}^{\infty}$ with $\Delta(k) \in \mathbf{\Delta}$, $\sigma_{\max}[\Delta(k)] < 1/\beta$, the time varying uncertain system,

$$\begin{bmatrix} x(k+1) \\ e(k) \end{bmatrix} = F_l(G_{ss}, \Delta(k)) \begin{bmatrix} x(k) \\ w(k) \end{bmatrix},$$

is zero-input, exponentially stable, and furthermore if $\{w(k)\}_{k=0}^{\infty} \in l_2$, then

$$c_2(1-\beta^2)\|x\|_2^2 + \|e\|_2^2 \leq \beta^2 \|w\|_2^2 + c_1\|x(0)\|^2.$$

In particular,

$$\|e\|_2^2 \leq \beta^2 \|w\|_2^2 + c_1\|x(0)\|^2.$$

The user now has a choice of robust performance tests to apply. The most appropriate depends on the assumed nature of the perturbations. If the state-space upper bound test is used, the class of allowable perturbations is now very much larger and includes perturbations with arbitrarily fast time variation. If the actual uncertainty were best modeled by a linear time-invariant perturbation then the state-space μ test could be conservative. The frequency domain upper bound is probably the most commonly used test. Even though the uncertainties in a true physical system will not be linear, this assumption gives suitable analysis results in a wide range of practical examples.

2.4.7 Analysis with both Real and Complex Perturbations

The above results only apply to the case where Δ is considered as a constant complex valued matrix at each frequency. In many engineering applications restricting certain of the Δ blocks to be real valued may result in a less conservative model. Analysis with such restrictions is referred to as the "mixed" μ problem.

For example, consider the LFT form of the engine combustion model developed in Section 2.2.4 (Equation 2.10). The block structure contains both real and complex perturbations. A closed-loop model will also include these perturbations and the robust stability and robust performance analyses will involve calculation of μ with respect to both real and complex valued perturbations. We could simply assume that all perturbations were complex; this would certainly cover the situation. However, such an assumption may be too conservative to be useful. Calculation of mixed μ will give a more accurate result in this case.

Efficient computation of μ in the mixed case is discussed by Doyle, Fan, Young, Dahleh and others [73, 74, 75, 76]. Accurate mixed μ analysis software will be available in the near future. Unlike the complex μ case, this will not directly lead to a compatible synthesis procedure. Significantly more work is required in this direction.

2.5 μ Synthesis and D-K Iteration

2.5.1 μ-Synthesis

We now look at the problem of designing a controller to achieve a performance specification for all plants, $P(s)$, in a set of plants, $\mathcal{P}$. The previous sections have dealt with the questions of performance and robust stability in depth and the same framework is considered for the synthesis problem. Figure 2.11 illustrates the generic synthesis interconnection structure.

The lower half of this figure is the same as that for the $\mathcal{H}_\infty$ and $\mathcal{H}_2$ design procedure. The controller measurements are y, and the controller actuation inputs to the system are u. The configuration differs from the standard $\mathcal{H}_\infty$ or $\mathcal{H}_2$ case in that $F_u(P(s), \Delta)$ (rather than the nominal plant, $P_{22}(s)$) is used as the design interconnection structure.

The problem is to find $K(s)$ such that for all $\Delta \in \mathbf{B\Delta}$, $K(s)$ stabilizes $F_u(P(s), \Delta)$ and

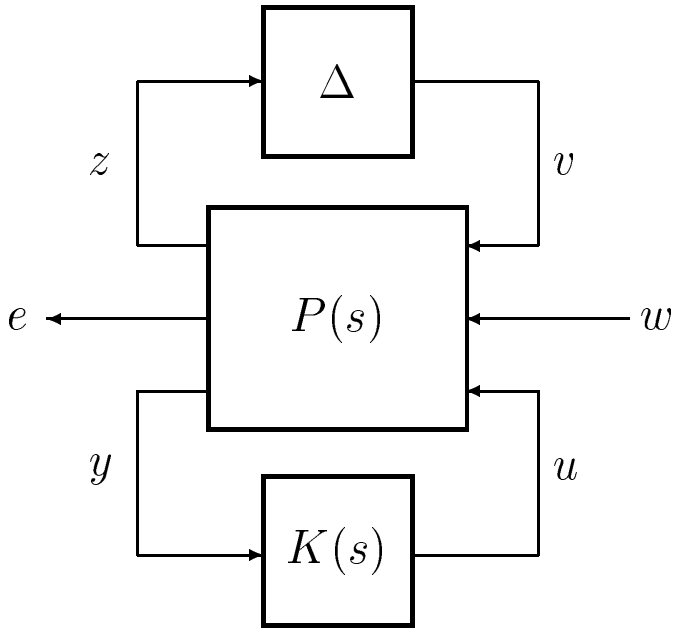


Figure 2.11: The generic interconnection structure for synthesis

$\|F_u(F_l(P(s), K(s)), \Delta)\|_\infty \leq 1$. This is equivalent to $K(s)$ satisfying $\mu[F_l(P(s), K(s))] < 1$. In other words, the closed loop system satisfies the robust performance specification.

Unfortunately this problem has not yet been solved, except is a few special cases. The current approach to this problem, known as *D-K* iteration, involves the iterative application of the $\mathcal{H}_\infty$ design technique and the upper bound μ calculation. We will give a brief conceptual overview here and give more algorithmic details in Section 2.5.2.

Consider applying $\mathcal{H}_\infty$ synthesis to the full $P(s)$ interconnection structure for this problem. Suppose that this gives a controller, $K(s)$ such that $K(s)$ stabilizes $P(s)$ and

$$\|F_l(P(s), K(s))\|_\infty \leq 1.$$

Recall that this is an upper bound for the μ problem of interest, implying that,

$$\mu[F_l(P(s), K(s)] \leq 1,$$

as required. However the upper bound may be conservative, meaning that in order to guarantee that $\mu[F_l(P(s), K(s)] \leq 1$, we have had to back off on the performance and/or the stability margins.

With the appropriate choice of D scalings the upper bound will be much closer to μ. In otherwords there exists D such that, $\|DF_l(P(s), K(s))D\|_\infty$ is a close upper bound to $\mu[F_l(P(s), K(s)]$. The μ-synthesis problem can be replaced with the following approximation (based on the upper bound):

$$\inf_{\substack{D\in\mathcal{D} \\ K(s)\ stabilizing}} \|DF_l(P(s), K(s))D^{-1}\|_\infty. \tag{2.18}$$

The reader is referred to Doyle [1] for details of this problem.

If this is considered as an optimization of two variables, D and $K(s)$, the problem is convex in each of the variables separately, but not jointly convex. Doyle [2] gives an example where this method reaches a local nonglobal minimum.

D-K iteration involves iterating between using $D \in \mathcal{D}$ and $K(s)$ to solve Equation 2.18. There are several practical issues to be addressed in doing this and we discuss those in the next section.

2.5.2 The D-K Iteration Algorithm

The objective is to design a controller which minimizes the upper bound to μ for the closed loop system;

$$\inf_{\substack{D\in\mathcal{D} \\ K(s)\ stabilizing}} \|DF_l(P(s), K(s))D^{-1}\|_\infty.$$

The major problem in doing this is that the D-scale that results from the μ calculation is in the form of frequency by frequency data and the D-scale required above must be a

dynamic system. This requires fitting an approximation to the upper bound D-scale in the iteration. We will now look at this issue more closely.

The D-K iteration procedure is illustrated schematically in Figure 2.12. It can be summarized as follows:

i) Initialize procedure with $K_0(s)$: $\mathcal{H}_\infty$ (or other) controller for $P(s)$.

ii) Calculate resulting closed loop: $F_l(P(s), K(s))$.

iii) Calculate D scales for μ upper bound:

$$\inf_{D(\omega)\in\mathcal{D}} \sigma_{\max}[D(\omega)F_l(P(s), K(s))D(\omega)^{-1}].$$

iv) Approximate frequency data, $D(\omega)$, by $\hat{D}(s) \in \mathcal{RH}_\infty$, with $\hat{D}(\jmath\omega) \approx D(\omega)$.

v) Design $\mathcal{H}_\infty$ controller for $\hat{D}(s)P(s)\hat{D}^{-1}(s)$.

vi) Go to step *ii*).

We have used the notation $D(\omega)$ to emphasize that the D scale arises from frequency by frequency μ analyses of $G(\jmath\omega) = F_l(P(\jmath\omega), K(\jmath\omega))$ and is therefore a function of ω. Note that it is NOT the frequency response of some transfer function and therefore we do NOT use the notation $D(\jmath\omega)$.

The μ analysis of the closed loop system is unaffected by the D-scales. However the $\mathcal{H}_\infty$ design problem is strongly affected by scaling. The procedure aims at finding at D such that the upper bound for the closed loop system is a close approximation to μ for the closed loop system. There are several details about this procedure that will now be clarified.

At each frequency, a scaling matrix, $D(\omega)$, can be found such that $\sigma_{\max}(D(\omega)G(\jmath\omega)D(\omega)^{-1})$ is a close upper bound to $\mu(G(\jmath\omega))$ (Figure 2.12c). The D scale is block diagonal and the block corresponding to the e and w signals can be chosen to be the identity. The part of D corresponding to the z signal commutes with Δ and cancels out the part of D^{-1} corresponding to the v signal. To illustrate this, consider the D scale that might result from a block structure with only m full blocks. At each

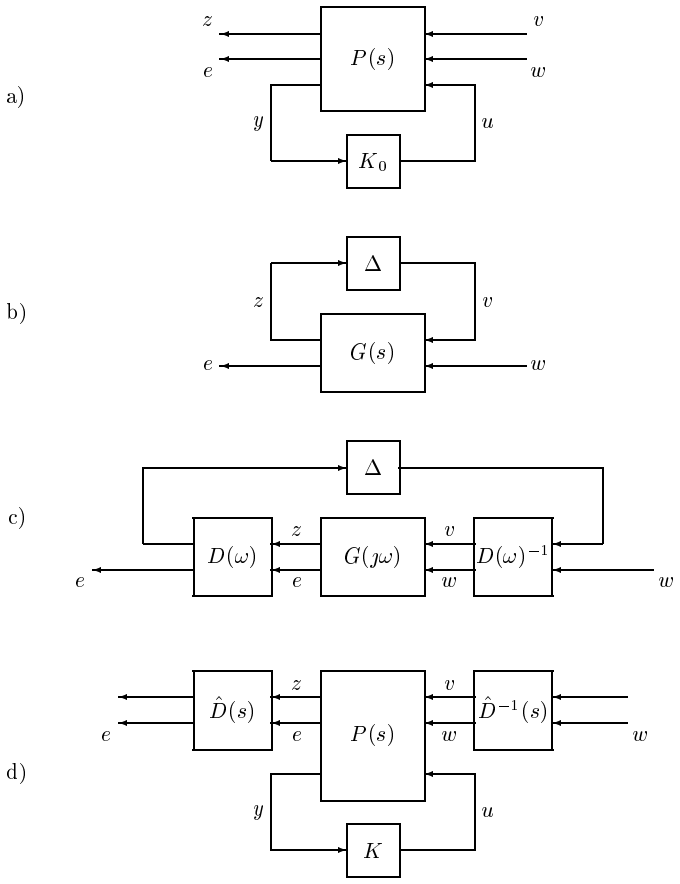


Figure 2.12: *D-K* iteration procedure: a) Design $\mathcal{H}_\infty$ (or other) controller: $K_0(s)$ [step *i*)]. b) Closed loop perturbed system for μ analysis [step *ii*)]. c) Frequency by frequency upper bound $D(\omega)$ scale approximation to μ analysis [step *iii*)]. d) Scaling of $\mathcal{H}_\infty$ design problem by $\hat{D}(s)$ where $\hat{D}(j\omega) \approx D(\omega)$ [steps *iv*) & *v*)].

frequency we would have,

$$D = \begin{bmatrix} d_1 I_1 & & & \\ & \ddots & & \\ & & d_m I_m & \\ & & & I_e \end{bmatrix},$$

where the identity I_e is of dimensions $\dim(e) \times \dim(e)$.

The calculation of a new $\mathcal{H}_\infty$ controller requires a state-space realization of $D(\omega)$. For each d_i in $D(\omega)$ we must fit a transfer function approximation, which we will denote by $\hat{d}_i(s)$. This is denoted by $\hat{D}(s)$ in the above discussion. The observant reader will notice that, as defined here, $\hat{D}(s)$ is not of the correct input dimension to multiply $P(s)$. We must append another identity of dimension equal to the dimension of the signal y. The final result is,

$$\hat{D}(s) = \begin{bmatrix} \hat{d}_1(s) I_1 & & & & \\ & \ddots & & & \\ & & \hat{d}_m(s) I_m & & \\ & & & I_e & \\ & & & & I_y \end{bmatrix}$$

and

$$\hat{D}^{-1}(s) = \begin{bmatrix} \hat{d}_1^{-1}(s) I_1 & & & & \\ & \ddots & & & \\ & & \hat{d}_m^{-1}(s) I_m & & \\ & & & I_w & \\ & & & & I_u \end{bmatrix}.$$

Throughout the theoretical discussion we have assumed that the perturbation blocks, Δ_i, were square. The software handles the non-square case. This makes a difference to $\hat{D}(s)$ and $\hat{D}^{-1}(s)$. The identity blocks (I_m, etc.) shown above will be of different sizes for $\hat{D}(s)$ and $\hat{D}^{-1}(s)$ if the corresponding Δ_i perturbation is non-square. Similarly, the I_w and I_u identities in $\hat{D}^{-1}(s)$ are not necessarily the same size as I_e and I_y in $\hat{D}(s)$.

Several aspects of this procedure are worth noting. For the μ analysis and D scale calculation, a frequency grid must be chosen. The range and resolution of this grid is a matter of engineering judgement. The μ analysis can require a fine grid in the vicinity of the lightly damped modes. The order of the initial controller, $K_0(s)$, is the same as the interconnection structure, $G(s)$. The order of $K(s)$ is equal to the sum of the orders of $G(s)$, $\hat{D}(s)$ and $\hat{D}^{-1}(s)$. This leads to a trade-off between the accuracy of the fit between D and $\hat{D}(s)$ and the order of the resulting controller, $K(s)$.

Another aspect of this to consider is that as the iteration approaches the optimal μ value, the resulting controllers often have more and more response at high frequencies. This may not show up in the μ calculation, the D scale fitting, or a frequency response of $K(s)$, because the dynamics are occuring outside of the user specified frequency grid. However these dynamics affect the next $\mathcal{H}_\infty$ design step and may even lead to numerical instability.

The above discussion used an $\mathcal{H}_\infty$ controller to initialize the iteration. Actually any stabilizing controller can be used. In high order, lightly damped, interconnection structures, the $\mathcal{H}_\infty$ design of $K_0(s)$ may be badly conditioned. In such a case the software may fail to generate a controller, or may give controller which doesn't stabilize the system. A different controller (the $\mathcal{H}_2$ controller is often a good choice) can be used to get a stable closed loop system, and thereby obtain D scales. Application of these D scales (provided that they do not add significantly many extra states) often results in a better conditioned $\mathcal{H}_\infty$ design problem and the iteration can proceed.

The robust performance difference between the $\mathcal{H}_\infty$ controller, $K_0(s)$, and $K(s)$, can be dramatic even after a single D-K iteration. The $\mathcal{H}_\infty$ problem is sensitive to the relative scalings between v and w (and z and e). The D scale provides the significantly better choice of relative scalings for closed loop robust performance. Even the application of a constant D scale can have dramatic benefits.

2.6 Model Reduction

High order interconnection structures will result in high order controllers. Often a controller of significantly lower order will perform almost as well. Approximating a state-space system by one of lower order is referred to as model reduction. There are several techniques available for this purpose in Xμ and the background to these techniques is discussed here.

2.6.1 Truncation and Residualization

The simplest form of model reduction is state truncation. Consider a system, $P(s)$, with a partitioned state matrix,

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

Truncating the states associated with A_{22} results in,

$$P_{trun}(s) = \left[\begin{array}{c|c} A_{11} & B_1 \\ \hline C_1 & D \end{array}\right].$$

In any practical application we would order the states so that those truncated do not significantly affect the system response. For example, to truncate high frequency modes, A is transformed to be diagonal (or with 2×2 blocks on the diagonal) and the eigenvalues are ordered in increasing magnitude. This results in $A_{21} = 0$ and A_{22} corresponds to the high frequency modes.

Truncation also affects the zero frequency response of the system. Residualization involves truncating the system and adding a matrix to the D matrix so that the zero frequency gain is unchanged. This typically gives a closer approximation to the original system at low frequency. If the original system rolls off with frequency, the low order residualized approximation will usually not share this characteristic. Using the above $P(s)$, the result is,

$$P_{resid}(s) = \left[\begin{array}{c|c} A_{11} - A_{12}A_{22}^{-1}A_{21} & B_1 - A_{12}A_{22}^{-1}B_2 \\ \hline C_1 - C_2A_{22}^{-1}A_{21} & D - C_2A_{22}^{-1}B_2 \end{array}\right].$$

2.6.2 Balanced Truncation

Consider a stable state-space system,

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

The controllability grammian, Y is defined as,

$$Y = \int_0^\infty \mathrm{e}^{At} B B^T \mathrm{e}^{A^T t} dt,$$

and the observability grammian, X, is defined as

$$X = \int_0^\infty \mathrm{e}^{A^T t} C^T C \mathrm{e}^{At} dt.$$

The grammians, X and Y, satisfy the Lyapunov equations,

$$\begin{aligned} AY + YA^T + BB^T &= 0 \\ A^T X + XA + C^T C &= 0, \end{aligned}$$

and this is typically how they are calculated. We can also see from the definitions that $X \geq 0$ and $Y \geq 0$. Actually $Y > 0$ if and only if (A,B) is controllable and $X > 0$ if and only if (C, A) is observable.

Now consider the effect of a state transformation on these grammians. Define a new state, $\hat{x}$, by $\hat{x} = Tx$, where T is invertible, to give

$$\begin{aligned} P(s) &= \left[\begin{array}{c|c} \hat{A} & \hat{B} \\ \hline \hat{C} & D \end{array}\right] \\ &= \left[\begin{array}{c|c} TAT^{-1} & TB \\ \hline CT^{-1} & D \end{array}\right]. \end{aligned}$$

The new grammians are $\hat{Y} = TYT^T$ and $\hat{X} = T^{-T} X T^{-1}$. The product of the grammians, $\hat{Y}\hat{X}$ is therefore given by,

$$\hat{Y}\hat{X} = TYXT^{-1}.$$

This illustrates that the eigenvalues of the product of the grammians is invariant under state similarity transformation.

We will now look at a particular choice of transformation. For a minimal realization, we can always find a transformation that gives,

$$\hat{Y} = TYT^T = \Sigma,$$

and

$$\hat{X} = T^{-T}XT{-}1 = \Sigma,$$

where $\Sigma = \text{diag}(\sigma_1, \ldots, \sigma_n)$ and $\sigma_i \geq 0$, $i = 1, \ldots, n$. This realization, where the grammians are equal, is called a balanced realization. Each mode of the balanced system can be thought of as equally controllable and observable. Balanced realization was first introduced by Moore [77].

The σ_i are known as the Hankel singular values of the system and are ordered such that σ_1 is the largest and σ_n is the smallest. Because the eigenvalues of the product of the grammians are invariant with respect to similarity transformations, the Hankel singular values are system invariants. We will denote the Hankel norm of a system as $\|P(s)\|_H$ and this is given by,

$$\|P(s)\|_H = \sigma_1.$$

The input-output interpretation of the Hankel norm is the following,

$$\|P\|_H = \sup_{u(t) \in \mathcal{L}_2(-\infty, 0)} \frac{\left\| y(t)\big|_{(0,\infty)} \right\|_2}{\|u(t)\|_2}.$$

The notation, $y(t)\big|_{(0,\infty)}$, denotes the system output, considered only over the time interval zero to ∞. So we are looking at the system output, from time zero to ∞, in response to input signals from $-\infty$ to zero. The Hankel norm is the maximum gain from past inputs to future outputs. Each signal, $u(t) \in \mathcal{L}_2(-\infty, 0)$ drives the system state to a particular location in the state-space, and the output (considered over $(0, \infty)$) is the corresponding transient decay from that state.

Balanced truncation involves obtaining a balanced realization of $P(s)$ and then truncating the states corresponding to the smallest Hankel singular values. Enns [78]

and Glover [79] independently obtained the following bound on the error induced by balanced truncation.

Theorem 13

Given a stable, rational, $P(s)$, and $P_{bal}(s)$, the balanced truncation of order $k < n$. Then,

$$\|P(s) - P_{bal}(s)\|_\infty \leq 2 \sum_{i=k+1}^{n} \sigma_i$$

and

$$\|P(s) - P_{bal}(s)\|_H \leq 2 \sum_{i=k+1}^{n} \sigma_i.$$

Unobservable or uncontrollable modes have a corresponding Hankel singular value of zero and we can see immediately from the above that their truncation does not affect the ∞-norm of the system.

2.6.3 Hankel Norm Approximation

We can also consider the problem of finding the kth order controller which gives the closest fit (in terms of the Hankel norm) to the original system. The results given here are due to Glover [79]. The first thing to consider is a lower bound on the error, which is specified in the following lemma.

Lemma 14

Given a stable, rational $P(s)$, and a kth order approximation, $P_k(s)$. Then

$$\sigma_{k+1} \leq \|P(s) - P_k(s)\|_\infty.$$

This tells us how well we can expect to do in terms of the ∞ norm. Actually, there exists a $P_k(s)$ which achieves this bound. The only problem is that it can have unstable (or anti-causal) parts to it.

Consider the problem of finding the stable, order k realization which minimizes the Hankel norm of the error. Define, $P_{hankel}(s)$ as the minimizing system. Then we have,

$$\sigma_{k+1} \quad \leq \quad \|P(s) - P_{hankel}(s)\|_H \quad = \quad \inf_{P_k(s) \ stable} \|P(s) - P_k(s)\|_H.$$

This system also satisfies ∞-norm bounds on the error, as illustrated in the following theorem.

Theorem 15

Given a stable, rational, $P(s)$, and the optimal kth order Hankel norm approximation, $P_{hankel}(s)$. Then

$$\|P(s) - P_{hankel}(s)\|_\infty \leq 2 \sum_{i=k+1}^{n} \sigma_i.$$

Furthermore, there exists a constant matrix, D_0, such that

$$\|P(s) - (P_{hankel}(s) + D_0)\|_\infty \leq \sum_{i=k+1}^{n} \sigma_i.$$

Careful examination of the previous section will indicate that the Hankel norm of a system is independent of the D term. The optimal Hankel norm approximation given above, $P_{hankel}(s)$, is considered to have a zero D term. It has the same error bounds as the balanced truncation. Theorem 15 states that we can find a D matrix to add to $P_{hankel}(s)$ to cut this bound in half.

The most common use of balanced truncations and Hankel norm approximations is to reduce the order of a controller. Note that this will give a small ∞-norm error with respect to the open-loop controller. It does not say anything about the preservation of closed loop properties. These should always be checked after performing a controller order reduction.

Chapter 3

Functional Description of Xμ

3.1 Introduction

This chapter describes the Xμ functions in the context of their intended usage. Chapter 2 provides the reader with an idea of the theoretical basis behind the various analysis and design calculations. Here we outline the software functions available for doing those calculations.

Robust control design uses a subset of data objects provided within Xmath. We discuss the details of the most heavily used objects: DYNAMIC SYSTEMs and PDMs. This coverage overlaps that given in the Xmath Basics manual; only the emphasis is different. There are several subtleties which arise when using these data objects in a robust control context. These issues are discussed in Section 3.2.

3.2 Data Objects

Xmath provides a wide range of data objects. There are several which are of primary interest in control design: matrices, PDMs and DYNAMIC SYSTEM. The transfer function object is useful for specifying systems although all calculations will be done with state space DYNAMIC SYSTEMs. The control uses of these objects is reviewed in this section.

3.2.1 Dynamic Systems

Xmath has a dynamic system data object which specifies a dynamic system in terms of A, B, C and D matrices. The dynamic equations of the system are,

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

in the continuous case, and

$$\begin{aligned} x(kT+T) &= Ax(kT) + Bu(kT), \\ y(kT) &= Cx(kT) + Du(kT), \end{aligned}$$

in the discrete time case. The discrete time sample period, T, is stored as part of the data object. The user can label the system inputs, $u(t)$, outputs, $y(t)$, and states, $x(t)$.

Also residing within the Xmath state-space object is the initial state, $x(0)$. This is used primarily for time response calculations. It is debatable whether or not the initial state is an intrinsic attribute of the system as one frequently changes it for simulation. It has the advantage of reducing the time response calculation to a simple multiplication and it can easily be changed without accessing all of the other system variables.

The Xmath core functions `system` and `abcd` are used to create state-space systems. The `system` function is also used to specify the initial state (and, as discussed in the next section, any other system attributes). The following example puts together a simple two-state system.

```
# Specify the A, B, C & D matrices
a = [-.15,.5;-.5,-.15]
b = [.2,4;-.4,0]
c = [5,5]
d = [.1,-.1]
sys = system(a,b,c,d)
[a,b,c,d] = abcd(sys)
```

Simple systems are easily generated as transfer functions. To achieve this we simply define the numerator and denominator polynomials and then divide one by the other.

As above, these polynomials can be specified by their roots or their coefficients. Note that we can specify the variable, and for continuous systems we use "s". To create a discrete system "z" is used.

```
# Generate the system from the numerator and denominator
# coefficients.
numerator = makepoly([-.1,19.97,-7.02725],"s");
denominator = makepoly([1,0.3,0.2725],"s");
# We can also do this by specifying the roots of each
# polynomial
numerator = -0.1*polynomial([199.3475;0.3525],"s");
denominator = polynomial( ...
         [-.15+0.5*jay;-.15-0.5*jay],"s");
# Note that multiplying the by -0.1 does the correct
# thing to the polynomial above.  The final system is
# obtained with the command:
sys1 = numerator/denominator
```

Labeling and Comments

Xmath allows the user to label all the inputs, outputs and states in a state-space system. Any Xmath variable can also have a comment string associated with it.

Keywords for the `system` function are used to label the system. In the following example, the two-input, single-output system generated above is used as the starting point. Comments can be attached to any variable in the workspace (or the workspace itself) with the `comment` command.

```
# Set up vectors of strings for the labels
inputs = ["disturbance";"actuator"]
outputs = ["measurement"]
states = ["x1";"x2"]
# and attach them to sys
sys = system(sys,inputNames = inputs,...
     outputNames = outputs,...
     stateNames = states)
# We can also attach a comment to sys
comment sys "Example system for the manual"
```

Because the dynamic system is a built-in data object, the label information will be displayed by simply typing the object name (`sys` in the above) or appending a question mark to the statement. The core function `commentof` is used to read the comment attached to a specified variable.

3.2.2 PDMs

A PDM can be thought of as a three-dimensional matrix. The typical control uses are time or frequency responses, where the domain variable (the third dimension) is time or frequency. The frequency response of a multiple-input, multiple-output (MIMO) system is a complex valued matrix at each frequency. A time response of a multiple-output system is a real valued vector at each time step.

As we would expect, there is a core function for constructing PDMs from the raw data: `pdm`. The data matrices can be either bottom or right concatenated. If there is an ambiguity, Xmath assumes that the data was bottom concatenated. Keywords can be used to specify the row and column dimensions of the data matrices.

To extract the data again, the function `makematrix` returns the matrix data in right concatenated form. The domain of a PDM is obtained with the function `domain`.

```
# Make a pdm with 2 values in the domain
mat1 = random(2,2)
mat2 = -1*random(2,2)
dom = [1;1+pi]
pdm1 = pdm([mat1;mat2],dom)
# Extract back the data and the domain
dom = domain(pdm1)
data = makematrix(pdm1)
```

The PDM data object also contains row, column and domain labels. These can be appended with the `pdm` command in exactly the same manner as the DYNAMIC SYSTEM labels illustrated above. Refer to the Xmath Basics manual for graphical illustration and further details about PDMs.

Appending and Merging Data

Time functions, for creating simulation inputs for example, can be created by combining PDMs. Xmath has two such core functions: `concatseg` and `insertSeg` The `concatseg` appends the data of one PDM to another. The domain is recalculated such that it is always regular. The user can specify a new beginning value for the domain and the spacing is determined by a series of prioritized rules. `insertSeg` inserts the data from one PDM into another. It can also "insert" before the beginning, or after the end, of the first PDM and the resulting gap is filled with zeros. Again the domain of the result is recalculated from a user specified beginning and is always regular. Both of these functions are useful in creating time domain waveforms.

In the typical robust control cases where we want to merge PDM data, the result will not have a regular domain. One example is merging experimental transfer function estimates from multiple experiments, prior to performing a least squares transfer function fit or error analysis. For this purpose, Xμ provides a `mergeseg` function. Keywords allow the user to specify whether to sort in ascending or descending order or even whether to sort at all. The following example illustrates the a typical use.

```
# pdm1 and pdm2 have identical row and column
# dimensions and at least some non matching
# independent variables
pdm3 = mergeseg(pdm1,pdm2,{ascending,!duplicates})
```

As an aside, note that the `sort` function in Xmath sorts each column of each matrix in a PDM, rather than sorting the domain.

Extracting Data by Independent Variable/Domain

It is often useful to be able to extract a segment of matrix data from a PDM. The function `extractseg` performs this function. The user has the option of resetting the domain to a new starting value.

```
# sys1g is a freq.  response from 0.1 to 100 rad/sec
# select data from 1 to 10 rad/sec as sys1g1
sys1g1 = extractseg(sys1g,1,10)
```

Data can also be extracted by independent variable index number, by providing a scalar argument to the PDM. In the following example the fifth through tenth and the twentieth independent variables are specified for the smaller PDM, `pdm2`.

```
size(pdm1)
ans (a row vector) =    1   1   100
pdm2 = pdm1([5:10,20])
size(pdm2)
ans (a row vector) =    1   1   7
```

Indexing and Finding Data

The core function `find` returns the indices of the elements of a matrix which meet a user defined criterion. By using a Boolean function of a matrix as the argument to `find`, the an index to the entries satisfying the Boolean function can be generated. This function works equally well for for PDMs.

This is illustrated in the following example.

```
# sys3g is a MIMO frequency response.
# We want to find the frequencies where the
# SVD of this response exceeds 1.
idx = find(max(svd(sys3g))>1)
domain(sys3g(idx(:,1)))
```

There are several points to observe here. Both `svd` and `max` operate on PDMs as well as matrices. Refer to Section 3.2.4 for additional details. The variable `idx` is a data object known as an indexlist. In this case the indexlist has three columns: the domain index, row index and column index. By selecting the domain index the appropriate sub-PDM can be selected.

The `indexlist` command creates the indexlist data object from a given three column matrix. The indexlist data type can also be used for assigning data to part of a PDM. This is illustrated in the following example.

```
# Assign the row 2, column 1 element of the 4th domain
```

```
# index of the pdm the value 100.
idxlst = indexlist([4,1,2])
pdm1(idxlst) = 100
```

Operations on the Independent Variables/Domain

The domain of a PDM is readily changed via the `pdm` command. The following example illustrates a common application; changing a frequency domain in Hertz to radians/second.

```
# The following scales the domain of a pdm,
# pdm1, by a factor of 2*pi
newpdm = pdm(pdm1,2*pi*domain(pdm1))
```

Xmath provides a general purpose `check` which can be used to check whether two PDMs have the same domain. For more information on this function, refer to Section 3.3.1. The following example illustrates this application.

```
# Check whether or not pdm1 & pdm2 have the
# same domains.
stat = check(pdm1,pdm2,{samedomain})
# stat = 1 if the domains are the same
```

3.2.3 Subblocks: selecting input & outputs

Because the DYNAMIC SYSTEM is an Xmath data object, standard matrix referencing can be used to select subsets of the inputs and outputs. This is illustrated in the following.

```
# Select inputs 1, 3, 4 & 5 and outputs 2 & 7 from the
# system:  bigsys.
subsys = bigsys([1,3:5],[2,7])
```

The same format has exactly the same function for PDMs.

```
# Select columns 1, 3, 4 & 5 and rows 2 & 7 from the
# pdm:  bigpdm.
subpdm = bigpdm([1,3:5],[2,7])
```

This referencing format can also be used to assign data to parts of a larger PDM. This is shown in the following example.

```
# Replace the 3,2 block of a pdm with its absolute value
pdm1(3,2) = abs(pdm1(3,2))
```

When selecting subblocks of a DYNAMIC SYSTEM or PDM, the appropriate original labels (input, output and state; row, column and domain) are appended to the subblock.

3.2.4 Basic Functions

In all Xμ functions which operator on DYNAMIC SYSTEMs a matrix input is interpreted as a system with constant gain: the equivalent of `system([],[],[],mat)`. This interpretation also applies to binary operations between DYNAMIC SYSTEMs and matrices in both Xmath and Xμ. For example, `+`, `*`, `daug`,....

In binary operations between PDMs and matrices, the matrix is assumed to have a constant value at each element of the domain of the PDM. Again, this is consistent with the interpretation of a matrix as a system with no dynamics.

Note that this interpretation works for Xμ and the basic Xmath operations; it will not necessarily apply to other Xmath modules or more sophisticated Xmath functions.

Augmentation

Augmentation is the building of matrices from component pieces. DYNAMIC SYSTEMs and PDMs can be augmented with the same syntax as matrices. Data objects can be augmented, side by side, with the command `[A, B]`. Similarly, `[A; B]`, places `A` above `B`. In DYNAMIC SYSTEMs `[A, B]` is analogous to summing the outputs of the systems `A` and `B`. `[A; B]` is analogous to creating a larger system where the input goes to both `A`

and B. Augmentation for PDMs simply performs the augmentation at each domain variable. The domains must be the same.

Diagonal augmentation can be performed with the Xμ function `daug`. This is the equivalent of the matrix augmentation: `[A, 0; 0, B]`, except that up to 20 arguments can be augmented in one function call.

Algebraic Operations

In binary operations (e.g. `+`, `-`, `*`) between a DYNAMIC SYSTEM and a matrix, the matrix is assumed to represent the D matrix of a system. In other words, a system with constant gain (no dynamics).

The transpose operator (`'`) is well defined for constants, PDMs and DYNAMIC SYSTEMs. In the DYNAMIC SYSTEM case it creates the equivalent of the following.

```
[A,B,C,D] = abcd(sys)
transsys = system(A',C',B',D')
```

The conjugate transpose operator (`*'`) of a DYNAMIC SYSTEM creates the adjoint operator. In other words.

```
[A,B,C,D] = abcd(sys)
adjsys = system(-A',-C',B',D')
```

Random PDMs can be created in with the Xμ function `randpdm`. This is useful for generating random time domain signals for use in simulations.

DYNAMIC SYSTEM Functions

The poles and zeros of DYNAMIC SYSTEM can be found with the Xmath core functions `poles` and `zeros`.

Random systems can be created with the Xμ function `randsys`. The user can also specify additional system properties; for example, stability, a frequency range for the

poles, or a zero D term. Generating random systems is useful for simulating systems with unknown subsystems.

Specialized Xμ functions are provided for useful manipulations of the state. For example transforming the A matrix to a real valued, 2×2 block diagonal form; here referred to as modal form. These state-space functions are tabulated below.

Description	Xμ function
state similarity transform	`simtransform`
state reordering	`orderstate`
transform to modal form	`modalstate`

The `simtransform` function is can also be used on constant matrices and PDMs. Xmath has a transfer function data object which does not have a uniquely defined state. Applying `simtransform` or `orderstate` to a transfer function data object returns a warning and an unchanged transfer function. Applying `modalstate` to a transfer function gives the appropriate state-space representation. With `modalstate` it is possible to specify whether the resulting modes are in ascending or descending magnitude order and whether or not the unstable modes are ordered separately from the stable ones.

PDM Functions

A wide range of matrix functions can also be applied to PDMs. Xμ provides several additional functions which are often of use in typical control applications.

The function `spectrad` calculates the spectral radius (maximum magnitude eigenvalue) of a matrix or PDM.

Xmath provides an interpolation function (`interpolate`) which does only first order interpolation. There is also a built-in function, `spline`, for higher order spline fitting. Zero order interpolation is often required, particularly for looking at fine details in the inputs of Dynamic System responses. An additional Xμ interpolation function (`interp`) is used for this purpose. This command is also useful for mapping data on an irregularly spaced domain (from experiments for example) to a regular domain.

Simple decimation is easily performed on PDMs by direct indexing. The following example illustrates this.

```
# N is the decimation ratio.
smallpdm = bigpdm([1:N:length(bigpdm)])
```

3.2.5 Continuous to Discrete Transformations

Xmath has a single function, `discretize`, for calculating continuous to discrete transformations. Several options are offered including forward and backward difference, Z-transform equivalent, bilinear and pole-zero matching.

Xmath also has a `makeContinuous` function which performs the inverse of the `discretize` function. Naturally, it is only an inverse if certain pole/zero location and sampling period relations are assumed to hold.

3.3 Matrix Information, Display and Plotting

3.3.1 Information Functions for Data Objects

The data object information is available in Xmath via the variable window. This displays the data object classification, row and column information and any associated comment for each variable in the workspace. For PDMs or DYNAMIC SYSTEMs the domain or state dimension is also displayed. If open, this window is updated continuously; closing it will speed up calculations.

The Xmath command `who` displays dimensional information in the command log window. PDMs and DYNAMIC SYSTEMs both appear as three dimensional objects and it is not possible to distinguish between them using this command alone. The core function `whatis` displys the data object type.

Within an Xmath function, data object attributes can be determined with the `check` or `is` functions. These have similar formats, with `check` being the more powerful. The function `size` is used to determine the actual dimensions.

3.3.2 Formatted Display Functions

It is often useful to consider complex numbers in terms of frequency and damping. This is particularly applicable when studying the poles or zeros of a system. The command `rifd` provides a real-imaginary-frequency-damping formatted display for complex valued input. The following example illustrates the most common usage.

```
# display the poles locations of system:  sys1
rifd(poles(sys1))
```

Using the keyword {`discrete`} will give the display in terms of z-plane rather than the s-plane. If the input to `rifd` is a DYNAMIC SYSTEM then both the poles and zeros are calculated and displayed.

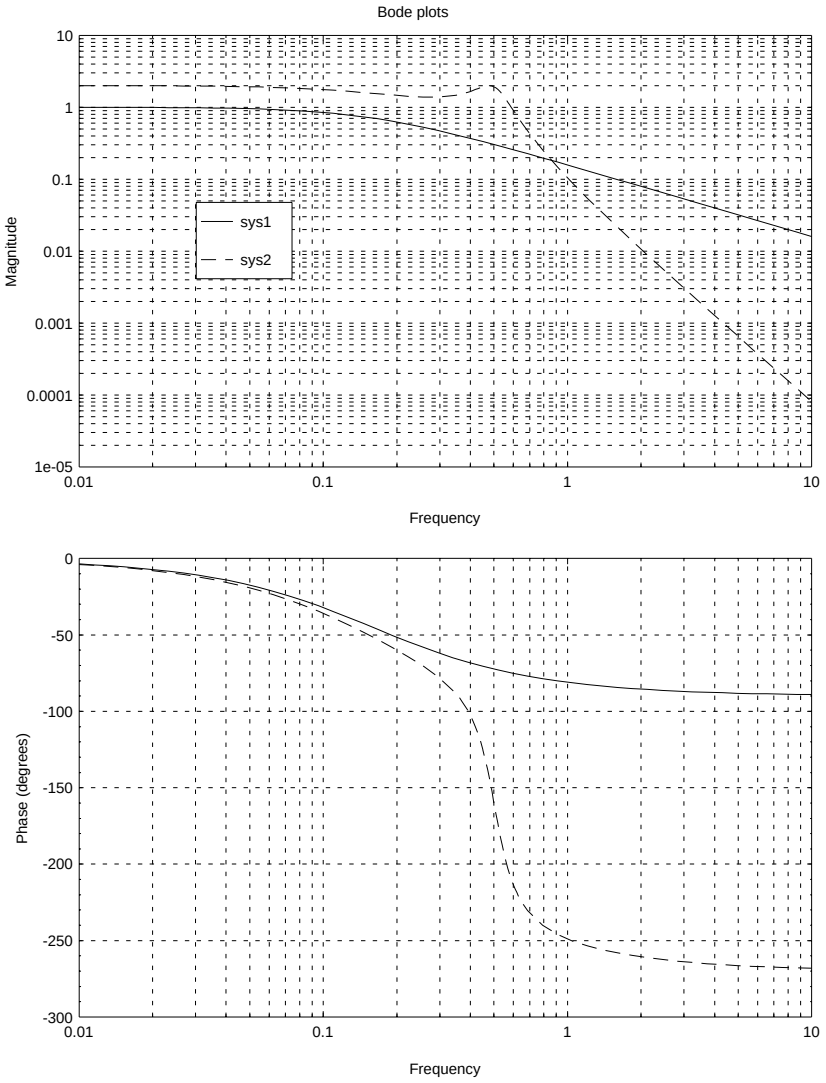
3.3.3 Plotting Functions

As PDMs are a native data object in Xmath, the standard Xmath `plot` function will correctly plot a PDM. Multiple PDMs over differing domains can be handled with repeated calls to the `plot` function by using the graphical object created from the previous `plot` call. The Xmath Basics manual describes this feature in detail.

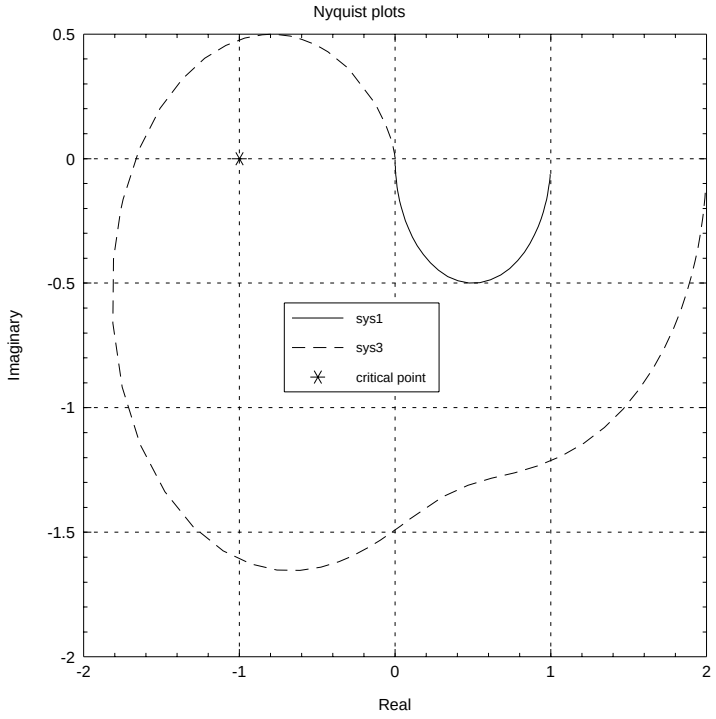
The Xμ function `ctrlplot` provides more control system specific plotting capabilities. Keywords allow the user to easily specify Bode, Nyquist, Nichols and other plots. Because it handles graphic objects in a similar manner to `plot`, the resulting plots can be modified by subsequent `plot` function calls. An example is given below.

```
sys1 = 1/makepoly([1,1],"s")
sys2 = 2*sys1*10/makepoly([1,1,10],"s")

w1 = logspace(0.01,10,50)'
w2 = sort([w1;[0.35:0.01:0.65]'])
sys1g = freq(sys1,w1)
sys2g = freq(sys2,w2)

# Bode plots
```

```
g1 = ctrlplot(sys1g,{bode});
g1 = ctrlplot(sys2g,g1,{bode});
g1 = plot({keep=g1,title = "Bode plots",...
           legend = ["sys1","sys2"]})?
```



```
# Nyquist plots

g2 = ctrlplot(sys1g,{nyquist});
g2 = ctrlplot(sys2g,g2,{nyquist});
g2 = ctrlplot(-1,g2,{nyquist,marker=1,line=0});
g2 = plot(g2,{projection="orthographic",...
              legend=["sys1","sys3","critical point"],...
              title="Nyquist plots"})?
```



3.4 System Response Functions

3.4.1 Creating Time Domain Signals

The Signal Analysis Module contains several functions which are useful for building time domain signals: `gcos` and `gsin`. Xμ provides `gstep` for the creation of stair-step signals. The example below illustrates the generation of a sine wave and a step function.

```
# A unit step over 10 seconds
time = 0:10:.1
y1 = gstep(time)
# multiple steps over 10 seconds
sdata = [1;-2;4]
tdata = [3;5;7]
y2 = gstep(time,tdata,sdata)
# Freq:  2 Hz, Ampl:  0.5
y3 = 0.5*gsin(time,2)
```

Because of the native PDM data type, and the ease in which PDMs can be augmented, vector (or even matrix) valued signals are easily created. For example:

```
# A function depending on t
time = 0:10:.1
y1 = pdm(exp(0.1*time),time) - gsin(time,3/(2*pi))
# A function independent of t
y2 = uniform(time)
# A 2 x 2 pdm
pt = pdm(time,time)
y3 = [pt/max(time),2*Cos(3*pt+0.2));...
     2+Sin(2*pt),sqrt(pt)+0.3*random(pt)]
```

3.4.2 Dynamic System Time Responses

In Xmath time responses can be calculated by simply multiplying a Dynamic System by a PDM. A zero-order hold equivalent is used with the sampling interval set to the

spacing in the input signal PDM. This means that the input PDM must be regularly spaced. Stair-case input functions with relatively few data points will often give erroneous results. The input signal should be interpolated (with `interp`) before being multiplied with the DYNAMIC SYSTEM. The function `defTimeRange` will calculate a default integration step size and form the required time vector. This can be useful in creating an input PDM or as an input to `interp`. Note that the initial state is specified within the Xmath DYNAMIC SYSTEM.

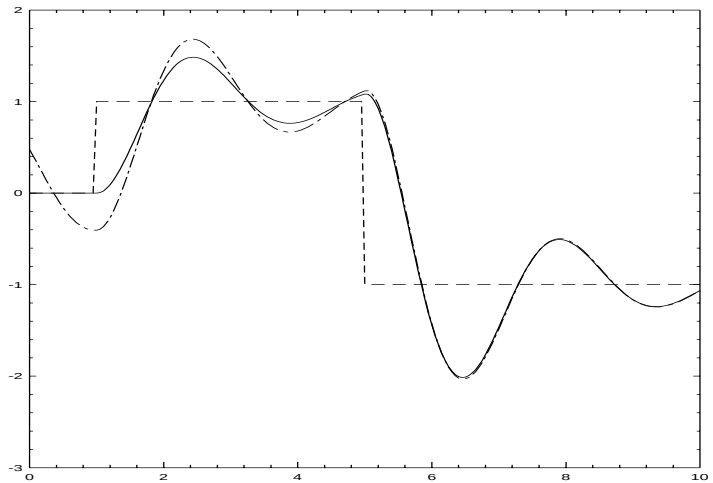
Discrete time DYNAMIC SYSTEMs do not require an additional command as the time increment is contained within the DYNAMIC SYSTEM data object. The time step in the input PDM must of course match the discrete system sample time.

Xmath also provides several core functions for common time response calculations: `step` calculates the unit step response; `impulse` calculates the impulse response; and `initial` calculates the unforced response from initial conditions specified within the DYNAMIC SYSTEM or provided as input arguments. In each of these functions the user may specify a time to override the default selected by `defTimeRange`.

Time response calculations are illustrated in the following example. Several of the functions discussed in the previous sections are also used.

```
# Create a second order lightly damped system.

sys = 5/makepoly([1,1,5],"s")

# Create an input signal with a step to +1 at
# 1 second and a step to -1 at 5 seconds.  A
# 10 second signal is created.

u = gstep([0:0.05:10],[0;1;5],[0;1;-1])

# Calculate the time response.

y0 = sys*u

# Repeat this calculation with an initial
# condition of [-1;0].  Note that this is
# only meaningful for a specified realization
# and Xmath forces us to choose one.  Here we
# choose that returned as the abcd default
```

```
[a,b,c,d] = abcd(sys)
sys = system(a,b,c,d)
y1 = system(sys,{X0=[-1;0]})*u

# Now plot the result

g1 = ctrlplot(u,{line_style=2});
g1 = ctrlplot(y0,g1,{line_style=1});
g1 = ctrlplot(y1,g1,{line_style=4});
g1 = plot(g1,{!grid})?
```



The native Xmath time domain simulation, using the `*` operation, has been supplemented with an Xμ function: `trsp`. This provides automatic selection of the integration time and, if necessary, interpolates the input signal appropriately. `trsp` also gives the option of linear interpolation of the input vector and a triangle hold equivalent discretization of the integration. This simulates the response of the system to signals which are linearly interpolated between samples. The syntax of the function is illustrated below.

```
y = trsp(sys,u,tfinal,{ord,intstep})
```

`sys`, `u`, and `tfinal` are the dynamic system, input signal and final time respectively. `ord`

and `intstep` are the interpolation order and integration step size. The system must be continuous. The integration order and sample time are prespecified for discrete time systems making the `*` operator is suitable for such simulations.

Xμ provides a sampled data simulation function (`sdtrsp`) based on the interconnection structure shown in Figure 3.1.

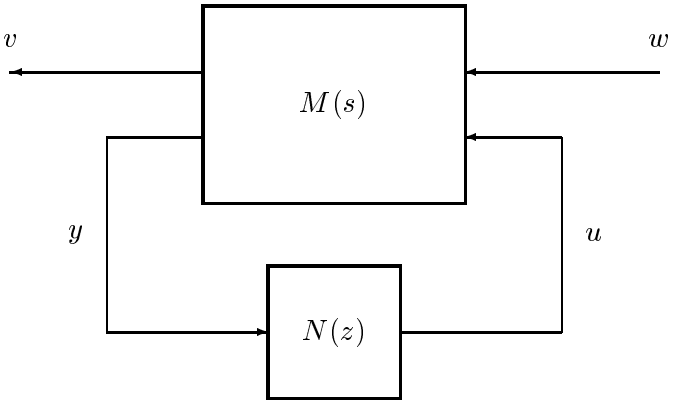


Figure 3.1: Interconnection structure for sampled data simulation

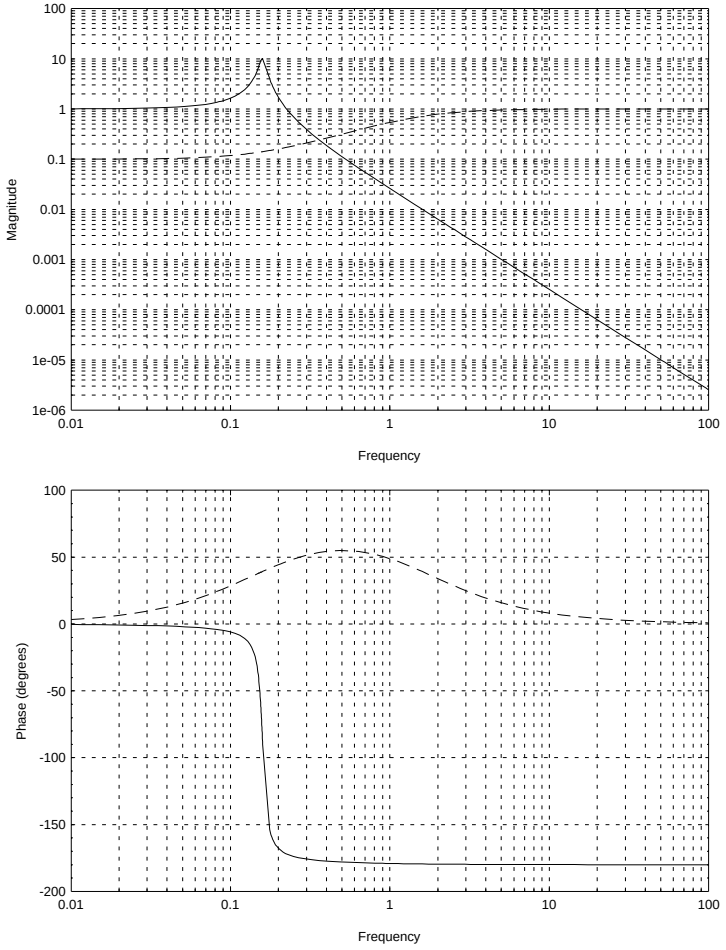
The signals v and y are the continuous outputs of the continuous system, $M(s)$. The simulation assumes that y is sampled and used as the input to the discrete system, $N(z)$. The output of $N(z)$, $u(s)$ is assumed to be held with a zero order hold. All signals are calculated with the sample spacing determined by the continuous time system integration step size. The output of $N(z)$ can be delayed by a fraction of the discrete time system sample time. This is useful for simulating systems with a partial period calculation delay; a situation which frequently occurs in practice.

3.4.3 Frequency Responses

The Xmath function `freq` is used for frequency response calculation. The input is a Dynamic System and a vector of frequency points. The frequency points can alternatively be specified by minimum and maximum variables and one of two other choices: the total number of points or phase tracking. If the total number of points is specified these are logarithmically spaced. If phase tracking is specified the points are selected such that the phase difference between points is less than a prespecified

maximum. This can be handy when first examining a high order system. An example is given below.

```
# Create a single-input, two-output system.

sys = [1/makepoly([1,0.1,1],"s");...
        makepoly([1,1],"s")/makepoly([1,10],"s")]

# The automatic point selection is used.

sysg = freq(sys,{Fmin=0.01,Fmax=100,track})

# Now plot the result

ctrlplot(sysg,{bode})?
```



Note that the default frequency units in Xmath are Hertz. This applies generically to all functions where the user specifies frequency information, although many functions allow radians/second to be specified via a keyword.

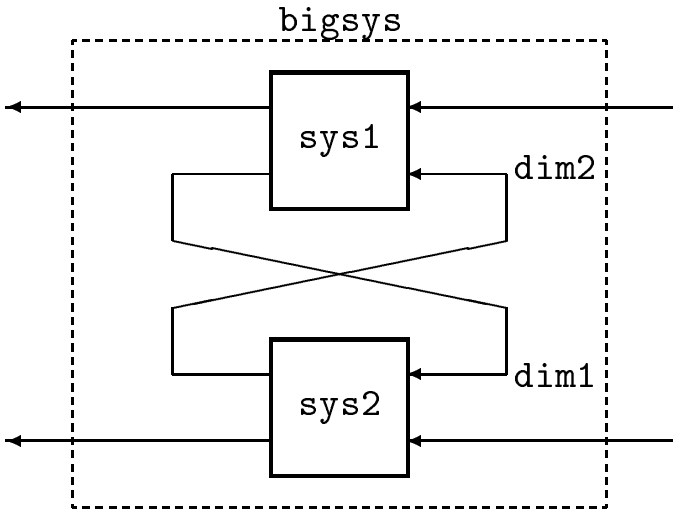


Figure 3.2: Generic Redheffer interconnection structure

3.5 System Interconnection

Interconnections of systems are used extensively in the design, analysis and simulation calculations. The most general form of interconnection, and the one used in forming closed loop systems, is the Redheffer (or star) product. The Xμ function `starp` performs this operation. This Redheffer product operation is illustrated in Figure 3.2. A MIMO Dynamic System (or matrix) is created from two MIMO Dynamic Systems.

The syntax of the command to form this interconnection is,

```
bigsys = starp(sys1,sys2,dim1,dim2)
```

A more powerful, general, interconnection capability is given with the Xμ function `sysic`. This can be used to interconnect subsystems (Dynamic Systems or constant gain matrices) in an arbitrary manner. This is best illustrated by an example. Consider the interconnection shown in Figure 3.3.

In each case it is assumed that the systems `p`, `c`, and `wght`, exist in the workspace prior to calling `sysic`. The final four-input, three-output system is shown in Figure 3.4.

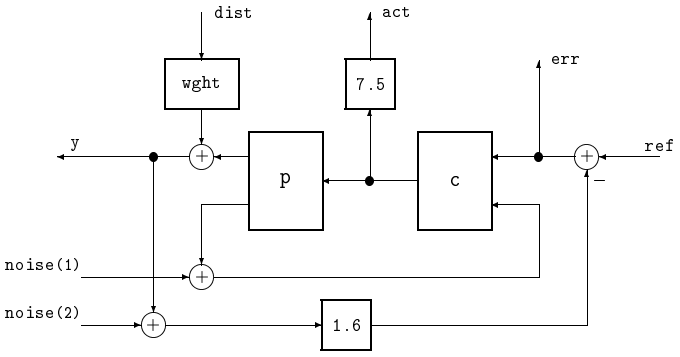


Figure 3.3: Example interconnection of subsystems

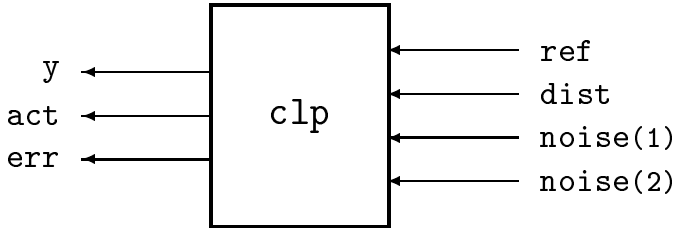


Figure 3.4: Example interconnected system

Using `sysic` to form this interconnection can be considered as four distinct operations.

- Specify the individual subsystems.
- Name and dimension the input signals.
- Specify, algebraically in terms of subsystem outputs or input signals, the output of the interconnected system.
- Specify, algebraically in terms of subsystem outputs or input signals, the input to each of the subsystems.
- Name the closed loop system.

The Xμ script used to create the closed-loop system variable `clp` is shown below. A single-input, two-output plant is considered. sysic is used to form the closed loop system for a particular controller. Noise and disturbance signals, in addition to a command reference, are considered as inputs. Tracking error and the control actuation are the outputs of interest.

```
# form the subsystems

p1 = [1;1]*(1/makepoly([1,1,40],"s"));
p = daug(1,1/makepoly([1,1],"s"))*p1;
c = [50*makepoly([1,1],"s")/makepoly([1,10],"s"),-20];
wght = 0.01;
```

Name the subsystems - these must match with the workspace variables.

```
snames = ["p"; "c"; "wght"]
```

Name the final system inputs (names are arbitrary) Parenthesis are used to give a dimension > 1.

```
inputs = ["ref"; "dist"; "noise(2)"]
```

Specify the outputs in terms of the subsystem names and the input names. Note that individual outputs of MIMO systems and simple arithmetic combinations can be specified. Parenthesis specify which output of a MIMO system is to be used.

```
outputs = ["p(1) + wght"; "7.5*c";
           "ref - 1.6*p(1) - 1.6*noise(2) - 1.6*wght"]
```

Now specify the inputs to each of the named subsystem. The order is important. Note how the input to a multiple-input system ("c") is specified.

```
conx = ["c";...
       "ref-1.6*p(1)-1.6*noise(2)-1.6*wght; p(2)+noise(1)";...
       "dist"]
```

Now the final system (`clp`) is formed.

```
clp = sysic(snames,inputs,outputs,conx,p,c,wght)

size(clp)?
ans (a row vector) =   3    4    4

rifd(clp)

Poles:

      real        imaginary      frequency      damping
                                 (rad/sec)       ratio

  -1.5627e+00     0.0000e+00     1.5627e+00     1.0000
  -4.4373e+00     0.0000e+00     4.4373e+00     1.0000
  -3.0000e+00    -9.4375e+00     9.9028e+00     0.3029
  -3.0000e+00     9.4375e+00     9.9028e+00     0.3029


Zeros:

      real        imaginary      frequency      damping
```

```
                                    (rad/sec)       ratio

   -1.0000e+00      0.0000e+00      1.0000e+00      1.0000
   -5.0000e-01      6.3048e+00      6.3246e+00      0.0791
   -5.0000e-01     -6.3048e+00      6.3246e+00      0.0791
   -1.0000e+01      0.0000e+00      1.0000e+01      1.0000
```

The order of the names in the `systemname` variable, must match the order of the rows in the `connections` variable and the order of the last arguments in the `sysic` function call.

3.6 $\mathcal{H}_2$ and $\mathcal{H}_\infty$ Analysis and Synthesis

This section discusses the synthesis functions available in Xμ. A weighted interconnection structure is set up so that either $\mathcal{H}_2$ or $\mathcal{H}_\infty$ design methods can be applied. This discussion assumes that the reader is familiar with the theory and application of these methodologies. Section 2.3 gives a more detailed overview of the theory and outlines the algorithms used in the calculations. For specific design examples and further discussion refer to the demos given in Section 4.1.

3.6.1 Controller Synthesis

The generic synthesis interconnection structure is illustrated in Figure 3.5. The objective is to design $K(s)$ such that the closed loop interconnection is stable and the resulting transfer function from w to e (denoted by $G(s)$) satisfies either an $\mathcal{H}_2$ or $\mathcal{H}_\infty$ norm objective.

Xμ provide functions to calculate the controllers minimizing either the $\mathcal{H}_2$ or $\mathcal{H}_\infty$ norm of $G(s)$; `h2syn` and `hinfsyn` respectively.

Recall from Section 2.3 that the minimizing $\mathcal{H}_2$ norm controller (calculated by `h2syn`) is unique. The format of this function is given below.

```
k = h2syn(p,nmeas,ncon)
```

The variable `p` is the open loop interconnection structure ($P(s)$ in Figure 3.5). This

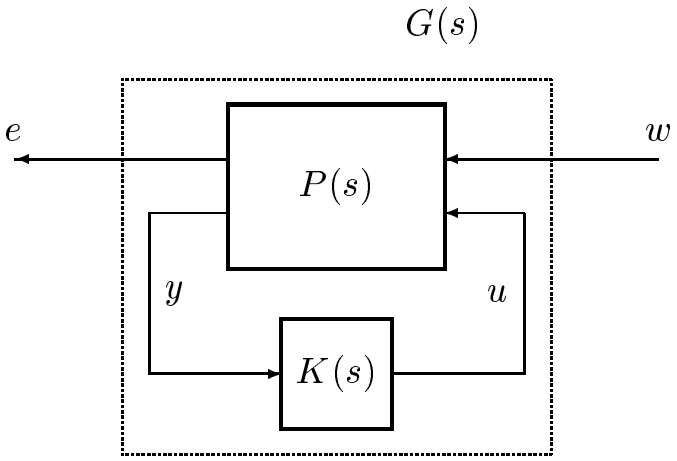


Figure 3.5: Interconnection structure for controller synthesis

structure contains more than just the open loop plant. It typically also contains frequency dependent weighting functions and specifies the structure of the interconnection between the open loop plant and the controller. The dimensions of y and u are specified by `nmeas` and `ncon`. The underlying Riccati equations can be solved by either an eigenvalue or Schur decomposition method. A keyword specifies the desired Riccati solution method. A simple is example is given at the end of this section.

The `hinfsyn` function calculates a controller, $K(s)$, which makes $\|G(s)\|_\infty \leq \gamma$ for a user specified γ. It is not possible to make γ arbitrarily small; there is a minimum value for γ (referred to as γ_{opt}) and γ_{opt} is not known *a priori* in a design problem. Therefore `hinfsyn` can also perform a bisection search for the smallest $\gamma > \gamma_{\text{opt}}$ and use this value of γ for the control design. Again, Section 2.3 gives the relevant theoretical details.

The syntax of `hinfsyn` is illustrated below. The final bound on the achieved γ is returned as `gfin`.

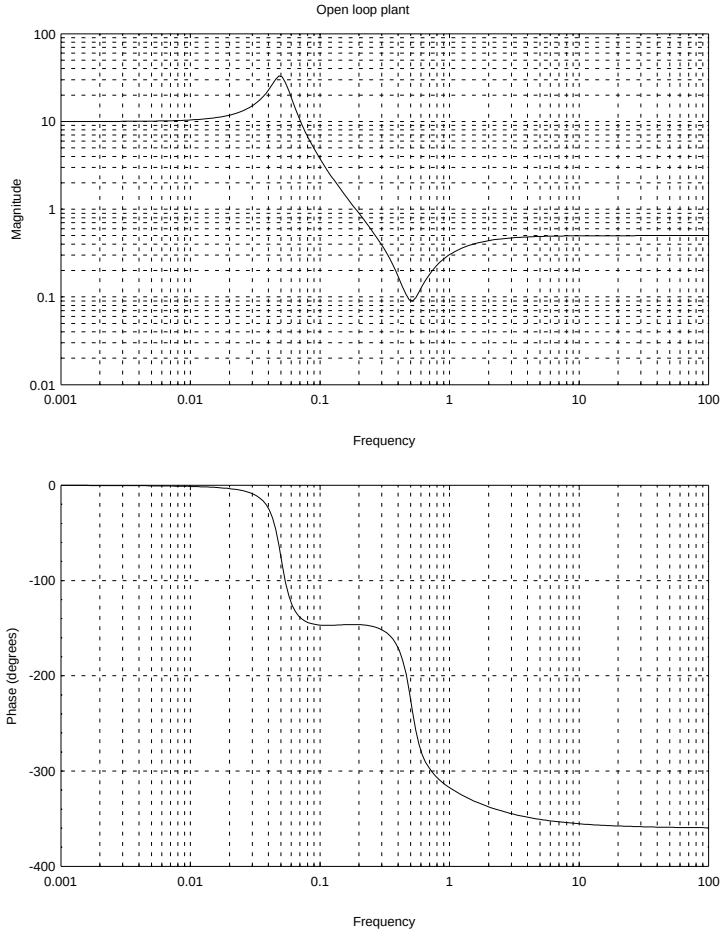
```
[k,gfin] = hinfsyn(p,nmeas,ncon,gamma)
```

If `gamma` is a scalar, the controller achieving that γ value is calculated, if one exists. If `gamma` is a two element vector a bisection search for the smallest γ value is performed. The function displays various intermediate calculations related to the eigenvalues of the Hamiltonian and the positivity of the Riccati solutions. As γ approaches γ_{opt} the Riccati

equation solution procedure often becomes poorly conditioned. Displaying intermediate calculation results allows the user to fine tune several tolerances if necessary. The intermediate Hamiltonian and Riccati solution details are displayed as the bisection proceeds. The bisection stopping tolerance, Riccati solution tolerances and the Riccati solution method are specified via keywords. Details on the meaning of these tolerances are given in Section 2.3.5.

We now give a simple example illustrating the use of these functions. An oscillatory non-minimum phase SISO system is to be controlled in a unity gain negative feedback configuration. This example is for pedagogical purposes only and does not illustrate the generality of the approach with respect to MIMO systems and more general control configurations. The weighting functions have been chosen to be appropriate for an $\mathcal{H}_\infty$ design. The performance weight, `Wperf`, is close to $1/s$ giving good tracking. The actuator weight, `Wact`, increases at high frequency, penalizing fast actuator action. A sensitivity function and step response comparison have been included to illustrate typical function calls for these procedures. Inappropriate weight choices make this comparison blantantly unfair to the $\mathcal{H}_2$ approach; the user should not drawn any conclusions about the relative merits of either approach from this example.

```
# Set up a simple closed loop problem.
plant = makepoly([0.1,-0.1,1],"s")*makepoly([1,1],"s")...
        /(makepoly([1,0.1,.1],"s")*(makepoly([0.2,1],"s")))

# Examine the plant frequency response.

omega = logspace(0.001,100,200)
plantg = freq(plant,omega)
g0 = ctrlplot(plantg,{bode});
g0 = plot(g0,{title="Open loop plant"})?
```



The desired closed loop configuration is illustrated in Figure 3.6.

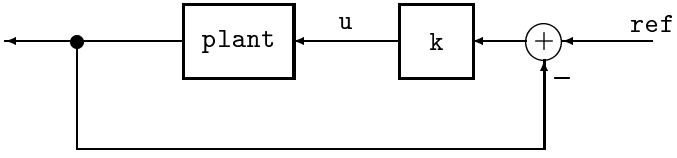


Figure 3.6: Closed loop configuration

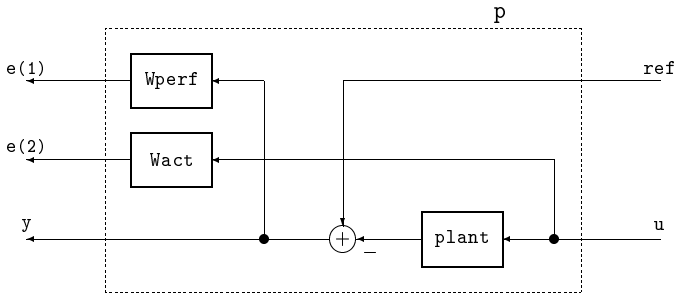
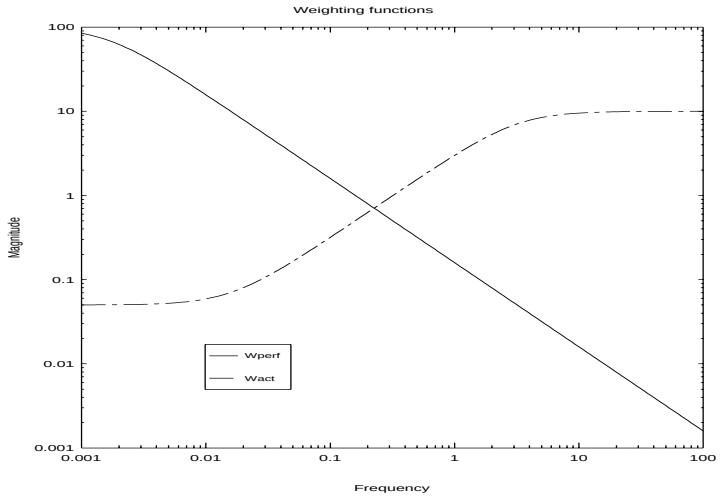


Figure 3.7: Weighted design interconnection structure: `p`

In order to set up the design problem, we consider `ref` as an unknown input and the tracking error (input to `k`), and the actuator signal, `u`, as outputs to be minimized. These outputs are weighted with the weights `Wperf` and `Wact` respectively. The weighted interconnection structure for design, `p`, is illustrated in Figure 3.7.

A more realistic problem would also include weighted noise on the measurement signal, `y`. We could also weight the `ref` input and add weighted disturbances to the plant input or output.

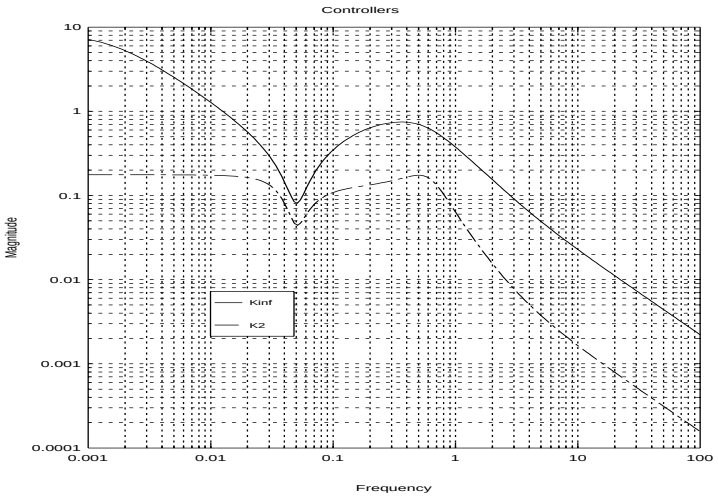
```
# Create weights

Wperf = 100/makepoly([100,1],"s")
Wact = makepoly([0.5,0.05],"s")/makepoly([0.05,1],"s")

Wperfg = freq(Wperf,omega)
Wactg = freq(Wact,omega)
g00 = ctrlplot(Wperfg,{logmagplot});
g00 = ctrlplot(Wactg,g00,{logmagplot,line_style=4});
g00 = plot(g00,{title="Weighting functions",...
                legend=["Wperf";"Wact"],!grid})?
```



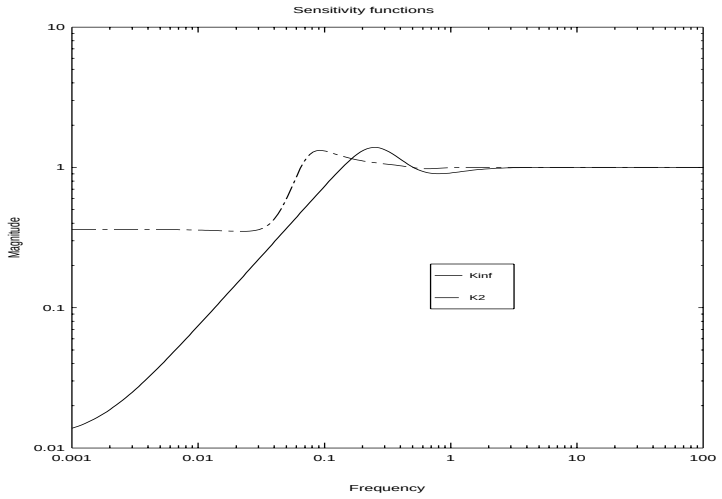

```
# Form the weighted interconnection structure

sysnames = ["plant";"Wperf";"Wact"]
sysinp = ["ref";"u"]
sysout = ["Wperf"; "Wact"; "ref-plant"]
syscnx = ["u";...           # input to plant
          "ref-plant";...   # input to Wperf
          "u"]              # input to Wact

p = sysic(sysnames,sysinp,sysout,syscnx,plant,...
              Wperf,Wact)

# Design Hinf controller

nctrls = 1
nmeas = 1
gmax = 25
gmin = 0
Kinf = hinfsyn(p,nmeas,nctrls,[gmax;gmin])

Test bounds:      0.0000  <  gamma  <=     25.0000
  gamma      Hx_eig     X_eig    Hy_eig      Y_eig   nrho_xy   p/f
   25.000     5.2e-01   1.7e-03   1.0e-02   0.0e+00   0.0000    p
```

```
   12.500     5.2e-01   1.7e-03   1.0e-02   0.0e+00   0.0000    p
    6.250     5.2e-01   1.7e-03   1.0e-02   0.0e+00   0.0000    p
    3.125     5.1e-01   1.7e-03   1.0e-02   0.0e+00   0.0000    p
    1.562     5.0e-01   1.7e-03   1.0e-02   0.0e+00   0.0000    p
    0.781     3.9e-01  -9.8e+01   1.0e-02   0.0e+00   0.0000    f
    1.172     4.8e-01   1.8e-03   1.0e-02   0.0e+00   0.0000    p

 Gamma value achieved:     1.1719
rifd(Kinf)
Poles:

      real        imaginary      frequency      damping
                                 (rad/sec)       ratio

  -1.0000e-02     0.0000e+00     1.0000e-02     1.0000
  -1.0347e+00     0.0000e+00     1.0347e+00     1.0000
  -2.6846e+00     2.3017e+00     3.5362e+00     0.7592
  -2.6846e+00    -2.3017e+00     3.5362e+00     0.7592
  -1.2692e+01     0.0000e+00     1.2692e+01     1.0000


Zeros:

      real        imaginary      frequency      damping
                                 (rad/sec)       ratio

  -5.0000e-02     3.1225e-01     3.1623e-01     0.1581
  -5.0000e-02    -3.1225e-01     3.1623e-01     0.1581
  -5.0000e+00     0.0000e+00     5.0000e+00     1.0000
  -2.0000e+01     0.0000e+00     2.0000e+01     1.0000


# Design H2 controller

K2 = h2syn(p,nmeas,nctrls)
rifd(K2)

Poles:

      real        imaginary      frequency      damping
```

```
                                (rad/sec)       ratio

  -1.4046e-01    -2.3161e-01     2.7087e-01     0.5186
  -1.4046e-01     2.3161e-01     2.7087e-01     0.5186
  -1.5863e+00     3.4754e+00     3.8203e+00     0.4152
  -1.5863e+00    -3.4754e+00     3.8203e+00     0.4152
  -5.2060e+00     0.0000e+00     5.2060e+00     1.0000


Zeros:

      real        imaginary      frequency      damping
                                 (rad/sec)       ratio

  -5.0000e-02    -3.1225e-01     3.1623e-01     0.1581
  -5.0000e-02     3.1225e-01     3.1623e-01     0.1581
  -5.0000e+00     0.0000e+00     5.0000e+00     1.0000
  -2.0000e+01     0.0000e+00     2.0000e+01     1.0000


# Look at frequency responses of the controllers
# Kinf and K2

Kinfg = freq(Kinf,omega)
K2g = freq(K2,omega)
g1 = ctrlplot(Kinfg,{logmagplot});
g1 = ctrlplot(K2g,g1,{logmagplot,line_style=4});
g1 = plot(g1,{title="Controllers",legend=["Kinf";"K2"]})?
```




```
# Examine sensitivity functions

sensinf = inv(1 + plant*Kinf)
sensinfg = freq(sensinf,omega)
sens2 = inv(1 + plant*K2)
sens2g = freq(sens2,omega)

g2 = ctrlplot(sensinfg,{logmagplot});
g2 = ctrlplot(sens2g,g2,{logmagplot,line_style=4});
g2 = plot(g2,{title="Sensitivity functions",...
               legend=["Kinf","K2"],!grid})?
```

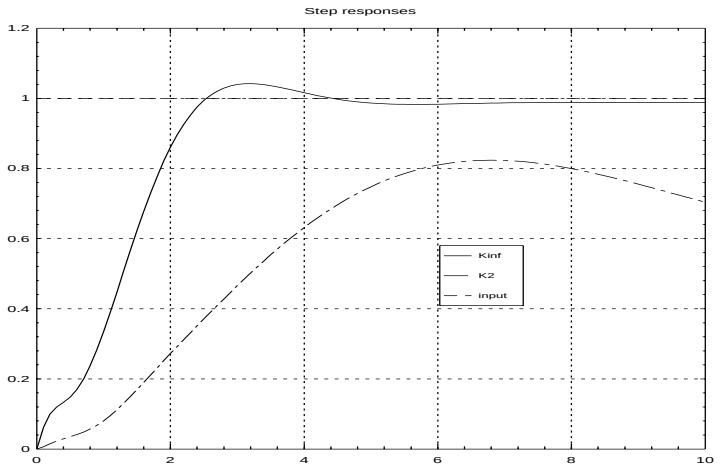


Note that with the interconnection structure shown in Figure 3.7, the closed loop transfer function from `ref` to `e(1)` is simply `Wperf*sensinf`. If the $\mathcal{H}_\infty$ of the resulting closed loop system was less than one, this would guarantee that `sensinf` was less than `inv(Wperf)` at every frequency. Note that from the above plot, this is not quite satisfied; as we might expect because the closed loop $\mathcal{H}_\infty$ norm is approximately 1.17.

Now look at some step responses. Use sysic to create an unweighted interconnection (because the weightings are only for design purposes and are not actually implemented in the system). We create an open-loop interconnection and close the loop with starp for each controller.

```
ic = sysic("plant",["ref";"ctrl"],["plant";"ref-plant"],...
           "ctrl",plant)
clpinf = starp(ic,Kinf)
clp2 = starp(ic,K2)

# Examine step response

step = gstep([0:0.1:10],0,1)
yinf = clpinf*step
y2 = clp2*step
g3 = ctrlplot(yinf);
g3 = ctrlplot(y2,g3,{line_style=4});
```

```
g3 = ctrlplot(step,g3,{line_style=2});
g3 = plot(g3,{title="Step responses",...
               legend=["Kinf";"K2";"input"]})?
```



3.6.2 System Norm Calculations

Functions are provided for calculating the $\mathcal{H}_2$ and $\mathcal{H}_\infty$ norms of DYNAMIC SYSTEMS. In the $\mathcal{H}_2$ case, this involves the solution of a Lyapunov equation. A bisection method, involving the calculation of eigenvalues of a scaled Hamiltonian matrix, is required for the $\mathcal{H}_\infty$ norm calculation.

The Xμ function for the two norm calculation is called `h2norm`. The syntax and operation are self explanatory.

The calculation of the $\mathcal{H}_\infty$ norm involves the iterative solution of a Riccati equation. The technique is a generalization of the the theoretical result given in Lemma 2 in Section 2.3.3. As a result, a tolerance can be specified, and the calculation gives upper and lower bounds. The function is `hinfnorm`. The syntax is illustrated below.

```
[out,omega] = hinfnorm(system,tol)
```

Bounds on the $\mathcal{H}_\infty$ norm are returned as `out`. An estimate of the frequency where the norm is achieved is returned as `omega`. Further control of the iteration is available via keywords.

The following example calculates the $\mathcal{H}_2$ and $\mathcal{H}_\infty$ norms of each of the closed loop systems arising from the previous example. Notice that `G2` has the minimum

$\mathcal{H}_2$ norm and `Ginf` has the minimum $\mathcal{H}_\infty$ norm. We can also see that the `Ginf` has a slightly lower norm than the bound guaranteed from the `hinfsyn` function call.

```
# Calculate the norms of each closed loop system.

Ginf = starp(p,Kinf)
G2 = starp(p,K2)

hinfnorm(Ginf)?
ans (a column vector) =

  1.17032
  1.16915

hinfnorm(G2)?
ans (a column vector) =

  36.2138
  36.1776


h2norm(Ginf)?
ans (a scalar) = 2.86197

h2norm(G2)?
ans (a scalar) = 2.78655
```

3.7 Structured Singular Value (μ) Analysis and Synthesis

This section covers the functions used in the D-K iteration procedure. The primary functions are the calculation of the controller (already discussed), the calculation of μ and the fitting of rational D scales. Several subroutines used for D scale fitting are useful in their own regard and are also discussed here. The discussion below assumes that the reader if familiar with the definition and use of μ; only the implementation issues are covered here. A simple application example is given here. To get a better idea of the standard approaches to μ and the D-K iteration, the reader should refer to the demos in Chapter 4. To find out more about the theoretical aspects of μ, including its application to robust stability and robust performance problems, refer to Section 2.4.

3.7.1 Calculation of μ

The function `mu` calculates the structured singular value. μ is is defined as a function of a matrix and a specified block structure. The most common use involves calculating μ at each frequency of a system frequency response. For this reason the μ function accepts PDM input arguments as well as matrices. The block structure is specified to the function in a coded form.

In general only upper and lower bounds for μ are calculated. The upper bound (for the square Δ block case) is given by,

$$\mu(M) \leq \inf_{D \in \mathcal{D}} \sigma_{\max}\left[DMD^{-1}\right],$$

where $\mathcal{D}$ is the set of all matrices which commute with the perturbation, Δ. Recall that the lower bound is given by,

$$\max_{Q \in \mathcal{Q}} \rho(MQ) \leq \mu(M),$$

where ρ denotes the spectral radius and $\mathcal{Q}$ is the set of all unitary perturbations of the specified block structure. The matrix Q achieving this maximization is a destabilizing Δ.

The outputs of the μ function are: the upper and lower bounds for μ; the D matrix for the upper bound; the Q matrix for the lower bounds; and a sensitivity estimate for the part of the D matrix corresponding to each block in Δ. The sensitivity estimate is essentially the gradient of the upper bound value with respect to the value of the D scale. It is useful in weighting the D scale fitting procedure.

The function syntax is shown below.

```
[mubnds,D,Dinv,Delta,sens] = mu(M,blk)
```

The upper and lower bounds are returned in `mubnds`. The variable `sens` gives a measure of the sensitivity of the upper bound to the D matrix. Both D and Q are returned as the matrices (or PDMs) `D` and `Delta`. Note that the inverse of D, used in the calculation of the upper bound, is also returned (as `Dinv`). This is done as, in the non-square Δ block case, the dimensions of D and D^{-1} are different.

The block structure is a vector of dimension: number of blocks $\times$ 2. For each block the output and input dimension is specified. To specify a scalar $\times$ identity block, the input dimension is set to zero.

A power iteration, with several random restarts, is used for the lower bound. The upper bound calculation uses an Osborne balancing method and enhances this with the Perron vector method for problems with less than 10 blocks. These methods have been found to be appropriate for the vast majority of practically motivated problems.

New algorithms for these calculations are currently under development. The most significant enhancement is the ability to calculate μ with respect to structures which include real valued blocks. Because of the development effort in this direction, a wide range of calculation options were not provided for the `mu` function.

The following example gives the simplest matrix with a gap between μ and the D-scale upper bound. It also illustrates the use of the `mu` function for constant matrices.

The following is the classic example showing that mu is not equal to its upper bound for more than three full blocks. We include a random scaling here to give a non-trivial D-scale.

```
gamma = 3 + sqrt(3); beta = sqrt(3) -1
a = sqrt(2/gamma); b = 1/sqrt(gamma)
```

```
c = 1/sqrt(gamma); d = -sqrt(beta/gamma)
f = (1+jay)*sqrt(1/(gamma*beta))
psi1 = -pi/2; psi2 = pi

U = [a,0; b,b; c,jay*c; d,f]
V = [0,a; b,-b; c,-jay*c; f*exp(jay*psi1), d*exp(jay*psi2)]
scl = diagonal(random(4,1)+0.1*ones(4,1))
M = scl*U*V*'*inv(scl)
```

Consider four 1×1 blocks. In this example mu is approximately 0.87.

```
blk1 = [1,1; 1,1; 1,1; 1,1]
[mubnds1,D1,Dinv1,Delta1] = mu(M,blk1)

max(svd(M))?            # a very crude upper bound
ans (a scalar) = 3.17155

max(svd(D1*M*Dinv1))?   # the D scale upper bound
ans (a scalar) = 1

mubnds1?
mubnds1 (a column vector) =

1
0.864113
```

Consider one 4×4 block (equivalent to maximum singular value).

```
blk2 = [4,4]
[mubnds2,D2,Dinv2,Delta2] = mu(M,blk2)
```

Note that the perturbation is such that $\det(I - MDelta) = 0$.

```
max(svd(D2*M*Dinv2))?
ans (a scalar) = 3.17155
```

```
mubnds2?
mubnds2 (a column vector) =

3.17155
3.17155

det(eye(4,4) - M*Delta2)?
ans (a scalar) = -2.62055e-16 + 5.82345e-17 j
```

3.7.2 The D-K Iteration

Recall from Section 2.5 that the D-K iteration is used as an approximation to μ synthesis. This section discusses how Xμ implements this procedure.

The D-K iteration procedure is as follows. The weighted design interconnection structure is referred to as `P`. The successive controllers are `K_i`, `i` = 1,... and the successive closed loop systems are `G_i`, `i` = 1,.... The block structure is coded within `blk`; `nmeas` is the number of controller measurements, and `nctrls` is the number of controller actuators outputs.

1. Set `i = 1`.
2. Design an initial $\mathcal{H}_\infty$ controller, `K_1`, for the interconnection structure, `P`.

   `K_1` = `hinfsyn`(`P`, `nmeas`, `nctrls`, `gamma_limits`).
3. Form the closed loop,

   `G_i` = `starp`(`P`, `K_i`).
4. Calculate μ(`G_i`) as follows.

   [`bnds`, `D_i`, `Dinv_i`, `Delta_i`, `sens_i`] = `mu`(`G_i`, `blk`).

   This calculation gives the D-scales for the upper bound: `D_i`. Figure 3.8 illustrates this step.
5. Compare the closed loop to the design specifications; this will involve more than just the calculation of μ. The user has several options at this point:

(a) Controller and closed loop are satisfactory so stop the iteration.

(b) The iteration has converged and the controller and closed loop are not satisfactory. In this case the weighted design problem must be reformulated.

(c) The iteration has not yet converged. Continue with step 6.

6. Fit rational approximations to `D_i` and `Dinv_i`. The function to do this, `musynfit`, is described in more detail in Section 3.7.3. A typical function invocation is,

$$[\texttt{Dsys\_i}, \texttt{Dinvsys\_i}] = \texttt{musynfit}(\texttt{D\_i}, \texttt{blk}, \texttt{nmeas}, \texttt{nctrls}, \texttt{sens\_i}).$$

7. Apply rational approximations to D-scales to the weighted interconnection structure. This is equivalent to,

$$\texttt{P\_D} = \texttt{Dsys\_i} * \texttt{P} * \texttt{Dinvsys\_i}.$$

8. Set `i = i + 1`

9. Design an $\mathcal{H}_\infty$ controller, `K_i`, for the interconnection structure, `P_D`.

$$\texttt{K\_i} = \texttt{hinfsyn}(\texttt{P\_D}, \texttt{nmeas}, \texttt{nctrls}, \texttt{gamma\_limits}).$$

This step is illustrated in Figure 3.9.

10. Go to step 3.

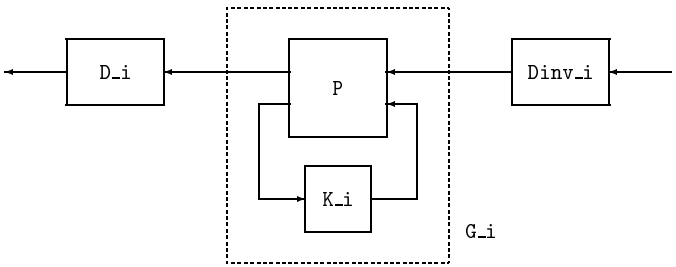


Figure 3.8: μ calculation in the D-K iteration: Step 3 in the enumerated procedure. Note that $\mu(\texttt{G\_i}) \leq \sigma_{\max}(\texttt{D\_i*G\_i*Dinv\_i})$

The above iteration uses a standard $\mathcal{H}_\infty$ design. It is possible to use the D-K iteration procedure with any MIMO design procedure ($\mathcal{H}_2$ for example).

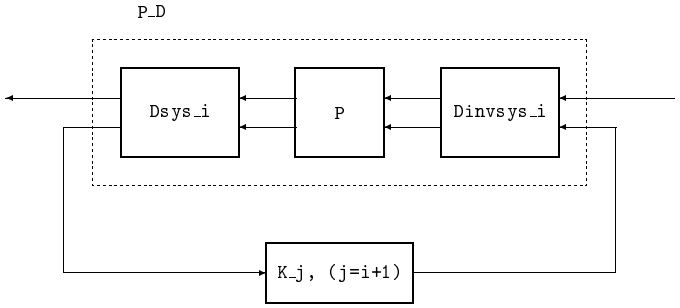


Figure 3.9: $\mathcal{H}_\infty$ controller design. Step 9 in the enumerated procedure

There is actually another possibility at step 5; numerical problems cause the iteration to diverge. As γ approaches its optimal value, the numerical properties of the calculation deteriorate. This may lead to `mu(G_i)` increasing as `i` is increased. This problem is observed more often in systems with very lightly damped modes.

A comparison of Figures 3.8 and 3.9 will show that `Dsys_i` is not quite a rational approximation to `D_i`. The reason is that `Dsys_i` has as inputs, the lower outputs of P. These are actually passed through an identity for the design of the next controller: `K_j` (with `j = i+1`). In other words,

$$\texttt{Dsys\_i} \approx \begin{bmatrix} \texttt{D\_i} & 0 \\ 0 & I \end{bmatrix}.$$

This identity is of dimension `nmeas` $\times$ `nmeas` and is the reason that `nmeas` and `nctrls` must be passed to the `musynfit` function. Do not confuse this identity with that corresponding to the last block in `D_i`.

3.7.3 Fitting D Scales

The Xμ D-scale fitting function is `musynfit`; the syntax is as follows.

```
[Dsys,Dinvsys] = musynfit(Dmagdata,blk,nmeas,nctrls,weight)
```

Both the D and D^{-1} systems (`Dsys` and `Dinvsys`) are returned. The D scale (`Dmagdata`) comes from a μ calculation on a closed loop system. However, `Dsys` and `Dinvsys` are required to multiply the open loop system. They must therefore contain the identity matrices for the inputs and outputs which correspond to the measurements and controls. This information is not contained in `blk` and must be specified in the argument list: `nmeas` is the number of measurements and `ncntrl` is the number of controls. The user can specify a frequency domain weight for the fitting. The variable `sens`, returned from the `mu` function, is a good option.

There are several choices of rational fitting functions available within `musynfit`. The D-scale input variables are magnitude data. Phase data, corresponding to a minimum-phase system is supplied with the Xμ function `mkphase`. The user has a choice of functions for the transfer function fitting: the Xmath function `tfid` and the Xμ function `fitsys`. The `fitsys` function is discussed in more detail in Section 3.7.3 below. For further information on `tfid` see the Xmath Basics Manual.

There are several choices of graphical display available to help the user select the most appropriate fit. These are:

1. The D-scale magnitude data and the last two transfer function fits are displayed.
2. The D-scale magnitude data and the last two transfer function fits are displayed. An additional plot shows the weighting function.
3. The D-scale magnitude data and the last two transfer function fits are displayed. An additional plot compares the μ upper bound (which uses the D-scale magnitude data) to the bound which would be obtained from a frequency response of the D-scale transfer function fit.

In a problem with n perturbation blocks, there are $n-1$ D scales requiring fitting. This is because one can be chosen as the identity without loss of generality. The D scale for each block requires user interaction for the selection of the system order. The function `fitsys` is called to fit each D scale block. This is available to the user and is described in more detail below.

Transfer Function Fitting Functions

The underlying functions used for the fitting of each block of the D scale are described here as they may be of independent interest to the user. Other possible uses include

creating weights from data and simple system identification.

Xμ provides two user callable functions for fitting SISO transfer functions to data. The first is `mkphase` calculates the phase corresponding to a minimum-phase stable system from magnitude data. This uses the complex cepstrum method described by Oppenheim and Schafer [80, p. 501] to generate the desired frequency response. The syntax of the function is given below.

```
cdata = mkphase(magdata)
```

The complex cepstrum method is used to generate the complex valued frequency response,`cdata`, of a SISO minimum-phase system with magnitude response `magdata`.

The Xμ function `fitsys` fits a dynamic system to frequency domain data. The syntax is illustrated below.

```
sys = fitsys(data,npoles,nzeros,weight)
```

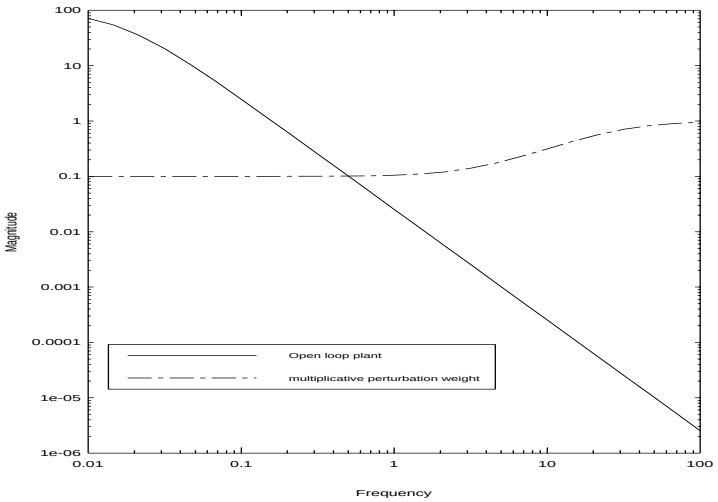
The number of poles and zeros in the system, `sys`, can be independently specified by the arguments `npoles` and `nzeros`. When called from `musynfit` these are always identical as an invertible system is required. This may not be appropriate for other applications of `fitsys`. For logscale frequency data a weight of $1/s$ is strongly recommended as this tends to balance the effects of high and low frequencies in the fit. In `musynfit` a $1/s$ weight is automatically applied and multiplies any other user specified weight.

Chebyshev polynomials are used as basis functions for both the numerator and denominator polynomials. For further information see the work by Adcock [81]. The `fitsys` function is similar to the Xmath function `tfid`. The algorithm in `fitsys` has been fine-tuned for D-scale fitting and usually outperforms `tfid` in this application.

We will now look at a simple design example using D-K iteration. The plant is a double integrator with an output perturbation. For more physically meaningful examples refer to Chapter 4.

```
# The nominal plant is a double integrator.
# A multiplicative perturbation weight reflects
# increased uncertainty at high frequencies
```

```
plant = 1/makepoly([1,0,-0.01],"s")
Wm = makepoly([1,20],"s")/makepoly([1,200],"s")
omega = logspace(0.01,100,25)
plantg = freq(plant,omega)
Wmg = freq(Wm,omega)
g1 = ctrlplot(plantg,{logmagplot});
g1 = ctrlplot(Wmg,g1,{logmagplot,line_style=4});
g1 = plot(g1,{!grid,legend=["Open loop plant";...
               "multiplicative perturbation weight"]})?
```



The open-loop system under consideration is illustrated in Figure 3.10.

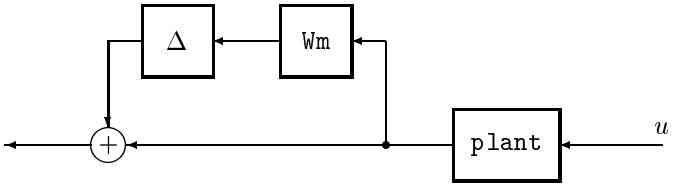
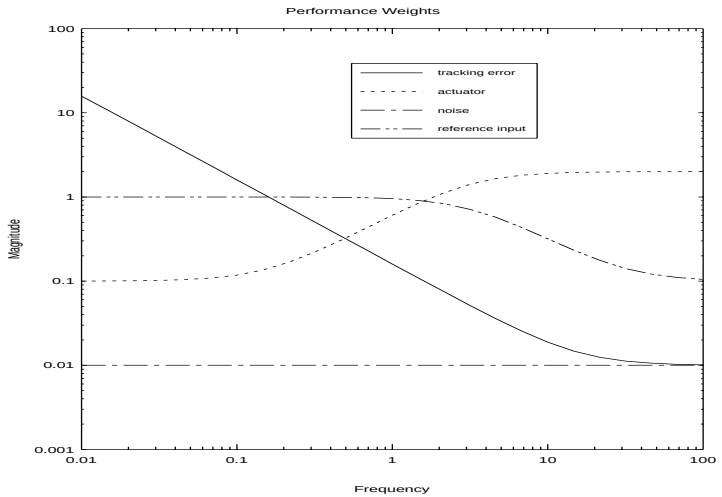


Figure 3.10: Open loop perturbation model

```
# Now include some weights for performance:
```

```
Wperf = makepoly([0.01,1],"s")/makepoly([1,0.01],"s")
Wact = 0.1* makepoly([1,1],"s")/makepoly([0.05,1],"s")
Wnoise = 0.01
Wref = makepoly([0.005,1],"s")/makepoly([0.05,1],"s")

Wperfg = freq(Wperf,omega)
Wactg = freq(Wact,omega)
Wnoiseg = conpdm(Wnoise,omega)
Wrefg = freq(Wref,omega)
g2 = ctrlplot(Wperfg,{logmagplot});
g2 = ctrlplot(Wactg,g2,{logmagplot,line_style=3});
g2 = ctrlplot(Wnoiseg,g2,{logmagplot,line_style=4});
g2 = ctrlplot(Wrefg,g2,{logmagplot,line_style=5});
g2 = plot(g2,{title="Performance Weights",y_min=0.001,...
             legend=["tracking error";"actuator";...
             "noise";"reference input"],!grid})?
```



Set up a weighted interconnection structure for unity gain negative feedback. This includes the perturbation weight as well as those for the design performance objectives. A scalar multiplier is factored out of the perturbation. This is done to give a more interesting D-scale problem for this example.

```
nms = ["plant";"Wm";"Wperf";"Wact";"Wnoise";"Wref"]
inp = ["delt";"ref";"noise";"u"]
outp = ["100*Wm"; "Wperf"; "Wact"; "Wref-0.01*delt-plant-Wnoise"]
cnx = ["u"; "plant"; "Wref-0.01*delt-plant"; "u"; "noise"; "ref"]
p = sysic(nms,inp,outp,cnx,plant,Wm,Wperf,Wact,Wnoise,Wref)
```

The interconnection structure, p, is illustrated in Figure 3.11.

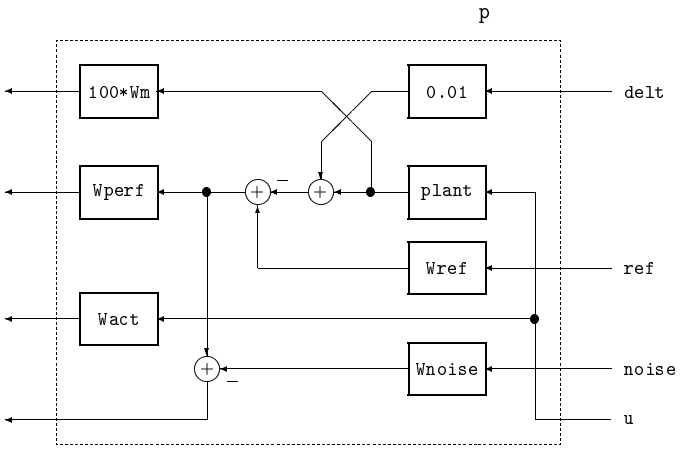
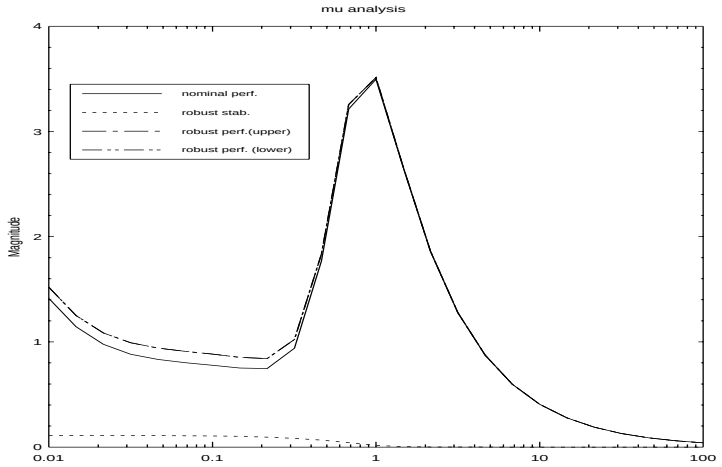


Figure 3.11: Weighted interconnection structure, p

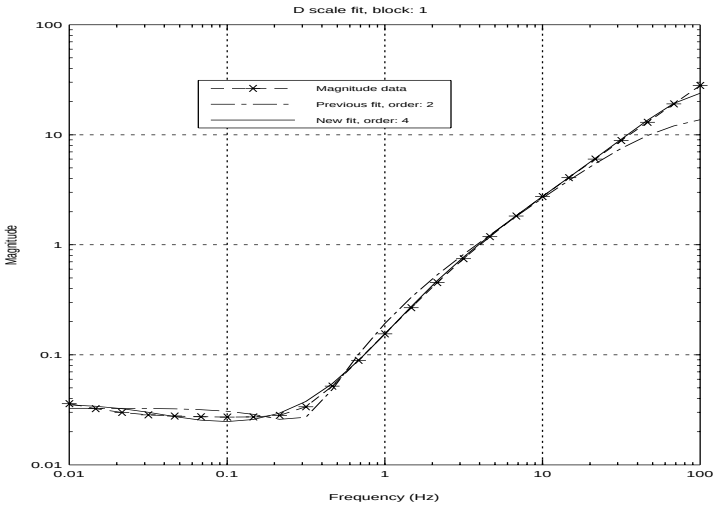
```
# Perform an H_infinity design.

nmeas = 1               # number of measurements
ncntrls = 1             # number of controls
gmin = 0
gmax = 100
Kinf = hinfsyn(p,nmeas,ncntrls,[gmin;gmax])

G = starp(p,Kinf)       # weighted closed loop system
Gg = freq(G,omega)
G11g = Gg(1,1)
G22g = Gg(2:3,2:3)
rs = max(svd(G11g))     # robust stability
np = max(svd(G22g))     # nominal performance
```

```
blk = [1,1; 2,2]
[rpbnds1,D1,Dinv1,Delta1,sens1] = mu(Gg,blk)

g3 = ctrlplot(np,{log});      # plot on a log-linear scale
g3 = ctrlplot(rs,g3,{log,line_style=3});
g3 = ctrlplot(rpbnds1,g3,{log,line_style=[4,5]});
g3 = plot(g3,{!grid,title="mu analysis",y_lab="Magnitude",...
        legend=["nominal perf.";"robust stab.";...
        "robust perf.(upper)";"robust perf.  (lower)"]})?
```

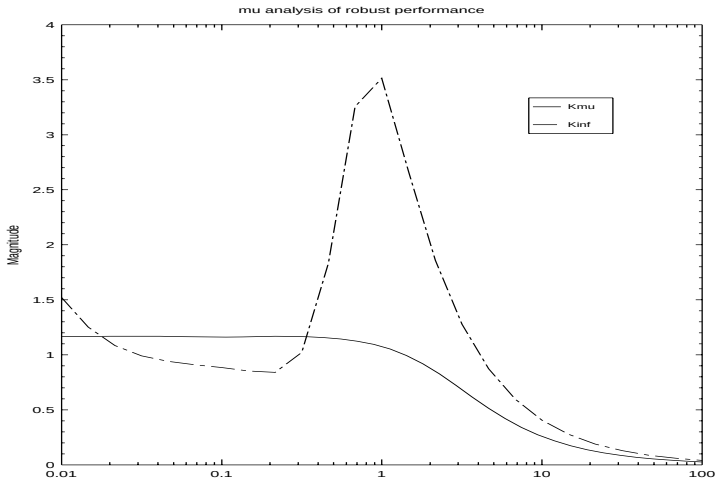



```
# Fit transfer functions to D1 & Dinv1 for a mu
# synthesis iteration

[Ds,Dinvs] = musynfit(D1,blk,nmeas,ncntrls,sens1,{!plotweight})
```

The following illustrates the `musynfit` screen display after selecting a 2nd order fit and then a 4th order fit.




```
# Apply the D scales to another H_infinity design

Kmu = hinfsyn(Ds*p*Dinvs,nmeas,ncntrls,[0;10])

# Close the loop around the weighted interconnection
# structure.

Gmu = starp(p,Kmu) # weighted closed loop (2nd it.)
omega = logspace(0.01,100,40)
Gmug = freq(Gmu,omega)

blk = [1,1; 2,2]
[rpbnds2,D2,Dinv2,Delta2,sens2] = mu(Gmug,blk)

# compare mu to the value from the previous iteration.

g4 = ctrlplot(rpbnds2(1,1),{log});
g4 = ctrlplot(rpbnds1(1,1),g4,{log,line_style=4});
g4 = plot(g4,{title="mu analysis of robust performance",...
         legend=["Kmu";"Kinf"],y_lab="Magnitude",!grid})?
```



3.7.4 Constructing Rational Perturbations

For simulation purposes it is useful to be able to construct a rational approximation to the Δ returned by the μ calculation. The approach is to choose a Δ at a particular frequency, for example the one where μ is at a maximum, and obtain a MIMO system which has a frequency response (gain and phase) equal to Δ at that frequency.

The function for this purpose is function `mkpert`. The syntax is given below.

```
pertsys = mkpert(Delta,blk,mubnds)
```

This function takes as arguments the variables `Delta`, `blk`, and `mubnds`. The meaning of these is identical to the `mu` case. The frequency selected for the interpolation is that where the lower bound (in `mubnds`) is maximum. Alternatively the user can use keywords to specify a frequency at which to do the interpolation and specify the norm of the resulting `pertsys`. `pertsys` will be an all-pass system.

Monte-Carlo simulation approaches require the ability to generate random perturbations having the correct block structure. The function for this purpose is `randpert` and its usage is illustrated below.

```
pert = randpert(blk,{sys,sfreq,complex,pnorm})
```

The user can specify whether the perturbation is a dynamic system or matrix, and whether it is real or complex valued, in addition to specifying the norm.

3.7.5 Block Structured Norm Calculations

It is possible to get an idea of the input/output combinations which are limiting the robust stability or robust performance of a system by examining the elements of the product, DMD^{-1} at the critical frequency. Large values indicate a potential problem in the corresponding input/output pair. For systems with MIMO blocks this can be simplified to looking at the block norm of the matrix. This is done by partitioning up the matrix into blocks which correspond to the inputs and outputs of each block of Δ. Each partition is then replaced by the maximum singular value of the partition. In the case where all the Δ blocks are 1×1 this reduces to the absolute value of the matrix. If there is only one Δ block this is equivalent to calculating the maximum singular value of the matrix.

The Xμ function for this purpose is called `blknorm` and its syntax is illustrated below.

```
normM = blknorm(M,blk,p,{Frobenius})
```

The user has the option of selecting other p norms ($1 \leq \mathrm{p} \leq \infty$) or the Frobenius norm. The Euclidean norm is the most commonly used and is the default.

3.8 Model Reduction

The model reduction functions described here are often useful in obtaining a lower order realization of a controller. μ-synthesis controllers for complicated systems are often of high order. The controller order can often be significantly reduced with very little degradation in the closed loop performance.

The user should be careful in reducing the order of an open-loop interconnection structure before doing a design. Very small changes in the interconnection structure can produce dramatic changes in the resulting closed loop system.

A greater range of model reduction functions is available in the Model Reduction Module. Some of the functions described here are cross-licensed with that module. Section 2.6 describes the theory behind these functions.

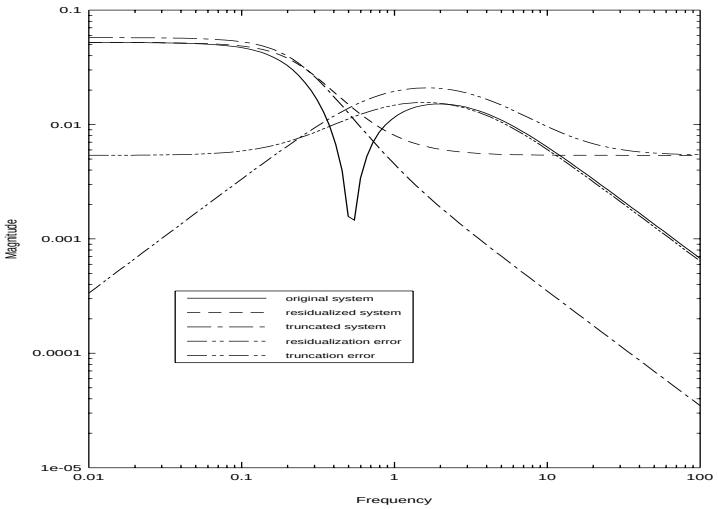
3.8.1 Truncation and Residualization

Truncation is provided by the `truncate` function (cross-licensed with the Model Reduction Module). Residualization is performed with the Xμ function: `sresidualize`. The user must specify the original system and the number of states to be retained. In both cases the states in the upper left corner of the A matrix are retained.

The following example illustrates the use of these functions and gives an idea of their error properties.

```
# Create a five state system for reduction.

a = daug(-0.891334,[-1.20857,0.799042;-0.799042,-1.20857],...
          -4.74685,-21.3013)
b = [0.0262569;-0.189601;-0.113729;0.211465;-0.538239]
c = [0.120725,-0.336942,0.397198,-0.700524,-1.02235]
d = 0
sys1 = system(a,b,c,d)

# Reduce to a 3 state system by residualization
# and truncation.

sysout1 = sresidualize(sys1,3)
sysout2 = truncate(sys1,3)

fHz = logspace(0.01,100,100)
sys1g = freq(sys1,fHz)
sysout1g = freq(sysout1,fHz)
sysout2g = freq(sysout2,fHz)
residerror = sys1g - sysout1g
truncerror = sys1g - sysout2g

g1 = ctrlplot(sys1g,{logmagplot});
g1 = ctrlplot(sysout1g,g1,{logmagplot,line_style=2});
```

```
g1 = ctrlplot(sysout2g,g1,{logmagplot,line_style=4});
g1 = ctrlplot(residerror,g1,{logmagplot,line_style=5});
g1 = ctrlplot(truncerror,g1,{logmagplot,line_style=6});
g1 = plot(g1,{!grid,legend=["original system";...
        "residualized system";"truncated system";...
        "residualization error";"truncation error"]})?
```

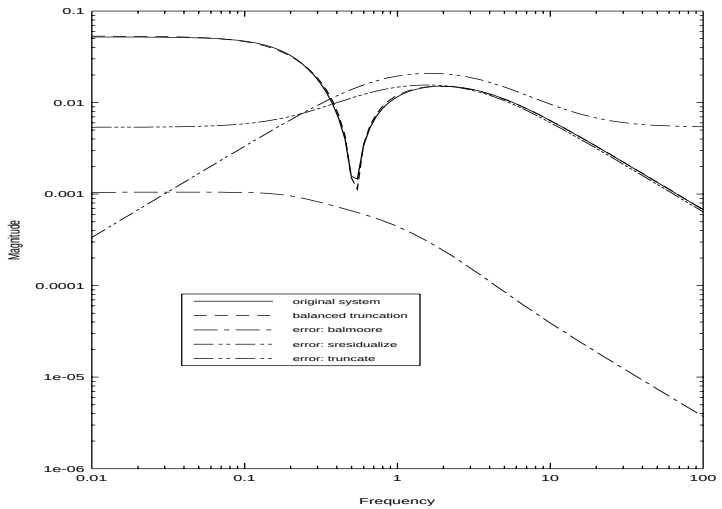


3.8.2 Balanced Realizations

The function `balmoore` (cross-licensed from the Model Reduction Module) produces a balanced realization and optionally truncates it. The can be used to obtain a balanced realization and the Hankel singular values without necessarily truncating the system. The following illustrates the use of the function using the same system as in the `truncate` and `sresidualize` example.

```
# Balanced truncation.

[sysout3,hsv] = balmoore(sys1,{nsr=3})
sysout3g = freq(sysout3,fHz)
balerr = sys1g - sysout3g
```

```
# Displaying the Hankel singular values shows which
# states are close to unobservable and uncontrollable.

hsv?
hsv (a column vector) =

  0.0741834
  0.0726887
  0.0264105
  0.000146401
  2.7699e-07

# Compare to the errors from the previous example.

g2 = ctrlplot(sys1g,{logmagplot});
g2 = ctrlplot(sysout3g,g2,{logmagplot,line_style=2});
g2 = ctrlplot(balerr,g2,{logmagplot,line_style=4});
g2 = ctrlplot(residerror,g2,{logmagplot,line_style=5});
g2 = ctrlplot(truncerror,g2,{logmagplot,line_style=6});
g2 = plot(g2,{!grid,legend=["original system";...
        "balanced truncation";"error:  balmoore";...
        "error:  sresidualize";"error:  truncate"]})?
```

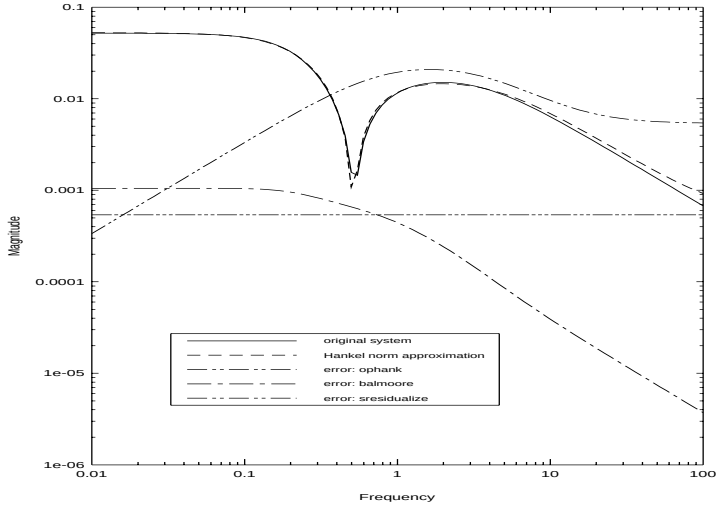


3.8.3 Hankel Singular Value Approximation

The function `ophank` (also cross-licensed from the Model Reduction Module) is used to perform optimal Hankel norm approximation. Recall from Section 2.6.3 that there is an astable system achieving the lower bound. The unstable part of this system is returned as the second argument.

The following applies the Hankel norm approximation technique to the previous example. The unstable part of the optimal approximation is `sysout4u` and the astable system, `sysout4 + sysout4u` would achieve the lower bound for the approximation error. The Hankel singular values can also be obtained by this function.

```
# Optimal Hankel singular value approximation

[sysout4,sysout4u,hsv] = ophank(sys1,{nsr=3})
sysout4g = freq(sysout4,fHz)
hankerr = sys1g - sysout4g

# Compare to the errors from the previous example.

g3 = ctrlplot(sys1g,{logmagplot});
g3 = ctrlplot(sysout4g,g3,{logmagplot,line_style=2});
g3 = ctrlplot(hankerr,g3,{logmagplot,line_style=6});
g3 = ctrlplot(balerr,g3,{logmagplot,line_style=4});
g3 = ctrlplot(residerror,g3,{logmagplot,line_style=5});
g3 = plot(g3,{!grid,legend=["original system";...
        "Hankel norm approximation";"error:  ophank";...
        "error:  balmoore";"error:  sresidualize"]})?
```


0.1
0.01
0.001
0.0001
1e-05
1e-06
Magnitude
0.01
0.1
1
10
100
Frequency
original system
Hankel norm approximation
error: ophank
error: balmoore
error: sresidualize

Chapter 4

Demonstration Examples

4.1 The Himat Example

The following demo can be run by executing the following Xmath command:

```
execute file = "$XMATH/demos/xMu/himatdemo"
```

4.1.1 Problem Description

The Himat is a small scale remotely piloted aircraft built to investigate high maneuverability fighter aircraft design. The vehicle was flight tested in the late 1970s. The example studied here considers control of only the longitudinal dynamics. These are taken to be uncoupled from the lateral-direction dynamics. The nominal model and control objectives are described by Safonov *et al.* [82]. Further details can be found in the work by Hartman *et al.* [83] and Merkel *et al.* [84]. The vehicle is currently attached to the outside wall of the Museum of Science in Los Angeles.

A four state rigid body model describes the dynamics. The states can be assigned the following physical meanings,

δv Perturbations along the velocity vector.

α Angle of attack. I.e. angle between the velocity vector and the aircraft's longitudinal axis.

q Rate-of-change of aircraft attitude angle.

θ Aircraft attitude angle.

Control can be exerted via the elevon and canard, denoted by δ_e and δ_c respectively. The angle of attack (α) and attitude angle (θ) are available as direct measurements.

A weighted output disturbance rejection problem will be considered. This problem also encompasses other maneuvering objectives. The closed-loop perturbation model of the vehicle is illustrated in Figure 4.1.

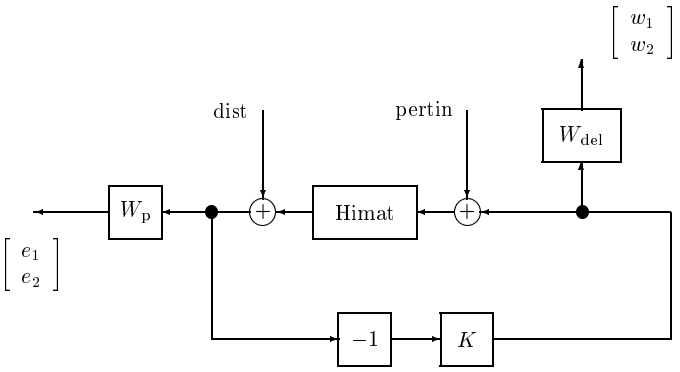


Figure 4.1: Himat open-loop perturbation model

Note that W_{p}, W_{del}, Himat and K are each two-input, two-output systems. Similarly, the inputs dist and pertin are two element vector signals. The desired result is the four-input, four-output system, denoted by `clp`, and shown in Figure 4.2. This is used in the design example given below.

4.1.2 State-space Model of Himat

The state space description of the Himat plane is given below. The states of the plant model are: forward speed (v), angle-of-attack (α), pitch rate (q) and pitch angle (θ).

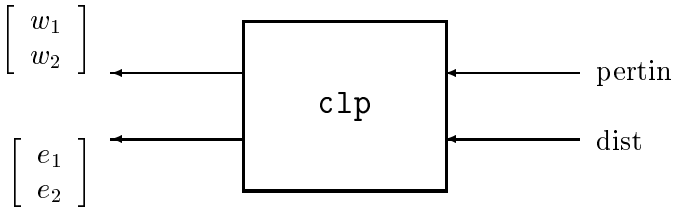


Figure 4.2: Interconnection structure for the himat design example

The inputs are the elevon position and the canard position. The outputs that are to be kept small are angle-of-attack (α) and pitch angle (θ).

The commands required to enter the state-space description are simply matrix assignments for each of A, B, C and D. Note that the states, inputs and outputs are named.

```
a = [-0.0226, -36.6000, -18.9000, -32.1000;...
           0,  -1.9000,   0.9830,        0;...
      0.0123, -11.7000,  -2.6300,        0;...
           0,        0,   1.0000,        0]

b = [      0,        0;...
     -0.4140,        0;...
    -77.8000,  22.4000;...
           0,        0]

c = [0, 57.3000, 0,       0;...
     0,       0, 0, 57.3000]

d = zeros(2,2)
himat = system(a,b,c,d)
comment himat "Himat vehicle state-space model"

states = ["forward speed";"angle-of-attack";"pitch rate";"pitch
angle"]
inputs = ["elevon";"canard"]
outputs = ["angle-of-attack";"pitch angle"]
himat = system(himat,{stateNames=states,inputNames=inputs,...
                        outputNames=outputs})?
```

```
himat (a state space system) =

  A
  -0.0226    -36.6    -18.9      -32.1
   0          -1.9      0.983      0
   0.0123    -11.7     -2.63       0
   0           0        1          0

  B
    0         0
   -0.414     0
  -77.8      22.4
    0         0

  C
  0    57.3    0     0
  0     0      0    57.3

  D
  0    0
  0    0

  X0
  0
  0
  0
  0

  State Names
  -----------
  forward speed    angle-of-attack    pitch rate    pitch angle

  Input Names
  -----------
  elevon
  canard

  Output Names
  ------------
```

```
angle-of-attack
pitch angle

System is continuous
```

4.1.3 Creating a Weighted Interconnection Structure for Design

The multiplicative input perturbation weight, W_{del}, is constructed as a transfer function. The weight is,

$$W_{\text{del}} = \frac{50(s+100)}{(s+10000)}.$$

The output error weight, W_{p}, is created as,

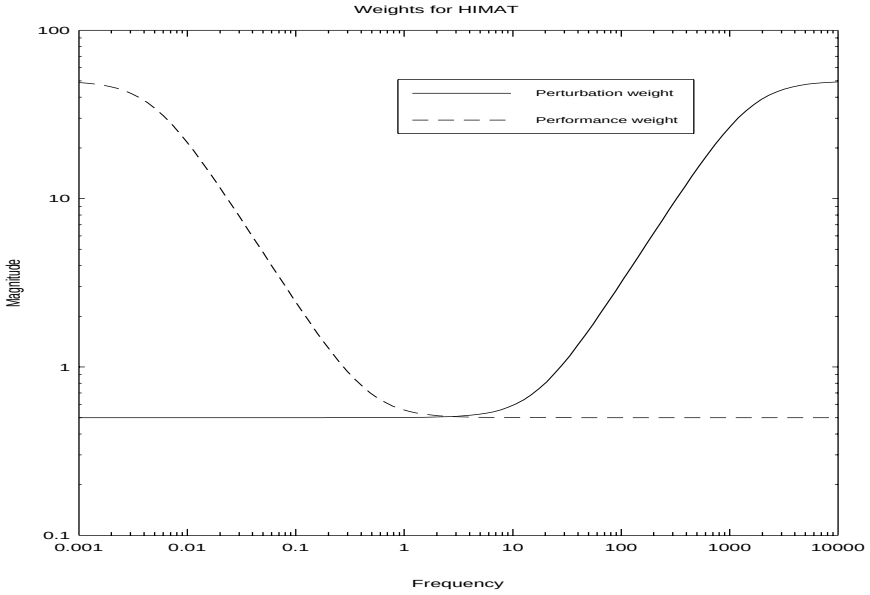
$$W_{\text{p}} = \frac{0.5(s+3)}{(s+0.03)}.$$

This will be used as the performance weight. The appropriate commands are:

```
wdel = makepoly([50,5000])/makepoly([1,10000])

wp = makepoly([0.5,1.5])/makepoly([1,0.03])
```

These can be displayed on a frequency response plot for comparison purposes.

```
om1 = logspace(0.001,10000,100)    # frequency vector (Hz)
wdelg = freq(wdel,om1)
comment wdelg "frequency response of wdel"

wpg = freq(wp,om1)
comment wpg "frequency response of wp"

gph1 = ctrlplot([wdelg,wpg],{logmagplot});
gph1 = plot(gph1,{title="Weights for HIMAT",!grid,...
         legend=["Perturbation weight";"Performance weight"]})?
```



The perturbation weight, W_{del}, should actually be two-input, two-output. This is also true of the performance weight, W_{p}. In this example, we are weighting each performance channel identically. There is no requirement to do this and if we were more concerned with errors in α than errors in θ, then the appropriate channel would have a large weight applied. The function `daug` is used to make the 2×2 systems.

```
wdel = daug(wdel,wdel)
comment wdel "perturbation weight"
wp = daug(wp,wp)
comment wp "performance weight"
```

The design interconnection structure is now created with the `sysic` function. The result is a state space system (8 states, 6 inputs and 6 outputs) called `himat_ic`.

The first two inputs and outputs correspond to the multiplicative perturbation block. Inputs and outputs 3 and 4 are the disturbance inputs & error outputs. The measurements going to the controller appear on outputs 5 & 6. The control actuation is put into the system on inputs 5 & 6.

```
sysn = ["himat";"wdel";"wp"]
in = ["pert(2)";"dist(2)";"control(2)"]
out = ["wdel";"wp";"himat + dist"]
inter = ["control + pert"; "control"; "himat + dist"]

himat_ic = sysic(sysn,in,out,inter,himat,wdel,wp)

comment himat_ic "Himat design interconnection structure"
```

4.1.4 H_∞ Design

The next step is to design an H_∞ control law for Himat. The function `hinfsyn` designs an H_∞ control law based on the interconnection structure provided. `hinfsyn` requires the design interconnection structure, number of measurements, number of controls and single gamma or bisection bounds on γ.

Optional input arguments are the tolerance for terminating the γ iteration (`tol`), an epsilon for Hamiltonian $j\omega$-axis eigenvalues (`epr`) and the epsilon for the Riccati solution positive definite tests (`epp`). Two Riccati solution methods are provided: eigenvalue or Schur (default) decomposition. `hinfsyn` returns the control law, `k` and the gamma value achieved, `gf1`.

In this example, the system interconnection structure is `himat_ic`, with 2 measurements, 2 controls, a γ lower bisection bound of 0.8, a γ upper bisection bound of 6, a tolerance on the γ iteration of 0.05, and we'll use the eigenvalue decomposition method to solve the Riccati equations. The default values of `epr` (`0.5*sqrt(eps)`) and `epp` (1e-6) will be used for the epsilon tests.

```
gamma_bounds = [0.8;6.0]
nmeas = 2       # 2 measurements:  attack angle & pitch
nctrls = 2      # 2 controls:  elevon & canard

comment nmeas "number of controller measurements"
comment nctrls "number of controller outputs"

[k1,gf1] = hinfsyn(himat_ic,nmeas,nctrls,gamma_bounds,{tol=0.05})

Test bounds:      0.8000  <  gamma  <=      6.0000
```

```
  gamma      Hx_eig     X_eig    Hy_eig      Y_eig   nrho_xy   p/f
 6.000     2.3e-02   5.6e-05   2.3e-02   0.0e+00   0.0626    p
 3.400     2.3e-02   5.7e-05   2.3e-02   0.0e+00   0.2020    p
 2.100     2.3e-02   5.9e-05   2.3e-02   0.0e+00   0.5798    p
 1.450     2.3e-02   6.4e-05   2.3e-02   0.0e+00   1.4678    f
 1.775     2.3e-02   6.1e-05   2.3e-02   0.0e+00   0.8652    p
 1.613     2.3e-02   6.2e-05   2.3e-02   0.0e+00   1.1028    f
 1.694     2.3e-02   6.1e-05   2.3e-02   0.0e+00   0.9725    p
 1.653     2.3e-02   6.2e-05   2.3e-02   0.0e+00   1.0343    f

 Gamma value achieved:     1.6938

comment k1 "controller:  iteration 1"
comment gf1 "gamma value:  iteration 1"

g1 = starp(himat_ic,k1)
comment g1 "closed loop:  iteration 1"
```

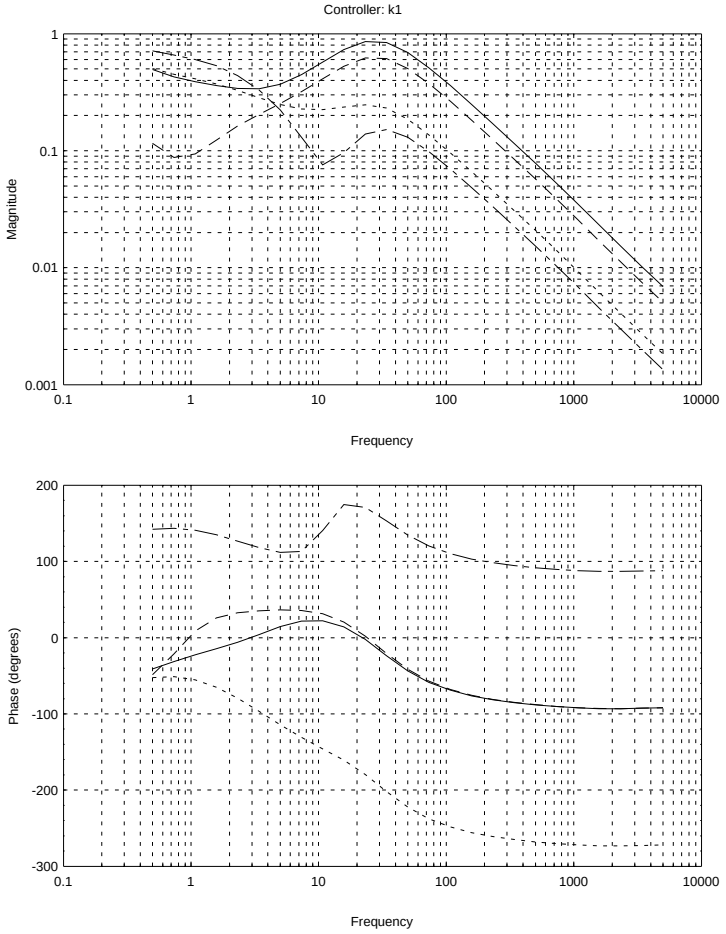
An H_∞ control law has been designed which achieves an infinity norm of 1.6938 for the interconnection structure provided. First, we will examine aspects of the controller that was just designed, starting with the controller poles.

```
rifd(k1)

Poles:

      real        imaginary      frequency      damping
                                 (rad/sec)       ratio

  -2.2609e-02     0.0000e+00     2.2609e-02     1.0000
  -3.0000e-02     0.0000e+00     3.0000e-02     1.0000
  -3.0000e-02     0.0000e+00     3.0000e-02     1.0000
  -1.3833e+01     0.0000e+00     1.3833e+01     1.0000
  -9.8959e+01     0.0000e+00     9.8959e+01     1.0000
  -1.4712e+02     9.6846e+01     1.7613e+02     0.8353
  -1.4712e+02    -9.6846e+01     1.7613e+02     0.8353
  -7.4256e+03     0.0000e+00     7.4256e+03     1.0000
```

```
Zeros:

      real        imaginary      frequency      damping
                                 (rad/sec)       ratio

  -2.2516e-02     0.0000e+00     2.2516e-02     1.0000
  -1.7226e+00     0.0000e+00     1.7226e+00     1.0000
  -3.0272e+00     0.0000e+00     3.0272e+00     1.0000
  -3.1034e+01     0.0000e+00     3.1034e+01     1.0000
  -1.0000e+04     0.0000e+00     1.0000e+04     1.0000
  -1.0000e+04     0.0000e+00     1.0000e+04     1.0000
```

Next, a magnitude plot of the frequency response of `k1` is plotted to check that it looks reasonable.

```
om2 = logspace(0.5,5000,25)
k1_g = freq(k1,om2)

comment k1_g "frequency response of k1"
comment om2 "frequency vector (Hz)"

gph2 = ctrlplot(k1_g,{bode});
gph2 = plot(gph2,{title="Controller:  k1"})?
```



Onto the closed loop, first checking that it is stable by looking at the pole positions.

```
rifd(g1)

Poles:

      real         imaginary      frequency      damping
                                  (rad/sec)       ratio
```

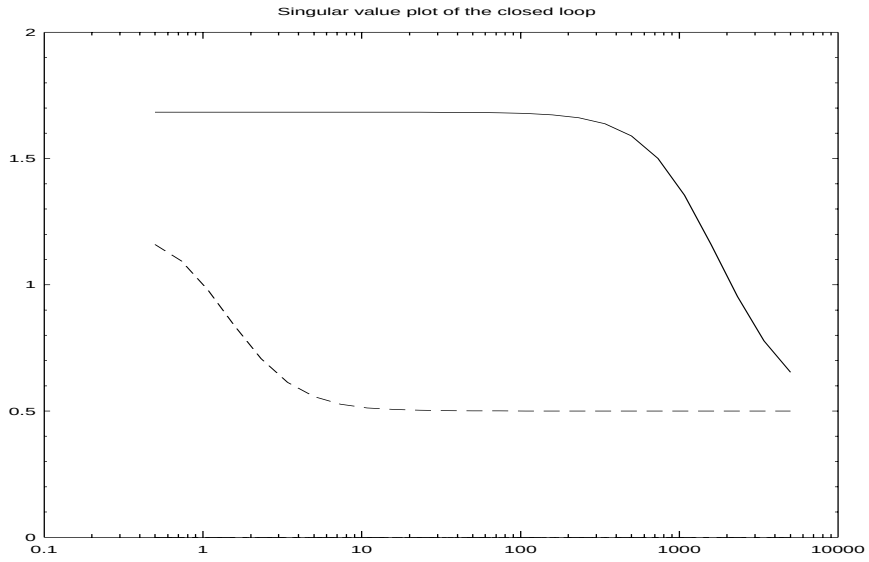
```
  -2.2517e-02     0.0000e+00     2.2517e-02     1.0000
  -2.2600e-02     0.0000e+00     2.2600e-02     1.0000
  -3.0000e-02     0.0000e+00     3.0000e-02     1.0000
  -3.0000e-02     0.0000e+00     3.0000e-02     1.0000
  -2.9369e+00     0.0000e+00     2.9369e+00     1.0000
  -2.9974e+00     0.0000e+00     2.9974e+00     1.0000
  -4.8310e+00     0.0000e+00     4.8310e+00     1.0000
  -6.5876e+00     0.0000e+00     6.5876e+00     1.0000
  -5.8350e+01    -5.6049e+01     8.0909e+01     0.7212
  -5.8350e+01     5.6049e+01     8.0909e+01     0.7212
  -8.8792e+01    -4.2881e+01     9.8604e+01     0.9005
  -8.8792e+01     4.2881e+01     9.8604e+01     0.9005
  -9.9778e+01     0.0000e+00     9.9778e+01     1.0000
  -7.4258e+03     0.0000e+00     7.4258e+03     1.0000
  -1.0000e+04     0.0000e+00     1.0000e+04     1.0000
  -1.0000e+04     0.0000e+00     1.0000e+04     1.0000


Zeros:

ans (a scalar) =   0
```

Now calculate a closed loop frequency response. This results in a 4x4 PDM: `g1g`

```
g1g=freq(g1,om2)
comment g1g "frequency response of g1"
```

The singular values of the closed loop system are calculated. The maximum of the singular values over frequency is also calculated and compared to γ. As γ is a guaranteed upper bound on the infinity norm, this value should be less.

```
g1gs = svd(g1g)
comment g1gs "closed loop singular value:  iteration 1"
gph3 = ctrlplot(g1gs,{log});
gph3 = plot(gph3,{!grid,...
            title="Singular value plot of the closed loop"})?
```



4.1.5 μ Analysis of the H_∞ Controller

The H_∞ control law can be analyzed using μ-analysis. The closed-loop system, `g1`, has 4 inputs and 4 outputs. The first two inputs and outputs correspond to the uncertainty block, and the second two correspond to the disturbance rejection block, or performance block. Therefore, we can define the uncertainty inputs and outputs as a full 2×2 uncertainty block and the disturbance rejection inputs and outputs as a full 2×2 performance block. If the μ value for this control design is 1, then we are able to achieve robust performance for the set of defined weights and this control design.

The `mu` function analyzes the robust performance and stability of the closed loop system. The syntax of the function is:

```
[bnds,D,Dinv,Delta,sens] = mu(M,blk)
```

The variable `M` is usually the frequency response of the closed loop system. `blk` defines the structure of the perturbations. In this example the block structure is two 2×2

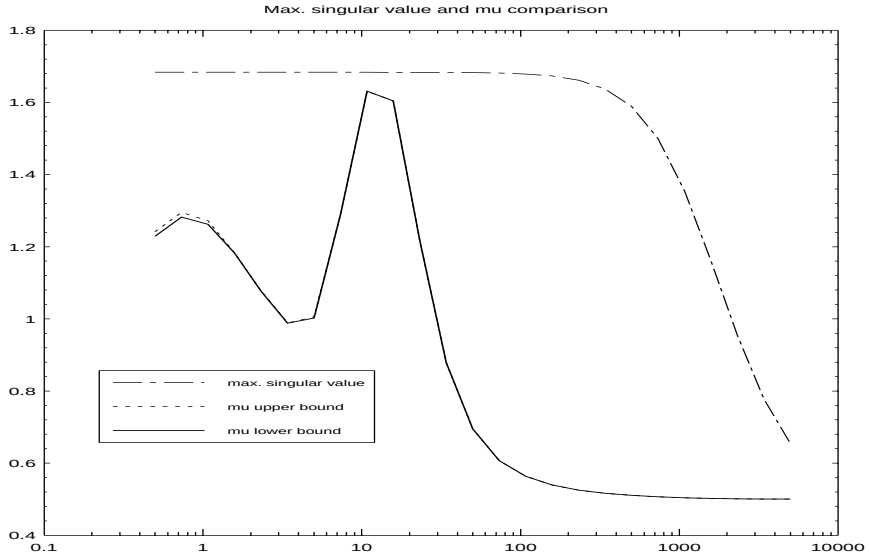
complex valued blocks.

The upper and lower bounds of the μ function are returned in `bnds`. Also returned are the scaling matrices, `D` and `Dinv`, corresponding to the upper bound. The smallest destabilizing perturbation at each frequency is returned as `Delta`. The variable `sens` is the sensitivity of the upper bound to the `D` and `Dinv` scaling matrices. This will be useful as a weighting function for fitting transfer functions to the `D` and `Dinv` scaling

```
blk = [2,2;2,2]
[bnds1,D1,D1inv,Delta1,sens1] = mu(g1g,blk)

comment blk "perturbation block structure"
comment bnds1 "mu bounds:  iteration 1"
comment D1 "D scale:  iteration 1"
comment D1inv "D inverse scale:  iteration 1"
comment Delta1 "worst case perturbation:  iteration 1"
comment sens1 "D scale sensitivity:  iteration 1"
```

We plot the maximum singular value and μ on the same plot. The performance and stability specifications have been achieved if μ is less than one at all frequencies.

```
gph4 = ctrlplot(g1gs(1,1),{log,line_style=4});
gph4 = ctrlplot(bnds1,gph4,{log,line_style=[1,3]});
gph4 = plot(gph4,{title="Max.  singular value and mu comparison",...
            legend=["max.  singular value";"mu upper bound";...
            "mu lower bound"],!grid})?
```



Note that μ(`g1g`) is not less than one at all frequencies — we have not met the design objectives. A D-K iteration will be used to lower μ and improve the robust performance with respect to these objectives.

4.1.6 Fitting D-scales for the D-K Iteration

In some cases the H_∞ controller is adequate for our purposes. It is not necessarily the controller which gives the best robust performance for our system — it essentially ignores the structure in the perturbations.

The D-K iteration procedure using the `D` and `Dinv` scaling matrices in the μ calculation to set up an H_∞ problem which will usually give better robust performance.

The first step is to pre and post multiply the interconnection, (`himat_ic` in this case) with `D` and `Dinv`. Two things need to be done first. The `D` and `Dinv` matrices produced by μ are PDMs and the interconnection is a state-space system. We must first fit transfer functions to the `D` and `Dinv` magnitude data before we can do the multiplication.

The second thing to note is that the interconnection structure has the additional control inputs and measurement outputs. The `D` and `Dinv` systems must be augmented with identities corresponding to these additional inputs and outputs.

The musynfit function performs both of these operations. The syntax of musynfit is:

```
[Dsys,Dinvsys] = musynfit(D,blk,nmeas,nctrls,sens,oldDsys)
```

The variables `D` and `sens` come directly from the `mu` function. The block structure is specified by `blk` and `nmeas` and `nctrls` are the number of measurements and controls respectively.

The outputs are the dynamic systems which approximate `D` and `Dinv`. The new scaled H_∞ problem can be set up with

```
new_ic = Dsys * old_ic * Dinvsys.
```

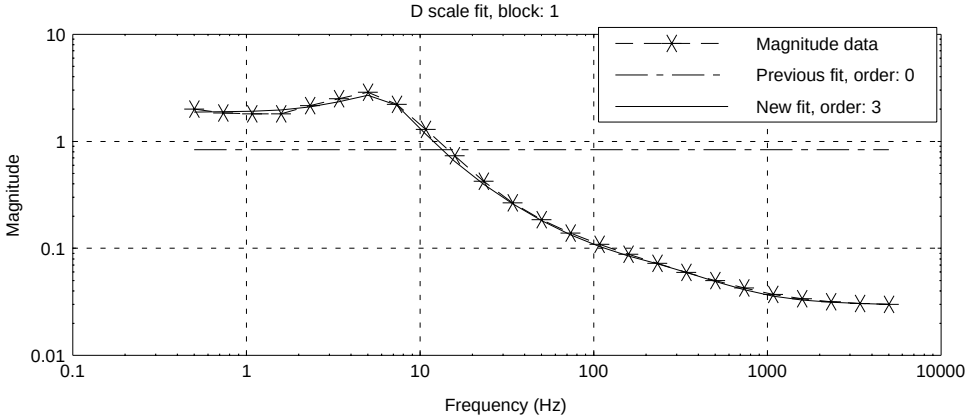
D-K iteration involves iterating between calculating and fitting D-scales and designing controllers, K.

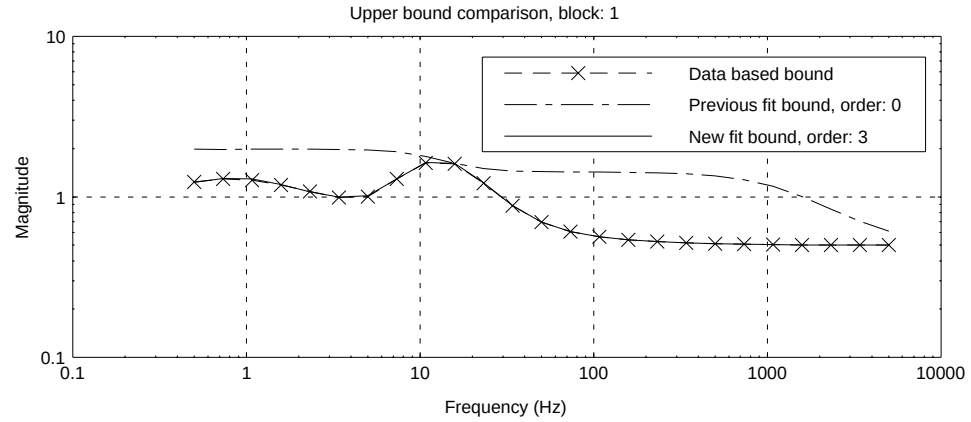
If there are N blocks in the μ problem set up, then the D scale matrices have $N-1$ different transfer functions that require fitting. The Nth transfer function is taken to be unity.

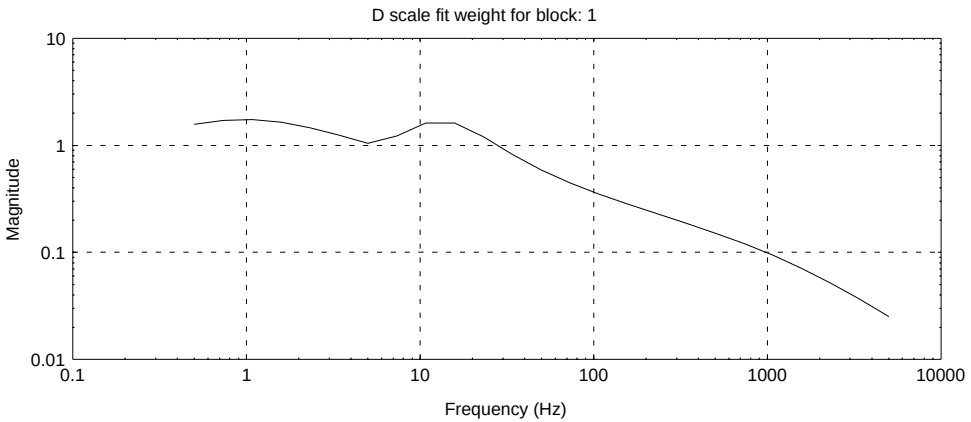
We recommend choosing a 3rd order transfer function for the fit. This increases the number of states in the interconnection structure by 3*(size of block)*2. A different order can be chosen — which will lead to a slightly different controller in the subsequent analysis.

Note that `g1g` is also passed to `musynfit`. This will provide the user with a comparison between the calculated upperbound and that based on the rational fit. This comparison is useful in deciding between fits of differing orders.

```
[D1sys,D1invsys] = musynfit(D1,blk,nmeas,nctrls,sens1,g1g,{Hertz})
```


D scale fit, block: 1
Magnitude data
Previous fit, order: 0
New fit, order: 3
Magnitude
Frequency (Hz)
10
1
0.1
0.01
0.1
1
10
100
1000
10000


Upper bound comparison, block: 1
Data based bound
Previous fit bound, order: 0
New fit bound, order: 3
Magnitude
Frequency (Hz)
10
1
0.1
0.1
1
10
100
1000
10000


D scale fit weight for block: 1
Magnitude
Frequency (Hz)
10
1
0.1
0.01
0.1
1
10
100
1000
10000

```
comment D1sys "system approx.  to D1"
comment D1invsys "system approx.  to D1inv"
```

4.1.7 Design Iteration #2

The new D scales can be pre and post multiplied onto the orginal interconnection structure.

```
himat_ic2 = D1sys * himat_ic * D1invsys
[,,nx] = size(himat_ic2)
display "himat_ic2 now has " + string(nx) + " states"
himat_ic2 now has 20 states

comment himat_ic2 "interconnection for iteration 2"
```

Note the increase in states due to the inclusion of the D scales. A new H_∞ controller can now be designed.

```
gamma_bounds = [0.9,6.0]
[k2,g2,gf2] = hinfsyn(himat_ic2,nmeas,nctrls,...
                      gamma_bounds,{tol=0.05})
Test bounds:      0.9000  <  gamma  <=      1.7000
  gamma      Hx_eig     X_eig    Hy_eig      Y_eig   nrho_xy   p/f
   1.700     2.3e-02  -1.1e-09   2.2e-02  -2.5e-27   0.3120    p
   1.300     2.3e-02  -3.1e-12   2.2e-02  -1.4e-29   0.6390    p
   1.100     2.3e-02  -1.2e-08   2.2e-02  -3.4e-22   1.1028    f
   1.200     2.3e-02   2.2e-14   2.2e-02  -2.1e-15   0.8180    p
   1.150     2.3e-02  -7.7e-10   2.2e-02  -1.1e-15   0.9423    p
   1.125     2.3e-02  -1.9e-08   2.2e-02  -5.8e-31   1.0171    f

Gamma value achieved:      1.1500


comment k2 "controller:  iteration 2"
comment gf2 "gamma value:  iteration 2"
```

Note that we calculate the new closed loop using the original interconnection: himat_ic.

```
g2 = starp(himat_ic,k2)
comment g2 "closed loop:  iteration 2"

[,,nx] = size(k2)
display "k2 now also has " + string(nx) + " states"
k2 now also has 20 states
```

This design probably resulted in a control law which achieved an infinity norm of approximately 1.1 for the new interconnection structure.

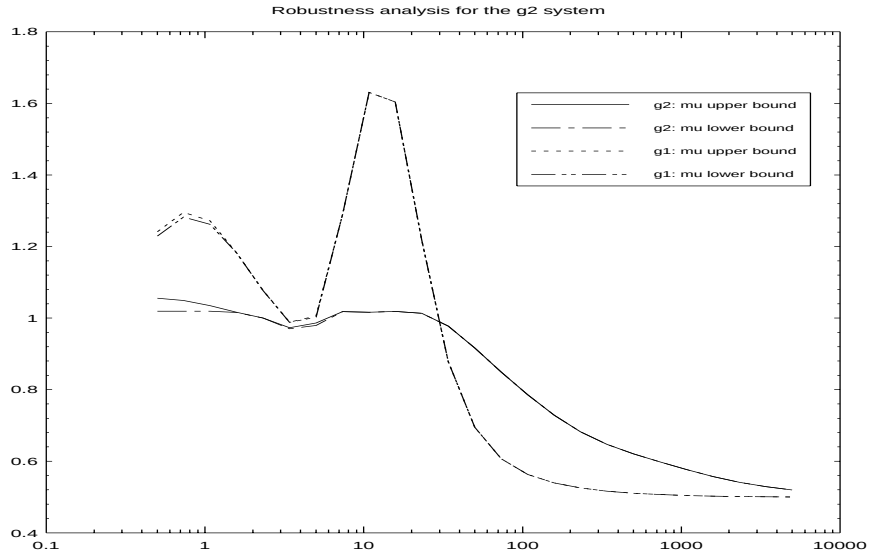
`k2` has been designed to reduce $\mu(g2)$. The singular values may actually get worse. We will see that this is the case here.

The stability of `g2` is examined with the function `check`. As used here, it will return a 1 if it is stable and a zero if not. A frequency response is then calculated.

```
check(g2,{stable})
ans (a scalar) = 1

g2g=freq(g2,om2)
comment g2g "g2 frequency response"
```

The μ analysis is repeated to assess the closed loop robust performance of the μ based controller (`k2`). The command format is identical to the last time.

```
[bnds2,D2,D2inv,Delta2,sens2] = mu(g2g,blk)

gph6 = ctrlplot(bnds2,{log,line_style=[1,4]});
gph6 = ctrlplot(bnds1,gph6,{log,line_style=[3,5]});
gph6 = plot(gph6,{title="Robustness analysis for the g2 system",...
            legend=["g2:  mu upper bound";"g2:  mu lower bound";...
            "g1:  mu upper bound";"g1:  mu lower bound"],!grid})?
```




```
comment bnds2 "mu bounds:  iteration 2"
comment D2 "D scale:  iteration 2"
comment D2inv "D inverse scale:  iteration 2"
comment Delta2 "worst case perturbation:  iteration 2"
comment sens2 "D scale sensitivity:  iteration 2"
```

Here we have only done a single D-K iteration. We are now much closer to the specifications although we still do not quite meet them. In practice several more iterations could be run to further improve the μ design. The next iteration would use the commands:

```
[D2sys,D2invsys] = musynfit(D2,blk,nmeas,nctrls,sens2,g2g)
himat_ic3 = D2sys*himat_ic*D2invsys
k3 = hinfsyn(himat_ic3,,nmeas,nctrls,gamma_bounds,tol=0.05)
```

and to analyze it we would use the commands:

```
g3 = starp(himat_ic,k3)
g3g = freq(g3,omega)
[bnds3,D3,D3inv,Delta3,sens3] = mu(g3g,blk)
```

At this point we could do another iteration (to get `k4`) or perhaps run some simulations to check out `k3` more thoroughly.

4.1.8 Simulation Comparison with a Loopshaping Controller

A loopshaping design is performed and compared to the H_∞ and μ designs. As the loopshaping design procedure is relatively standard, only the final controller is given here (`klp`).

```
a =[-5.8928e-02,-6.3295e+00,-1.0440e+00,...
               -1.6190e-02, 1.8469e+00, 1.0405e-02;...
    -1.0283e+00,-1.5776e+03,-1.1900e+03,...
               -9.9924e+00, 1.7895e+01, 3.0329e+00;...
     1.0440e+00, 1.0146e+03,-4.2005e+01,...
               -1.0953e+00, 2.1755e+02, 9.9809e-01;...
     1.3287e-02, 9.6897e+00,-5.9979e-01,...
               -2.5448e-02, 1.3634e+01, 4.5286e-02;...
     6.4800e+00, 4.0284e+01,-5.9317e+02,...
               -1.5215e+01,-2.3726e+04,-8.1459e+01;...
     1.3057e-02,-4.0389e+00,-1.2179e+00,...
               -1.4332e-02,-5.3180e+01,-2.3590e-01]

b =[1.2425e+00, -6.0707e-02; ...
    1.5602e+01, -6.9151e+01; ...
   -1.0002e+01,  5.5826e-01; ...
   -1.2752e-01,  1.1238e-01; ...
   -6.9355e+01, -1.4693e+01; ...
   -1.4459e-01 ,-1.2232e-01]

c = [3.5741e-01,7.9890e-01,2.8247e+00,...
              8.7133e-02,-7.0893e+01,-1.7902e-01;...
    1.1915e+00,7.0885e+01,9.6107e+00,...
              1.4594e-01, 3.3669e-01,-6.1799e-02]
```

```
d = zeros(2,2)
klp = system(a,b,c,d)
comment klp "loop shape controller"
```

We will compare the designs, with no error or uncertainty weights, for the nominal case and with a perturbation block of Δ = [0.1,0;0,-0.1] for the input multiplicative perturbation.

The time response will be from 0 to 2 seconds with a sample time of 0.01 seconds. We'll look at a unit step input into the first channel.

The unweighted closed loop system is now formed for each controller. The inputs and outputs to the perturbations are closed around a $\Delta = 0$ perturbation initially. This is equivalent to selecting the nominal inputs and outputs.

```
sysnames = ["himat";"wdel"]
invars = ["pert(2)"; "dist(2)"; "control(2)"]
outvars = ["wdel"; "himat+dist"; "himat+dist"]
connections = ["control + pert";"control"]
gsim = sysic(sysnames,invars,outvars,connections,himat,wdel)

comment gsim "unweighted interconnection"

gsim_mu = starp(gsim,k2)
gsim_hinf = starp(gsim,k1)
gsim_lp = starp(gsim,klp)

comment gsim_mu "closed loop system:  mu ctrl"
comment gsim_hinf "closed loop system:  hinf ctrl"
comment gsim_lp "closed loop system:  klp ctrl"
```

The nominal systems are studied first.

```
gsim_mu_nom = gsim_mu(3:4,3:4)
gsim_hinf_nom = gsim_hinf(3:4,3:4)
gsim_lp_nom = gsim_lp(3:4,3:4)
```

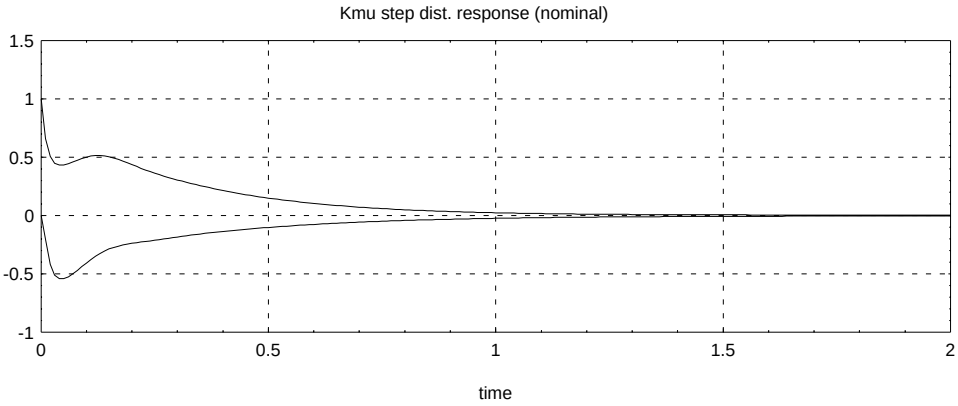
```
comment gsim_mu_nom "nominal closed loop sys:  mu ctrl"
comment gsim_hinf_nom "nominal closed loop sys:  hinf ctrl"
comment gsim_lp_nom "nominal closed loop sys:  klp ctrl"
```

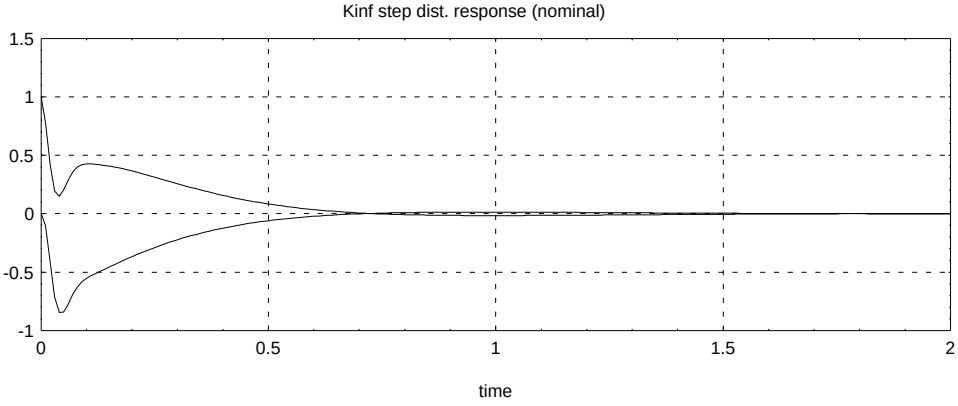
A step disturbance is introduced into the first channel.

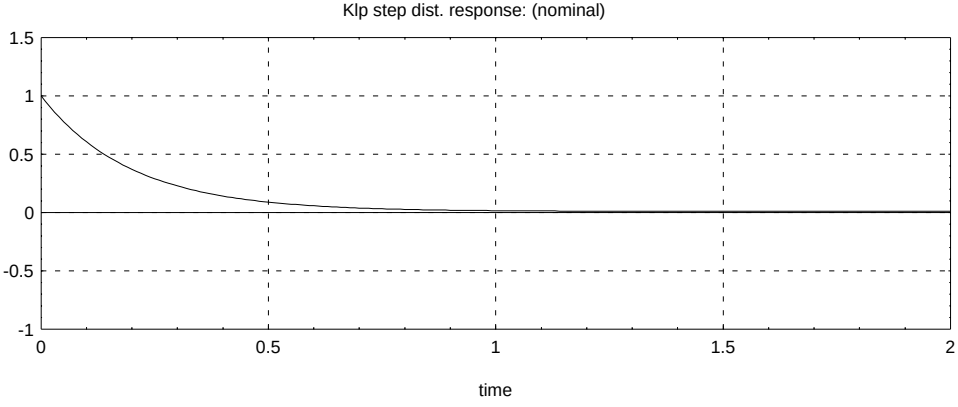
```
time = 0:2:0.01
u = gstep(time)         # a unit step is the default
u = [u;0*u]             # put zero into the other channel

comment u "simulation input"

y_mu_nom = gsim_mu_nom*u
y_hinf_nom = gsim_hinf_nom*u
y_lp_nom = gsim_lp_nom*u

comment y_mu_nom "nominal response:  mu ctrl"
comment y_hinf_nom "nominal response:  hinf ctrl"
comment y_lp_nom "nominal response:  klp ctrl"
```

All controllers perform well on the nominal system. This can be seen by plotting the time histories of each.

```
gph7 = plot(y_mu_nom,{rows=3,row=1,grid,...
                title="Kmu step dist.  response (nominal)",...
                x_lab="time",y_max=1.5,y_min=-1});
gph7 = plot(y_hinf_nom,gph7,{row=2,grid,...
                title="Kinf step dist.  response (nominal)",...
                x_lab="time",y_max=1.5,y_min=-1});
gph7 = plot(y_lp_nom,gph7,{row=3,grid,...
                title="Klp step dist.  response:  (nominal)",...
                x_lab="time",y_max=1.5,y_min=-1})?
```


Kmu step dist. response (nominal)
1.5
1
0.5
0
-0.5
-1
0
0.5
1
1.5
2
time


Kinf step dist. response (nominal)
1.5
1
0.5
0
-0.5
-1
0
0.5
1
1.5
2
time


Klp step dist. response: (nominal)
1.5
1
0.5
0
-0.5
-1
0
0.5
1
1.5
2
time

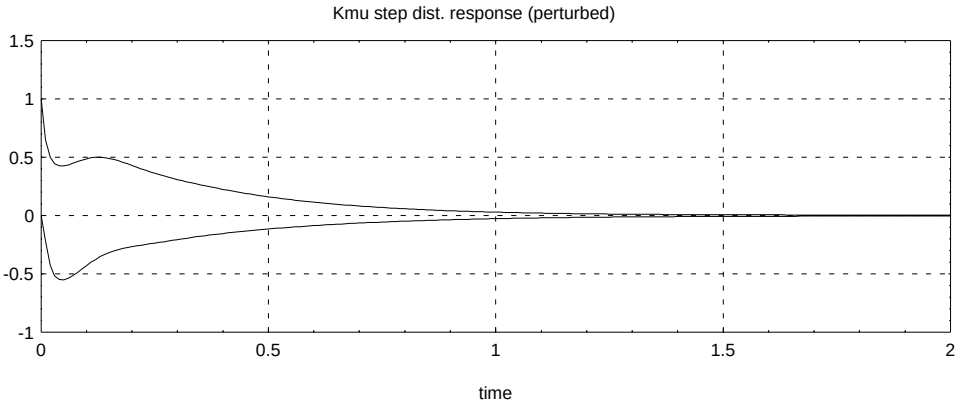
The loopshaping design gives a decoupled response. Both the $\mathcal{H}_\infty$ and μ designs trade decoupling for speed of response and, as we shall see, robustness with respect to perturbations.

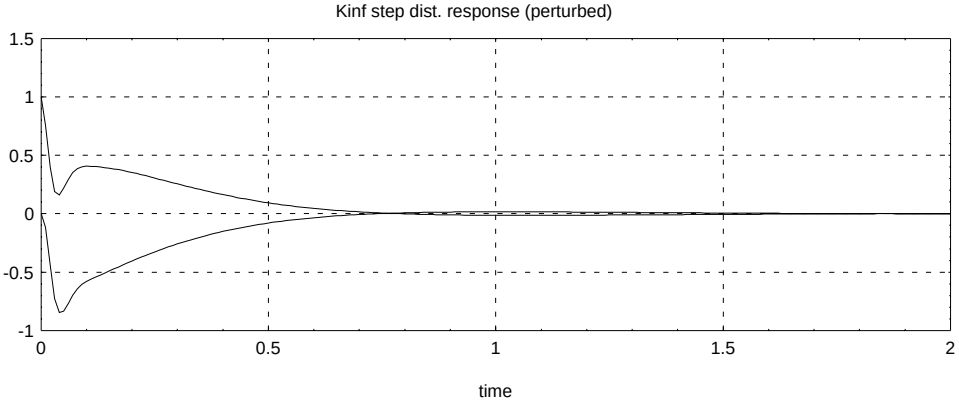
The simulation is repeated with a perturbation of size 0.1. Note that this is only 10% of the size perturbation that we were analyzing and designing for in the above.

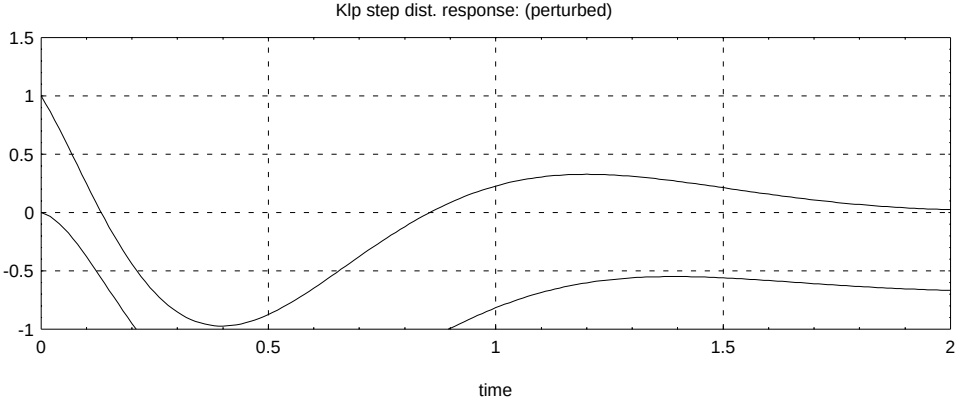
```
delta = [.1,0;0,-0.1]

comment delta "example perturbation"

gsim_mu_pert = starp(delta,gsim_mu)
gsim_hinf_pert = starp(delta,gsim_hinf)
gsim_lp_pert = starp(delta,gsim_lp)

comment gsim_mu_pert "perturbed closed loop sys:  mu ctrl"
comment gsim_hinf_pert "perturbed closed loop sys:  hinf ctrl"
comment gsim_lp_pert "perturbed closed loop sys:  klp ctrl"

y_mu_pert = gsim_mu_pert*u
y_hinf_pert = gsim_hinf_pert*u
y_lp_pert = gsim_lp_pert*u

comment y_mu_pert "perturbed response:  mu ctrl"
comment y_hinf_pert "perturbed response:  hinf ctrl"
comment y_lp_pert "perturbed response:  klp ctrl"
```

The loop shaping controller performs poorly on the perturbed system. Again the time response of each system is plotted.

```
gph8 = plot(y_mu_pert,{rows=3,row=1,grid,...
                title="Kmu step dist.  response (perturbed)",...
                x_lab="time",y_max=1.5,y_min=-1});
gph8 = plot(y_hinf_pert,gph8,{row=2,grid,...
                title="Kinf step dist.  response (perturbed)",...
                x_lab="time",y_max=1.5,y_min=-1});
gph8 = plot(y_lp_pert,gph8,{row=3,grid,...
                title="Klp step dist.  response:  (perturbed)",...
                x_lab="time",y_max=1.5,y_min=-1})?
```


Kmu step dist. response (perturbed)
1.5
1
0.5
0
-0.5
-1
0
0.5
1
1.5
2
time


Kinf step dist. response (perturbed)
1.5
1
0.5
0
-0.5
-1
0
0.5
1
1.5
2
time


Klp step dist. response: (perturbed)
1.5
1
0.5
0
-0.5
-1
0
0.5
1
1.5
2
time

The loopshaping controller had good nominal performance and very poor robust performance. This was illustrated with a relatively small perturbation. The difference between the μ and H_∞ controllers was small in both the nominal and perturbed cases. This may not always be the case for several reasons.

Only a single D-K iteration was performed here. Further iterations would further improve the performance of the μ controller.

The perturbation chosen for the above simulation was not the worst case one. Note that here, each of the three closed loop systems will have a different worst case perturbation. To find these, perform a μ calculation on each closed loop system and use `mkpert` to construct the appropriate perturbations.

The theoretical measure of performance is the H_∞ norm of the closed loop transfer function. When assessing different controllers by simulation we are applying additional, unformalized performance measures.

In this case, the performance and perturbation channels were about equally scaled. The resulting D-scales were within an order of magnitude of unity. Choosing a different set of units for α and/or θ would change the H_∞ norm of the result without changing μ. A poor choice of engineering units could therefore lead to a larger difference between the H_∞ controller and the μ controller.

4.2 A Simple Flexible Structure Example

The demonstration script, `jplphBdemo.ms`, runs through a D-K iteration design for a simple flexible structure problem.

The following demo can be run by executing the following Xmath command:

```
execute file = "$XMATH/demos/xMu/jplphBdemo"
```

where $XMATH is the path to your Xmath source location.

The problem comes from an experiment in the NASA Control Structures Interaction (CSI) program and is located at the Jet Propulsion Laboratory. A more complete description is given by Spanos *et al.* [85, 86, 87].

The problem involves controlling the length of a laser path which reflects off of a series of mirrors mounted on a flexible structure. One measurement is available to the controller: the pathlength. There are two actuators, a voice coil and a piezo-electric, each driving a mirror in the path. These actuators are effectively in parallel at the same point. They differ significantly in their characteristics and uncertainty descriptions.

This is a relatively simple problem and the user should be able to achieve similar results through standard classical techniques. The code given here serves as a suitable template for user written design scripts. It should be noted that a large number of frequency points are used in this design — perhaps more than necessary — and this significantly slows down the calculation of μ.

4.2.1 The Control Design Problem

The optical configuration is illustrated schematically in Figure 4.3. The laser system consists of a laser and optical interferometer. It is mounted on a fixed optical bench. The laser is directed to the actuated mirrors mounted on the flexible structure. The path continues back to the optical bench where it hits a target mirror and reflects back through the structure mirrors to the interferometer. The interferometer gives a measurement of the optical pathlength to an accuracy of 2.5 nm.

Vibrations in the flexible structure affect the optical pathlength and the objective is to maintain a constant pathlength in the presence of such vibrations. Complicating the

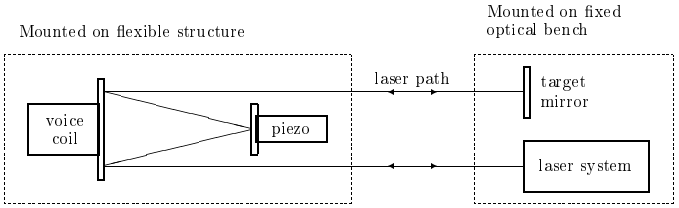


Figure 4.3: Schematic diagram of the JPL Phase B optical testbed design problem

problem is the fact that the voice-coil mirror assembly has significant mass and its movement excites a mode in the structure. An identically driven counterbalance effectively makes the piezo-electric actuator reactionless.

The closed loop design problem is illustrated in Figure 4.4.

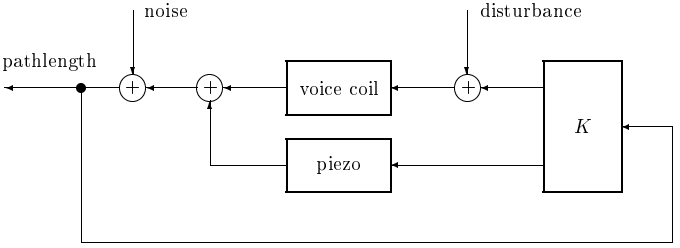


Figure 4.4: JPL Phase B closed loop optical control configuration

We now construct a model of the voice-coil mirror actuator, `vcmodel`. This model has been obtained from identification experiments. Note that it has an oscillatory pole pair at a frequency very close to an oscillatory zero pair.

```
vcmodel1 = makepoly([1,.056309,1162.8],"s")/...
                  makepoly([1,.78756,1189.1],"s")
vcmodel2 = -252861.0/makepoly([1,1.0936,19.673],"s")
vcmodel = vcmodel1*vcmodel2
delete vcmodel1 vcmodel2
size(vcmodel)?
ans (a row vector) =   1    1    4
```

In the frequency range of interest the piezo mirror driver can nominally be modeled as a

static gain. We will include some dynamic uncertainty in the actual design.

```
piezo = 1
```

4.2.2 Creating the Weighted Design Interconnection Structure

The weighted, open-loop design interconnection structure is illustrated in Figure 4.5. For clarity, the two perturbations, Δ_1 and Δ_2, have been shown inside the structure. We have included perturbation models for both the actuators, involving the weights `Wavoice`, `Wmvoice` and `Wmpiezo`.

In addition there is an error performance weight, `Wperf`, and two actuator penalties, `Wactv` and `Wactp`. The relative sizes of the noise and disturbance inputs are specified by the weights `Wnoise` and `Wdist` respectively.

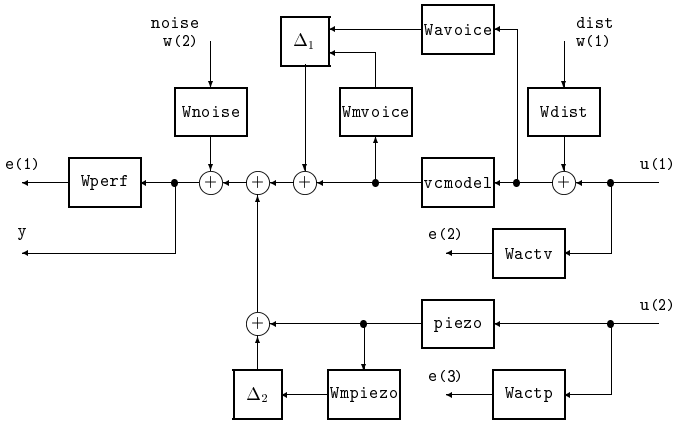


Figure 4.5: Weighted open-loop interconnection structure for the JPL Phase B optical design problem

The voice coil additive weight, `Wavoice`, is simply a constant. Because the nominal voice coil model rolls off at 40db/decade this weight indicates significant gain and phase uncertainty for the frequencies where $|$`Wavoice`$| > |$`vcmodel`$|$.

```
Wavoice = 1
```

The additive weight clearly provides for significant high frequency uncertainty. A multiplicative weight models the low frequency uncertainty. The value selected is somewhat arbitrary and can be considered as a tunable design weight.

```
Wmvoice = 0.1
```

A multiplicative perturbation is used to model uncertainty in the piezo driver. Experimental observations suggest a 5% deviation from nominal across the frequency range of interest.

```
Wmpiezo = 0.05
```

W_{dist} is a weight for disturbances on the flexible structure. The magnitude of this has been estimated by comparing experimental disturbance responses to the voice coil system response. This weight can also be considered as a variable in the design problem. Increasing the weight will place more emphasis on disturbance rejection in the final design.

```
Wdist = 0.01
```

The noise weight, W_{noise} is selected based on what is necessary to achieve a final resolution of 10nm.

```
Wnoise = 0.005
```

The performance weight, W_{perf} is used to specify system performance up to 80 Hz. We use a 2nd order Butterworth filter to roll off at frequencies beyond 80 Hz.

```
Wperf = butterworth(2,{Fc=80,dT=0})
```

There are actuator penalties for the piezo driver, W_{actp}, and the voice coil driver, W_{actv}. The piezo seems to have usable bandwidth out to about 400 Hz. We use a first order roll up, starting at 100 Hz. to penalize higher frequency actuation. The piezo penalty is set at about 200 times greater than the voice coil penalty. This is because the piezo actuator will saturate at 30 micrometers and the voice coil saturates at 6 mm.

```
F = 100
Wactp = makepoly([1/(2*pi*F),1],"s")/...
                 makepoly([1/(200*pi*F),1],"s")
Wactp = Wactp*4
```

The lower frequency response of the voice-coil system means that the W_{actv} weight should begin rolling up at around 10 Hz.

```
F = 10
Wactv = makepoly([1/(2*pi*F),1],"s")/...
                makepoly([1/(200*pi*F),1],"s")
Wactv = Wactv*0.02
```

Both of these weights can be adjusted to trade between the relative responses from the voice-coil and the piezo. This should be done after examining simulations, or experimental closed loop data.

The weights are concatenated together for easier display.

```
weights = [Wperf;Wavoice;Wmvoice;Wmpiezo;Wdist;Wnoise;Wactp;Wactv]
```

We now construct the weighted design interconnection structure using `sysic`.

```
ssnames = ["vcmodel"; "Wavoice"; "Wmvoice"; "Wmpiezo";...
        "Wdist"; "Wnoise"; "Wperf"; "Wactp"; "Wactv"; "piezo"]

inps = ["d1i"; "d2i"; "dist"; "noise"; "vact"; "pact"]

ops = ["Wavoice";"Wmvoice";"Wmpiezo";"Wperf";"Wactv";"Wactp";...
       "d1i + vcmodel + d2i + piezo + Wnoise"]

cnx = ["Wdist + vact"; "Wdist + vact"; "vcmodel"; "piezo";...
       "dist"; "noise"; "d1i + vcmodel + d2i + piezo + Wnoise";...
       "pact"; "vact"; "pact"]

P = sysic(ssnames,inps,ops,cnx,vcmodel,Wavoice,Wmvoice,Wmpiezo,...
```

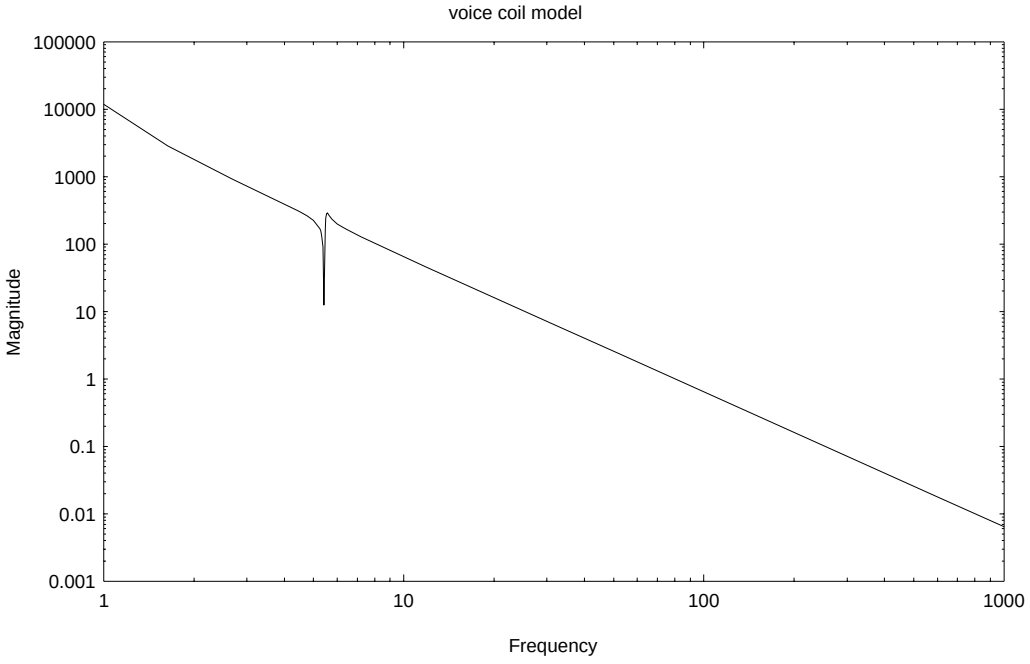
```
            Wdist,Wnoise,Wperf,Wactp,Wactv,piezo)
size(P)?
ans (a row vector) =   7    6    8
```

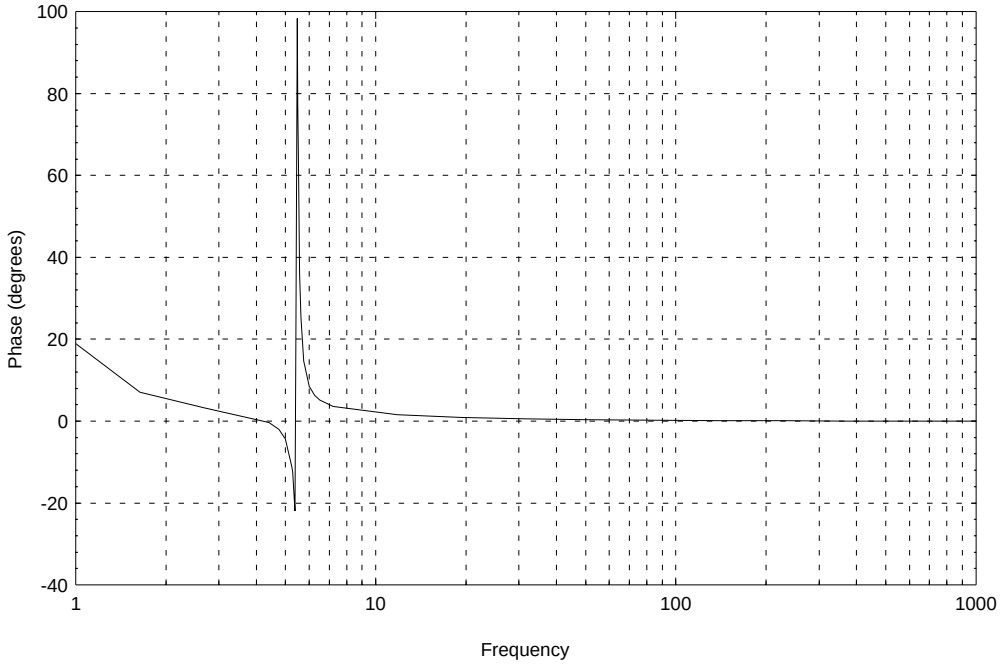
Now select a frequency grid for calculating the frequency responses. Some additional points are included near the oscillatory modes.

```
omega = logspace(1,1000,15)'
omega = sort([omega; [5.275:0.05:5.625]'; [4.5:0.25:6.5]'])'
```

Examine the frequency response of the open-loop system.

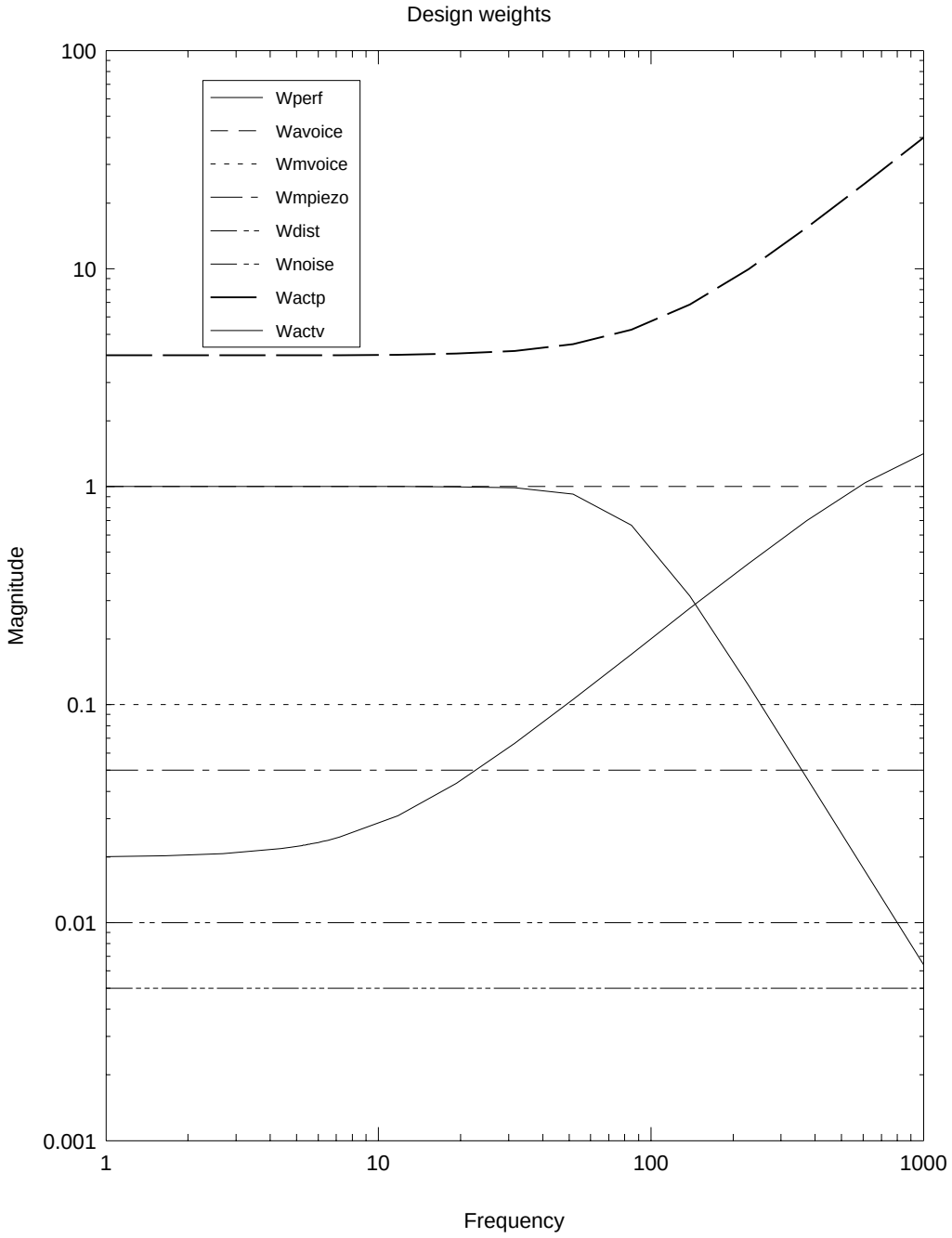
```
vcmodelg = freq(vcmodel,omega)
gph1 = ctrlplot(vcmodelg,bode);
gph1 = plot(gph1,{title="voice coil model",!grid})?
```


voice coil model
100000
10000
1000
100
10
1
0.1
0.01
0.001
Magnitude
1
10
100
1000
Frequency


100
80
60
40
20
0
-20
-40
Phase (degrees)
1
10
100
1000
Frequency

And examine the design weights.

```
weightsg = freq(weights,omega)
gph2 = ctrlplot(weightsg,{logmagplot});
gph2 = plot(gph2,legend=["Wperf";"Wavoice";"Wmvoice";...
           "Wmpiezo";"Wdist";"Wnoise";"Wactp";"Wactv"],...
           title="Design weights",!grid)?
```


Design weights
Wperf
Wavoice
Wmvoice
Wmpiezo
Wdist
Wnoise
Wactp
Wactv
100
10
1
0.1
0.01
0.001
Magnitude
1
10
100
1000
Frequency

4.2.3 Design of an $\mathcal{H}_\infty$ Controller

An $\mathcal{H}_\infty$ design is now performed. Recall that we have one interferometer measurement and two controller outputs.

```
nmeas = 1
ncon = 2
glimits = [0;20]
[Khinf,gamma] = hinfsyn(P,nmeas,ncon,glimits,{tol=0.25})
Test bounds:      0.0000  <  gamma  <=     20.0000
  gamma      Hx_eig     X_eig    Hy_eig      Y_eig   nrho_xy   p/f
   20.000     2.8e-02   3.1e-07   2.8e-01  -6.5e-17   0.0007    p
   10.000     2.8e-02   3.1e-07   2.8e-01  -4.5e-17   0.0026    p
    5.000     2.8e-02   3.1e-07   2.8e-01  -1.7e-16   0.0109    p
    2.500     2.8e-02   3.1e-07   2.8e-01   0.0e+00   0.0487    p
    1.250     2.8e-02   3.1e-07   2.8e-01  -1.6e-16   0.4059    p
    0.625     2.5e-07  *******    2.8e-01  -2.2e-16  *******  f
    0.938     2.8e-02  -1.9e+06   2.8e-01  -1.6e-16   1.7198    f
    1.094     2.8e-02   3.1e-07   2.8e-01  -1.9e-16   1.0727    f
    1.172     2.8e-02   3.1e-07   2.8e-01   0.0e+00   0.5933    p

 Gamma value achieved:     1.1719
```

Now form the closed loop and check the pole positions. As expected, it is stable.

```
Ghinf = starp(P,Khinf)
rifd(Ghinf)
Poles:

      real        imaginary      frequency      damping
                                 (rad/sec)       ratio

  -2.8248e-02     3.4100e+01     3.4100e+01     0.0008
  -2.8248e-02    -3.4100e+01     3.4100e+01     0.0008
  -2.7911e-01     3.4212e+01     3.4213e+01     0.0082
  -2.7911e-01    -3.4212e+01     3.4213e+01     0.0082
  -2.9727e+01    -3.0010e+01     4.2241e+01     0.7038
  -2.9727e+01     3.0010e+01     4.2241e+01     0.7038
```

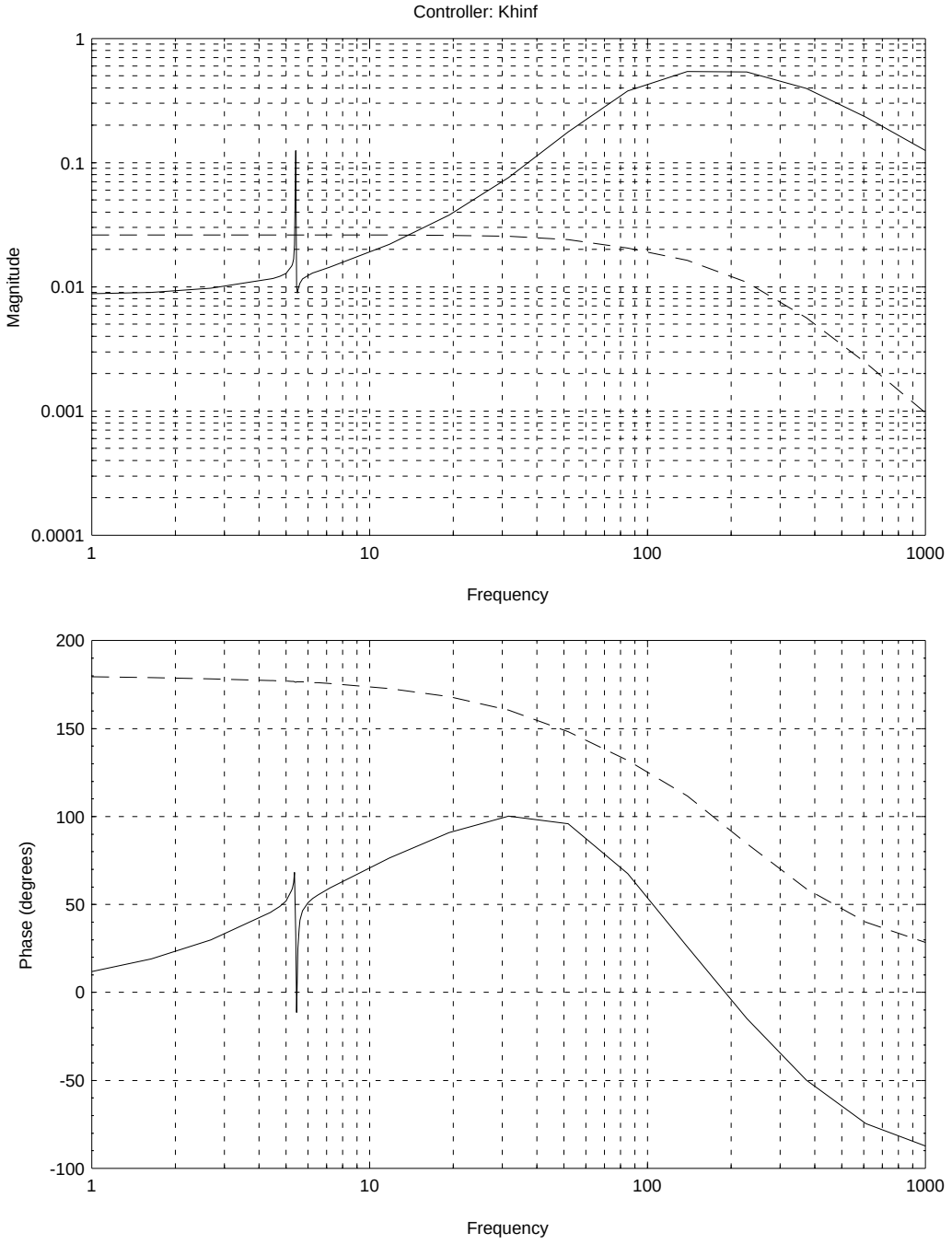
```
  -3.5543e+02    -3.5543e+02     5.0265e+02     0.7071
  -3.5543e+02     3.5543e+02     5.0265e+02     0.7071
  -6.4598e+02    -3.3537e+01     6.4685e+02     0.9987
  -6.4598e+02     3.3537e+01     6.4685e+02     0.9987
  -3.8358e+02    -5.3777e+02     6.6055e+02     0.5807
  -3.8358e+02     5.3777e+02     6.6055e+02     0.5807
  -1.6934e+03     0.0000e+00     1.6934e+03     1.0000
  -2.8107e+03     0.0000e+00     2.8107e+03     1.0000
  -6.2832e+03     0.0000e+00     6.2832e+03     1.0000
  -6.2832e+04     0.0000e+00     6.2832e+04     1.0000
Zeros:
```

We can also look at the controller poles. It turns out that our controller is stable.

```
rifd(Khinf)
Poles:

      real        imaginary      frequency      damping
                                 (rad/sec)       ratio

  -2.8002e-02     3.4100e+01     3.4100e+01     0.0008
  -2.8002e-02    -3.4100e+01     3.4100e+01     0.0008
  -3.5543e+02    -3.5543e+02     5.0265e+02     0.7071
  -3.5543e+02     3.5543e+02     5.0265e+02     0.7071
  -6.4828e+02     0.0000e+00     6.4828e+02     1.0000
  -1.1873e+03    -3.0188e+02     1.2251e+03     0.9692
  -1.1873e+03     3.0188e+02     1.2251e+03     0.9692
  -2.8870e+03     0.0000e+00     2.8870e+03     1.0000

Zeros:
```

We also look at a frequency response of the controller. Recall that it is single-input, two-output.

```
Khinfg = freq(Khinf,omega);
gph3 = ctrlplot(Khinfg,{bode});
gph3 = plot(gph3,{title="Controller:  Khinf"})?
```


Controller: Khinf
Magnitude
1
0.1
0.01
0.001
0.0001
1
10
100
1000
Frequency
Phase (degrees)
200
150
100
50
0
-50
-100
1
10
100
1000
Frequency

4.2.4 Robustness Analysis

The block structure has two perturbation Δ blocks and a "performance" block. The two voice-coil perturbations are put into a single 1×2 block as they enter the system at the same point. Note that this is not identical to two separate blocks — for example a perturbation in which both blocks have magnitude one is now longer included. The advantage of doing this is that it give one less block in the resulting analysis and design problem. This makes the μ calculation easier (we now have three blocks and so the upper bound is actually equal to μ) and gives one less D-scale to be approximated in the D-K iteration.

```
blk = [1,2; 1,1; 2,3]
```

The frequency response of the closed loop system is calculated. The nominal performance test simply involves checking the $\mathcal{H}_\infty$ norm of the nominal closed loop system. This is a maximum singular value test.

```
Ghinfg = freq(Ghinf,omega);
npbnds = norm(svd(Ghinfg(4:6,3:4)),inf)
```

Robust stability is a μ test as there are two pertubation blocks. Note that here we calculate μ with respect to the G_{11} partition and use a block structure containing only the perturbation blocks.

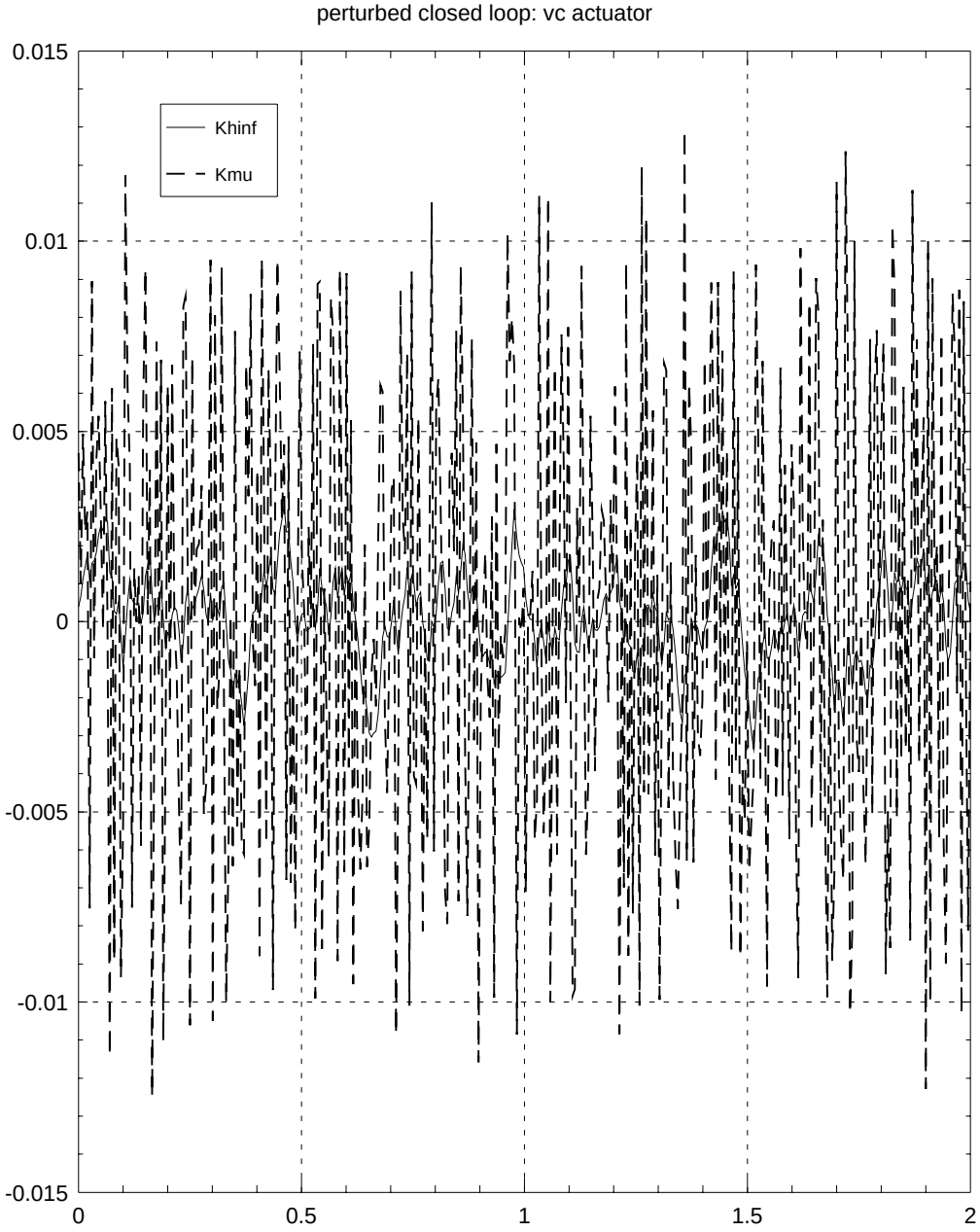
```
[rsbnds,Drs,Drsinv,Deltars,sensrs] = mu(Ghinfg(1:3,1:2),[1,2; 1,1])
```

Robust performance is a μ test on the entire G matrix.

```
[rpbnds,D,Dinv,Delta,sens] = mu(Ghinfg,blk)
```

These results are plotted. Note that μ tests give bounds and in this case the upper and lower bounds are almost indistinguishable.

```
gph4 = ctrlplot(npbnds,{log,line_style=4});
gph4 = ctrlplot(rsbnds,gph4,{log,line_style=[3,5]});
gph4 = ctrlplot(rpbnds,gph4,{log,line_style=[1,2]});
gph4 = plot(gph4,{!grid,legend=["Nom perf";"Rob stab (up bnd)";...
           "Rob stab (lw bnd)"; "Rob perf (up bnd)";...
           "Rob perf (lw bnd)"],title="mu analysis of Ghinf"})?
```


perturbed closed loop: vc actuator
Khinf
Kmu
0.015
0.01
0.005
0
-0.005
-0.01
-0.015
0
0.5
1
1.5
2

4.2.5 D-K Iteration

We will now perform one D-K iteration to generate the controller `Kmu`. Significant robustness and performance improvement is achieved with only one iteration.

Transfer functions are fit to the D-scales from the previous robust performance μ test. Here we preselect an order of 2 for each D-scale. This has been found to give a satisfactory result.

```
[Dsys,Dinvsys] = musynfit(D,blk,nmeas,ncon,sens,[],2)
```

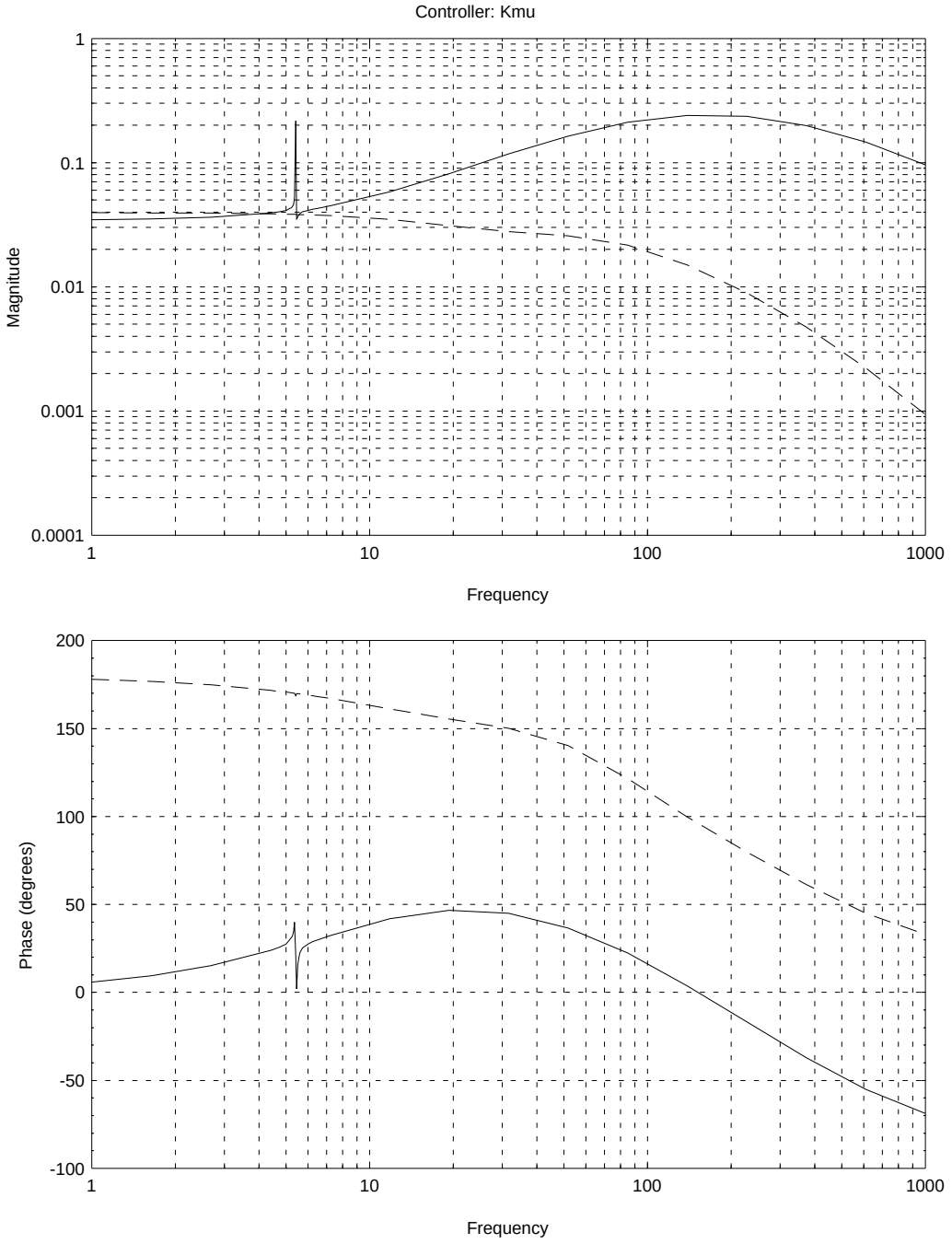
Now a new weighted interconnection is formed by pre- and post-multiplying by the D-scale approximations. A second $\mathcal{H}_\infty$ design is performed to get `Kmu`.

```
Pd = Dsys*P*Dinvsys
glimits = [0;20]
Kmu = hinfsyn(Pd,nmeas,ncon,glimits,{tol=0.1,epr=1e-10,epp=1e-4})
Test bounds:      0.0000  <  gamma  <=     20.0000
  gamma      Hx_eig     X_eig    Hy_eig      Y_eig   nrho_xy   p/f
   20.000     2.9e-02   3.5e-10   1.2e-01  -1.2e-15   0.0001    p
   10.000     2.9e-02   3.3e-10   1.2e-01  -3.8e-16   0.0006    p
    5.000     2.9e-02   3.7e-10   1.2e-01  -3.7e-16   0.0022    p
    2.500     2.9e-02   3.1e-10   1.2e-01  -3.7e-16   0.0090    p
    1.250     2.9e-02   3.6e-10   1.2e-01  -1.8e-15   0.0364    p
    0.625     2.8e-02   3.4e-10   1.2e-01  -2.1e-16   0.1518    p
    0.312     2.8e-02   3.8e-10   1.2e-01  -2.2e-29   0.7425    p
    0.156     2.1e-02  -4.3e+06   1.5e-01  -2.2e-17  18.9961    f
    0.234     2.6e-02   3.6e-10   1.3e-01  -2.3e-16   1.7666    f
    0.273     2.7e-02   3.8e-10   1.2e-01  -9.8e-17   1.0735    f
    0.293     2.7e-02   3.9e-10   1.2e-01  -1.9e-16   0.8828    p

 Gamma value achieved:     0.2930
```

Note that this value of γ is significantly lower than even the μ value from the `Ghinf` closed loop system. Again, both the controller and closed loop system are stable.

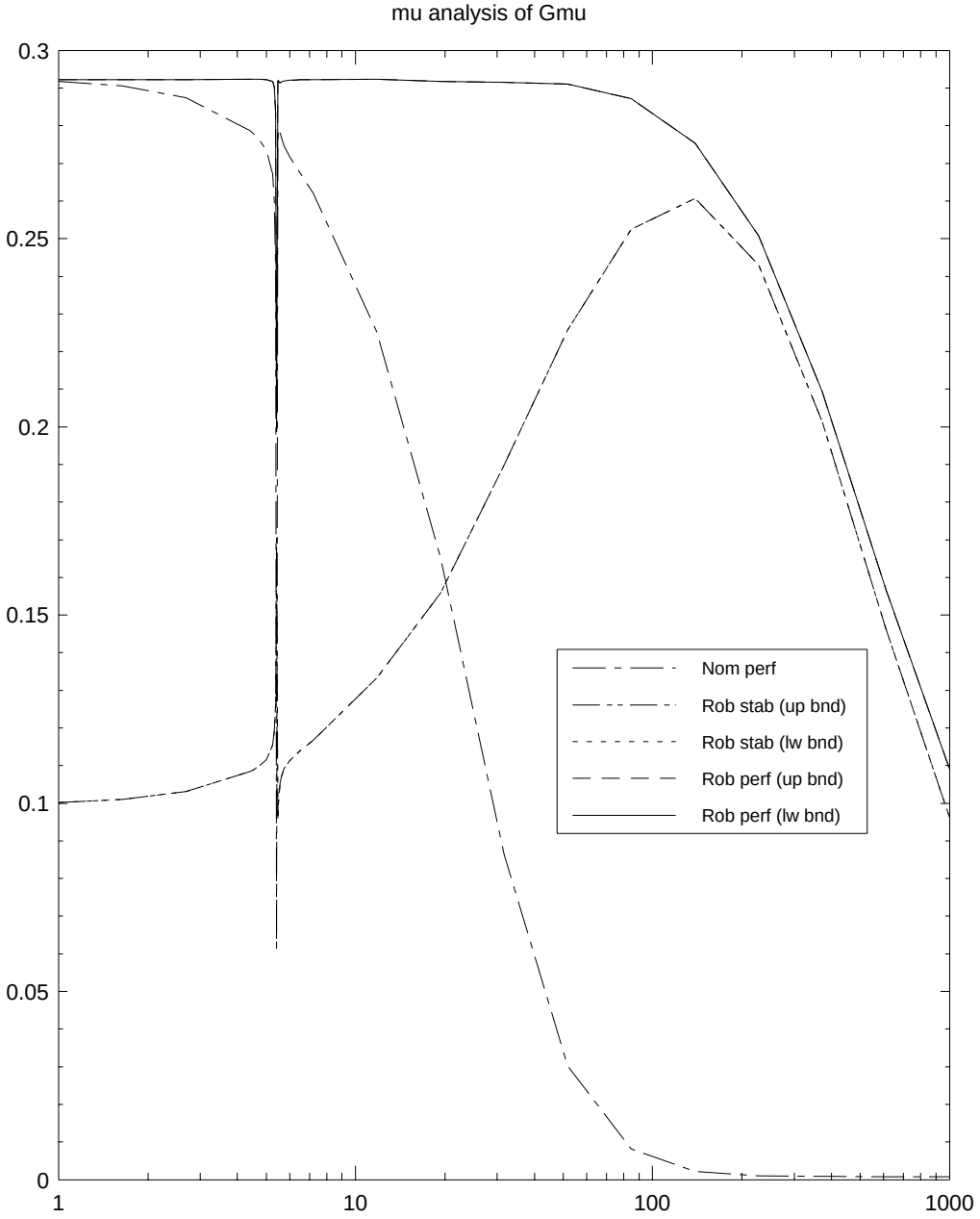
A frequency response of `Kmu` is calculated and plotted.

```
Kmug = freq(Kmu,omega)
gph5 = ctrlplot(Kmug,{bode});
gph5 = plot(gph5,{title="Controller:  Kmu"})?
```


Controller: Kmu
Magnitude
1
0.1
0.01
0.001
0.0001
1
10
100
1000
Frequency
Phase (degrees)
200
150
100
50
0
-50
-100
1
10
100
1000
Frequency

We now examine the robustness properties of the new closed loop system. We already know that the robust performance test will be less than the γ value from the $\mathcal{H}_\infty$ synthesis above (in this case 0.2930). The results are again displayed graphically.

```
Gmug = freq(Gmu,omega)
npbnds = norm(svd(Gmug(4:6,3:4)),inf)
[rsbnds,Drs,Drsinv,Deltars,sensrs] = mu(Gmug(1:3,1:2),[1,2; 1,1])
[rpbnds,D,Dinv,Delta,sens] = mu(Gmug,blk)

gph6 = ctrlplot(npbnds,{log,line_style=4});
gph6 = ctrlplot(rsbnds,gph6,{log,line_style=[3,5]});
gph6 = ctrlplot(rpbnds,gph6,{log,line_style=[1,2]});
gph6 = plot(gph6,{!grid,legend=["Nom perf";"Rob stab (up bnd)";...
            "Rob stab (lw bnd)"; "Rob perf (up bnd)";...
            "Rob perf (lw bnd)"],title="mu analysis of Gmu"})?
```


mu analysis of Gmu
0.3
0.25
0.2
0.15
0.1
0.05
0
1
10
100
1000
Nom perf
Rob stab (up bnd)
Rob stab (lw bnd)
Rob perf (up bnd)
Rob perf (lw bnd)

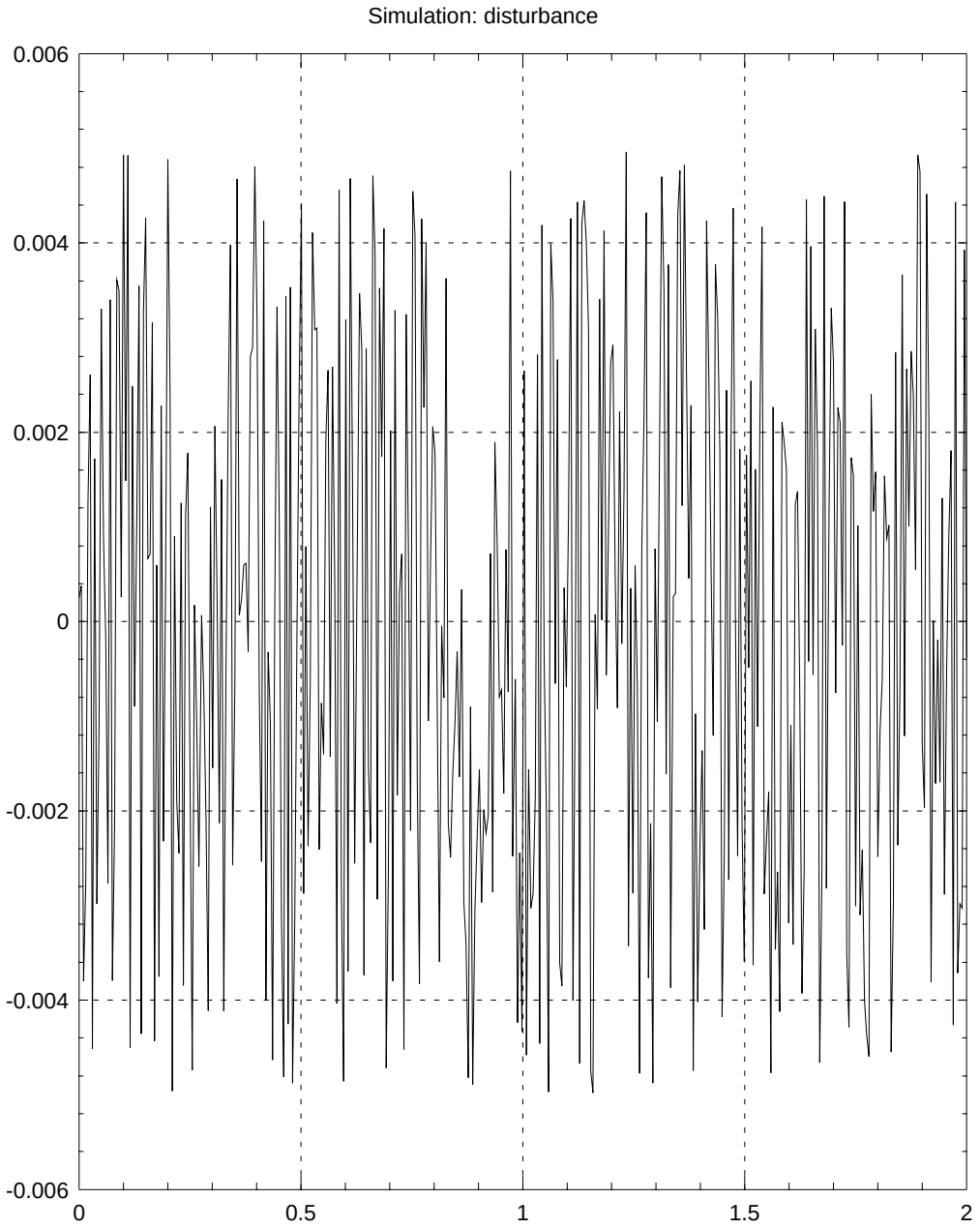
4.2.6 A Simulation Study

Now the two controllers (`Khinf` and `Kmu`) are studied by simulation. An unweighted interconnection is set up with `sysic` and `starp` is used to close the loop for each controller.

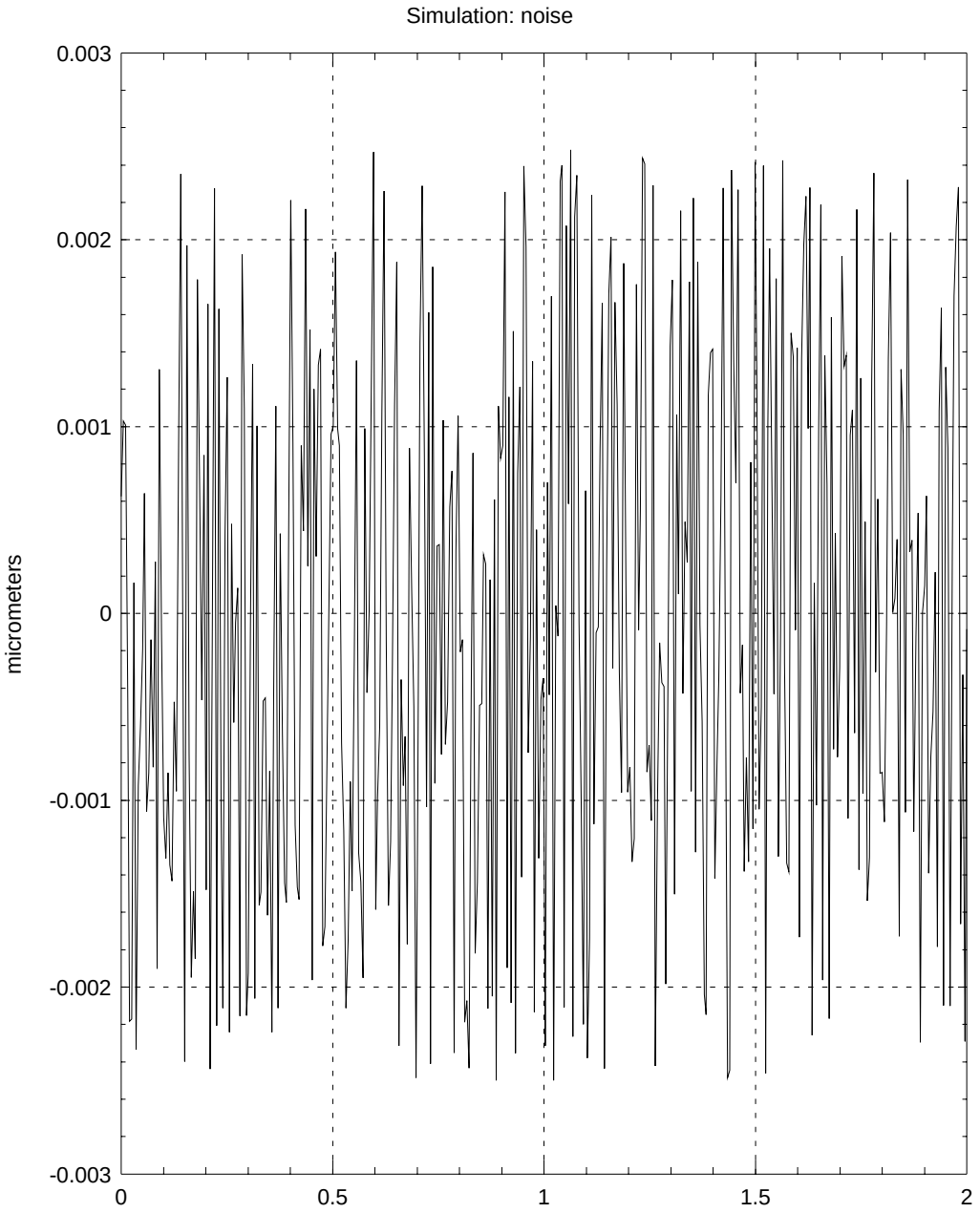
```
ssnames = ["vcmodel"; "Wavoice"; "Wmvoice"; "Wmpiezo"; "piezo"]
inps = ["d1i"; "d2i"; "dist"; "noise"; "vact"; "pact"]
ops = ["Wavoice"; "Wmvoice"; "Wmpiezo";...
      "d1i + vcmodel + d2i + piezo + noise"; "vact"; "pact";...
      "d1i + vcmodel + d2i + piezo + noise"]
cnx = ["dist + vact"; "dist + vact"; "vcmodel"; "piezo"; "pact"]

Pnom =
sysic(ssnames,inps,ops,cnx,vcmodel,Wavoice,Wmvoice,Wmpiezo,piezo)
```

Random inputs are created for the noise and structure disturbances. Both are normally distributed. These are passed through their respective performance weights to give signals of the appropriate size (and if necessary frequency content). Both signals are plotted.

```
u1 = randpdm(400,1,1,{Dlast=2,regular,zeromean})
u2 = randpdm(400,1,1,{Dlast=2,regular,zeromean})
u = [Wdist*u1; Wnoise*u2]

gph7 = ctrlplot(u(1,1));
gph7 = plot(gph7,{title="Simulation:  disturbance"})?
```

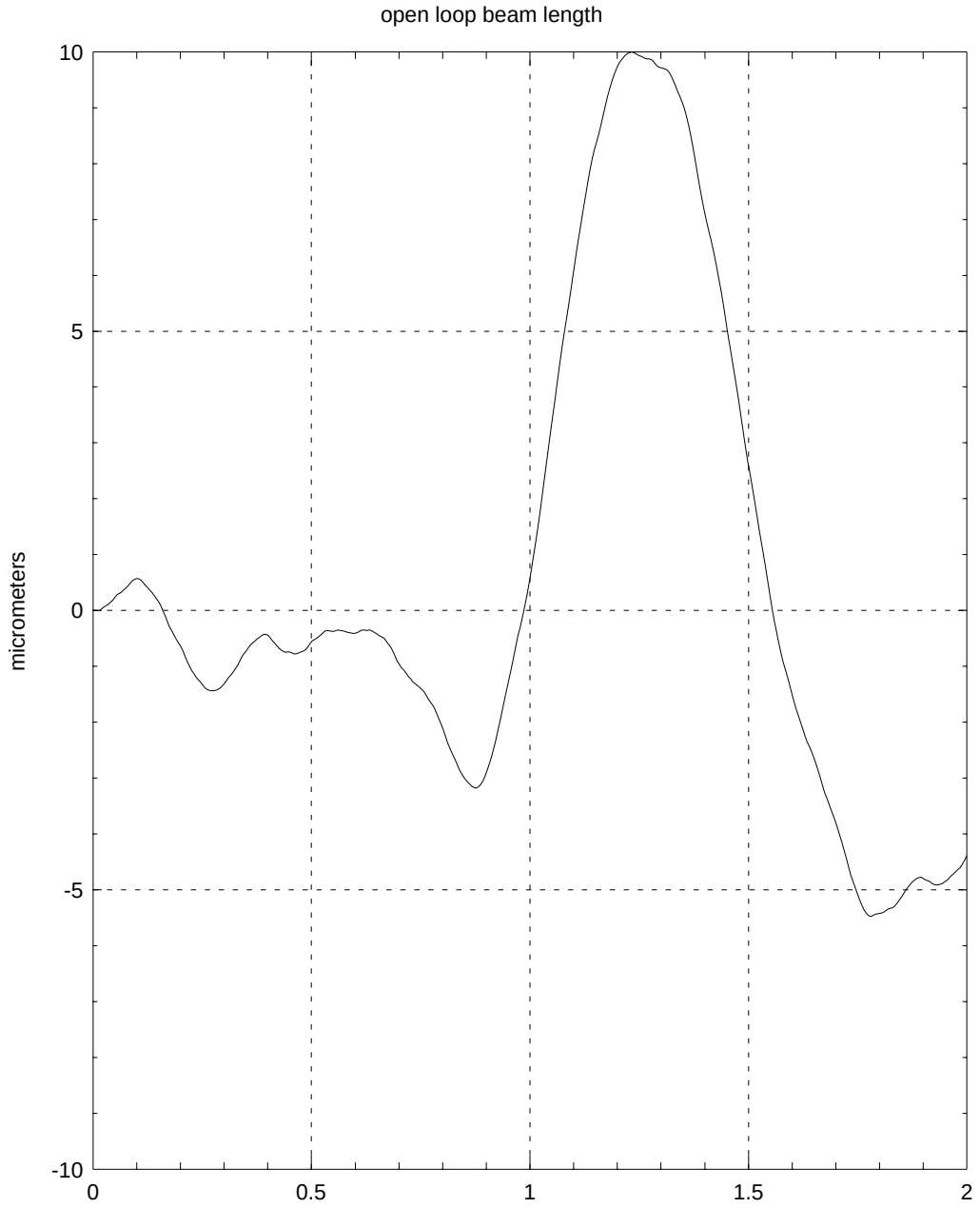

Simulation: disturbance
0.006
0.004
0.002
0
-0.002
-0.004
-0.006
0
0.5
1
1.5
2

```
gph8 = ctrlplot(u(2,1));
gph8 = plot(gph8,{title="Simulation:  noise",...
            y_lab="micrometers"})?
```


Simulation: noise
micrometers
0.003
0.002
0.001
0
-0.001
-0.002
-0.003
0
0.5
1
1.5
2

A nominal response is calculated by setting $\Delta = 0$. To get the open-loop simulation model we close the unweighted interconnection structure with a controller equal to zero.

```
deltazero = zeros(2,3)
Kzero = zeros(2,1)
nomolp = starp(deltazero,Pnom)
nomolp = starp(nomolp,Kzero)
yolp = nomolp*u

gph9 = ctrlplot(yolp(1,1));
gph9 = plot(gph9,{title="open loop beam length",...
            y_lab="micrometers"})?
```

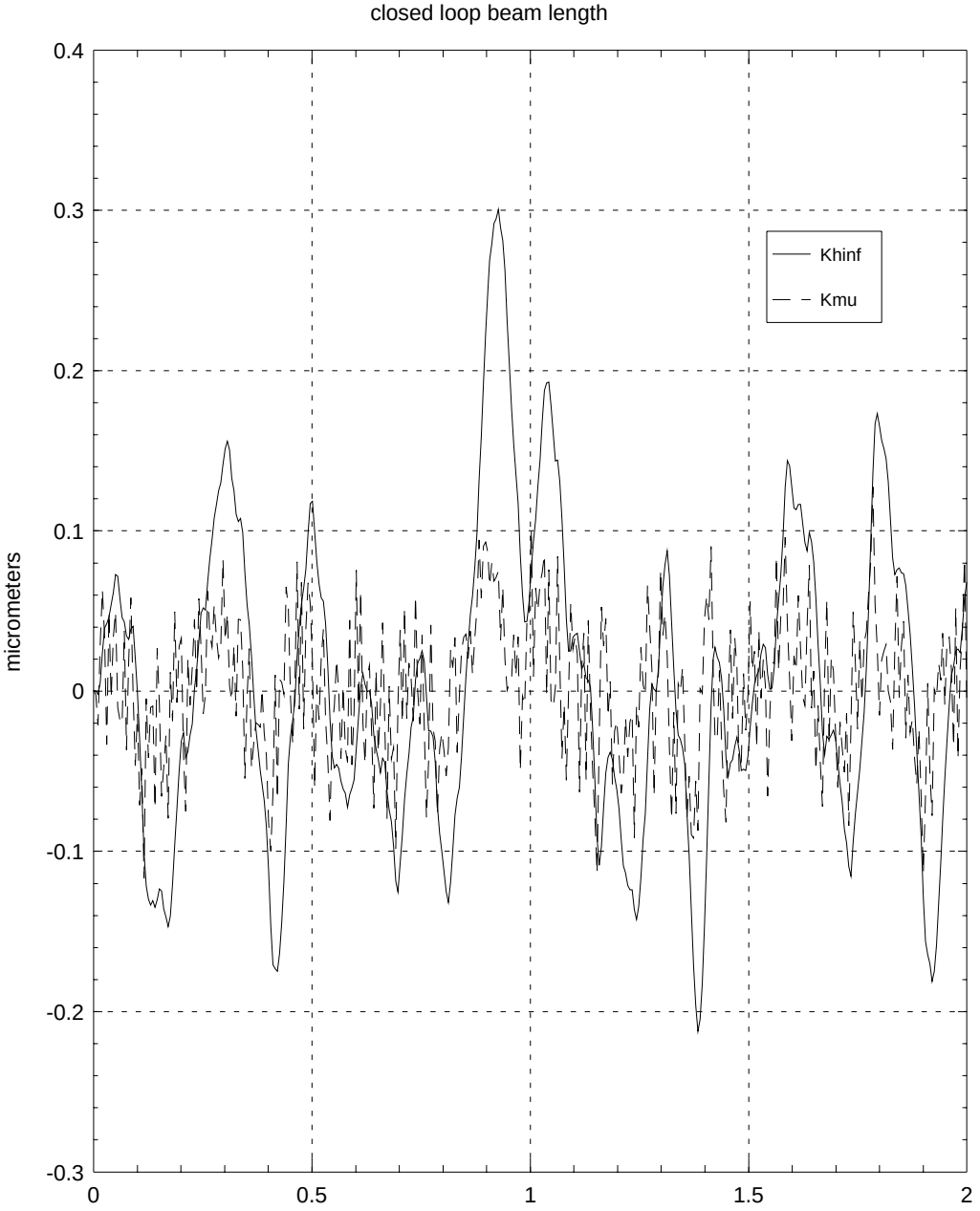

open loop beam length
micrometers
10
5
0
-5
-10
0
0.5
1
1.5
2

Now we consider the closed-loop nominal response with `Khinf` and `Kmu`. The closed-loop system happens to have a large number of high frequency poles which do not contribute significantly to the response. They have the effect of forcing a very fine time discretization in the simulation, resulting in a long calculation time. We remove all poles of frequency greater than 100 rad/sec. by residualization. This is not intended as a general procedure — in some situations the high frequency behavior will be significant.

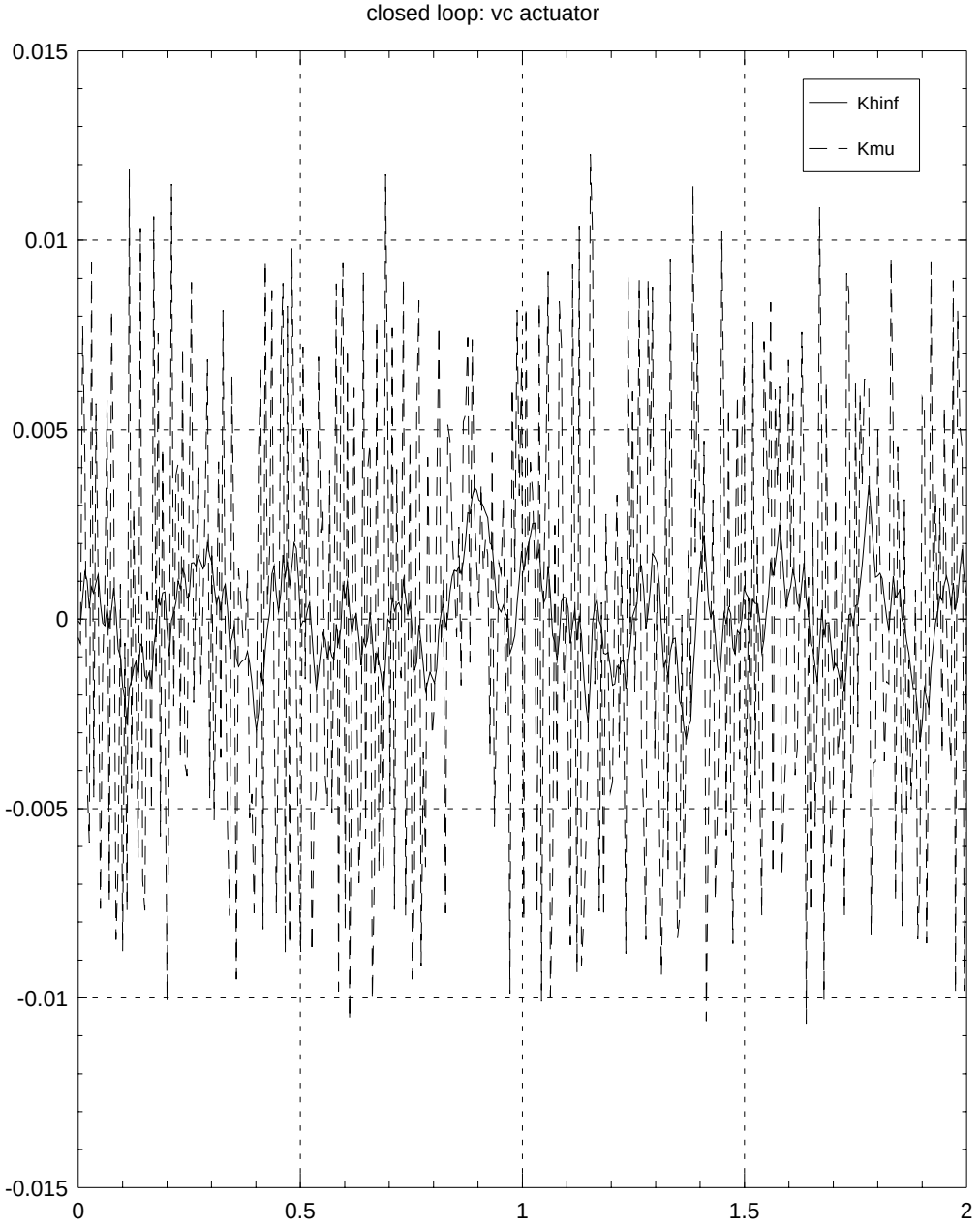
```
nomclp = starp(deltazero,Pnom)
nomclphinf = starp(nomclp,Khinf)
nomclpmu = starp(nomclp,Kmu)

fmax = 100;
nlfpoles = sum(abs(poles(nomclphinf))<fmax)
str = "Residualizing closed loop system to " + ...
                      string(nlfpoles)+" states"
display(str)
Residualizing closed loop system to 6 states
nsysclphinf = modalstate(nomclphinf)
nsysclphinf = sresidualize(nsysclphinf,nlfpoles)

nlfpoles = sum(abs(poles(nomclpmu))<fmax)
nsysclpmu = modalstate(nomclpmu)
nsysclpmu = sresidualize(nsysclpmu,nlfpoles)
```

The closed-loop responses are calculated and plotted.

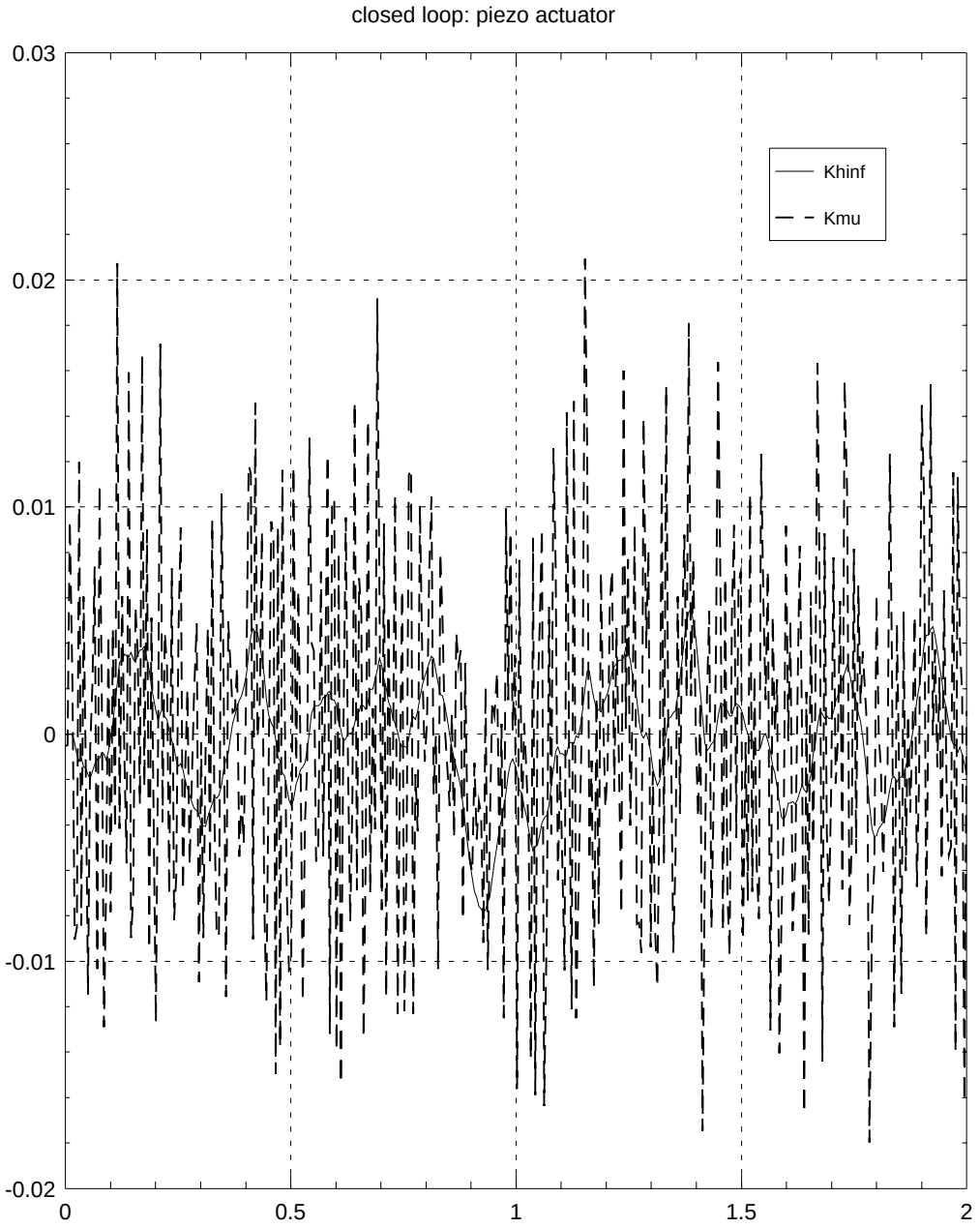
```
yclphinf = nsysclphinf*u
yclpmu = nsysclpmu*u

gph10 = ctrlplot(yclphinf(1,1));
gph10 = ctrlplot(yclpmu(1,1),gph10);
gph10 = plot(gph10,{legend=["Khinf";"Kmu"],...
             title="closed loop beam length",...
             y_lab="micrometers"})?
```


closed loop beam length
micrometers
Khinf
Kmu
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
0
0.5
1
1.5
2

```
gph11 = ctrlplot(yclphinf(2,1));
gph11 = ctrlplot(yclpmu(2,1),gph11);
gph11 = plot(gph11,{legend=["Khinf";"Kmu"],...
             title="closed loop:  vc actuator"})?
```


closed loop: vc actuator
Khinf
Kmu
0.015
0.01
0.005
0
-0.005
-0.01
-0.015
0
0.5
1
1.5
2

```
gph12 = ctrlplot(yclphinf(3,1));
gph12 = ctrlplot(yclpmu(3,1),gph12);
gph12 = plot(gph12,{legend=["Khinf";"Kmu"],...
             title="closed loop:  piezo actuator"})?
```


closed loop: piezo actuator
0.03
0.02
0.01
0
-0.01
-0.02
0
0.5
1
1.5
2
Khinf
Kmu

Note that `Kmu` achieves better performance at the expense of greater actuator effort.

We will now repeat this simulation for a perturbed system. A bad Δ is chosen and scaled to have norm 0.5. This is obtained from destabilizing Δ calculated for the μ lower bound. An all-pass system is fitted to Δ at one frequency to create a real-rational perturbation. The frequency selected is that where μ is at its maximum. In this case we choose the Δ that comes from the robust stability μ test. This gives the perturbation which comes closest to destabilizing the closed loop system. Using Δ from the robust performance test would allow us to create a perturbation which minimizes the robust performance of the closed loop system.

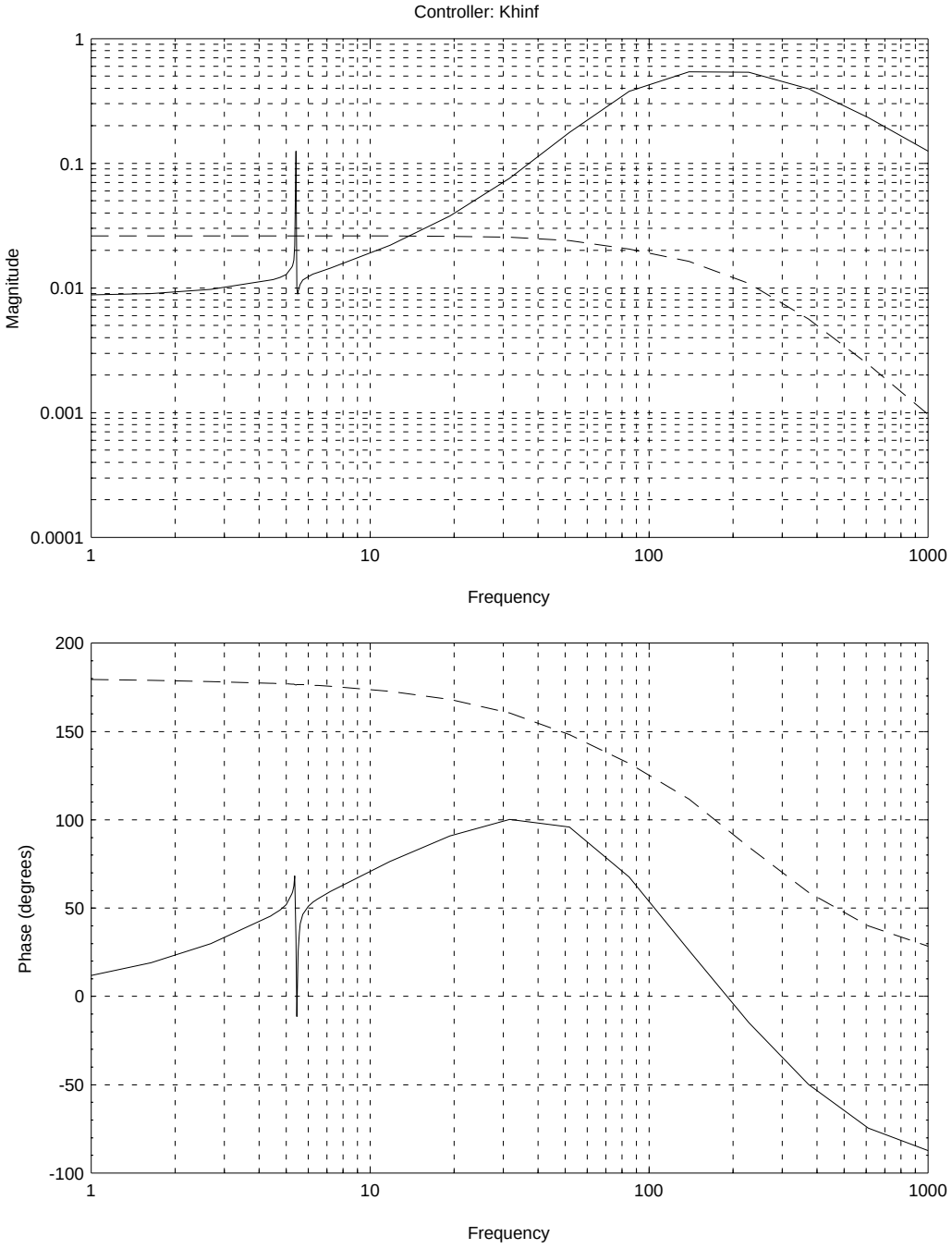
Here we use the worst-case Δ for `Kmu`. This is almost certainly not the worst-case Δ for `Khinf`. Again, the closed loop systems are residualized prior to calculating the responses.

```
deltabad = mkpert(Deltars,blk(1:2,1:2),rsbnds,{pnorm=0.5})

badclp = starp(deltabad,Pnom)
badclphinf = starp(badclp,Khinf)
badclpmu = starp(badclp,Kmu)

nlfpoles = sum(abs(poles(badclphinf))<fmax)
str = "Residualizing closed loop system to "+...
                     string(nlfpoles)+" states"
display(str)
Residualizing closed loop system to 6 states
nbclphinf = modalstate(badclphinf)
nbclphinf = sresidualize(nbclphinf,nlfpoles)

nlfpoles = sum(abs(poles(badclpmu))<fmax)
nbclpmu = modalstate(badclpmu)
nbclpmu = sresidualize(nbclpmu,nlfpoles)
```

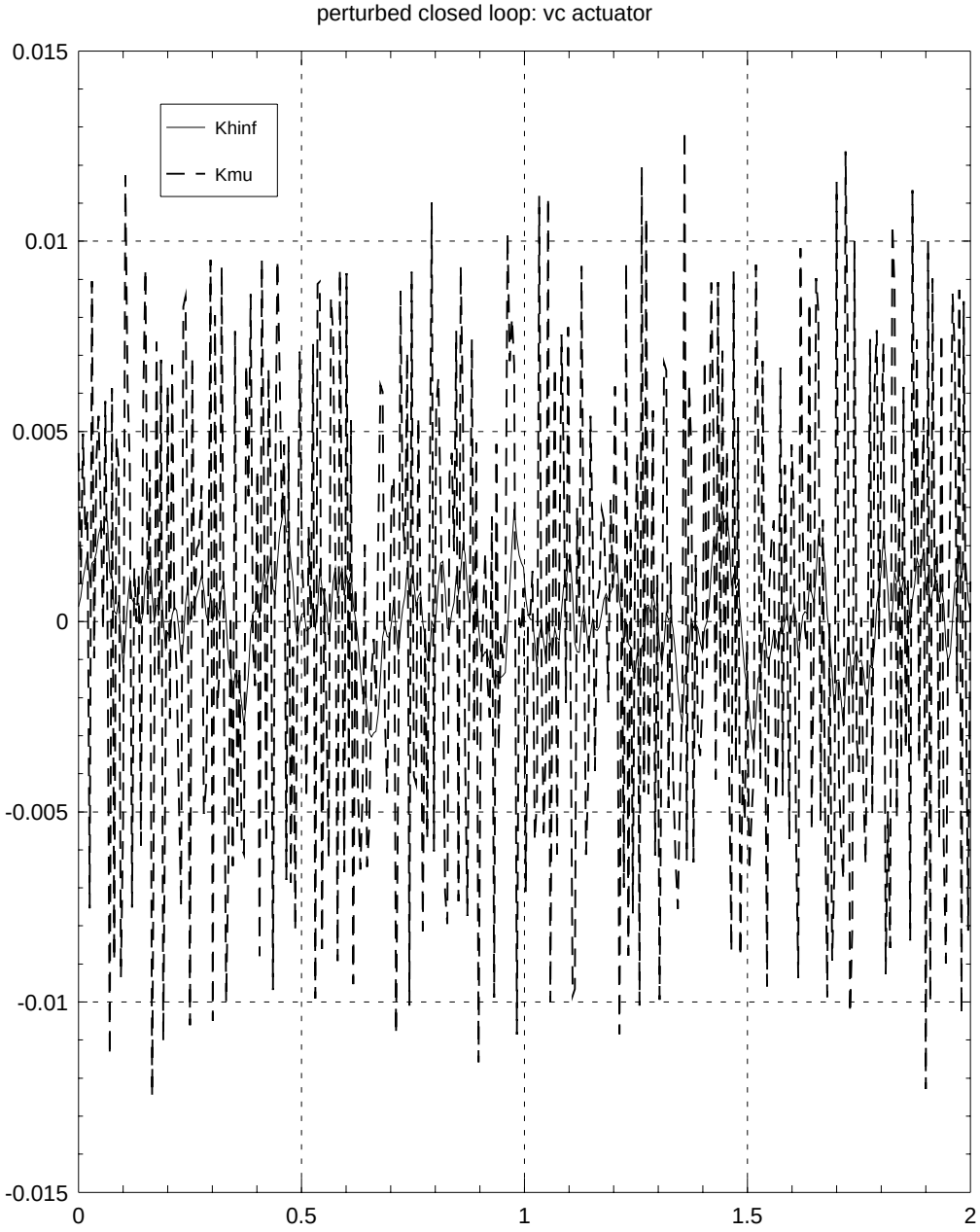
The responses are calculated and plotted.

```
ybclphinf = nbclphinf*u
ybclpmu = nbclpmu*u

gph13 = ctrlplot(ybclphinf(1,1));
```

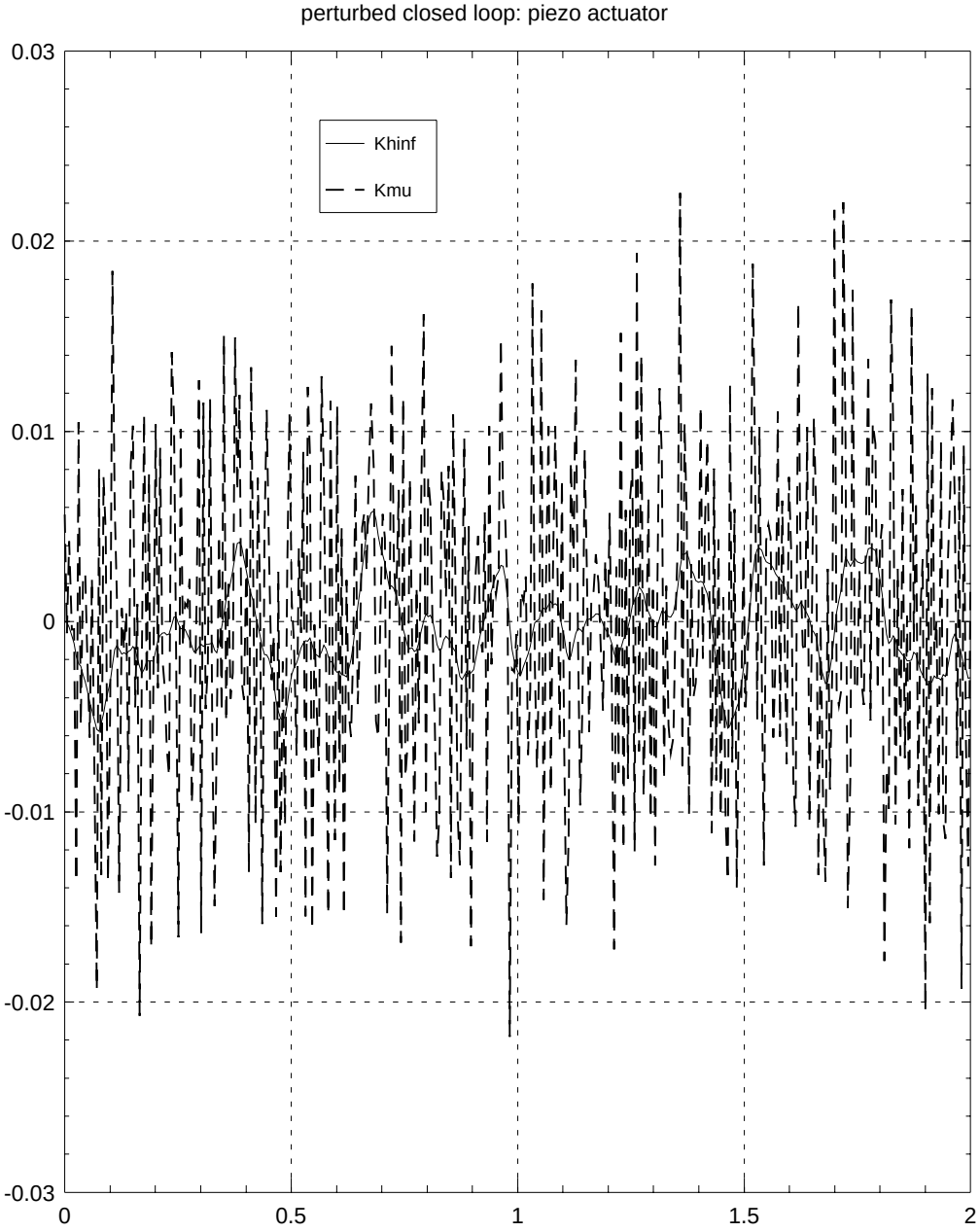
```
gph13 = ctrlplot(ybclpmu(1,1),gph13);
gph13 = plot(gph13,{legend=["Khinf";"Kmu"],...
             title="perturbed closed loop beam length",...
             y_lab="micrometers"})?
```


Controller: Khinf
Magnitude
1
0.1
0.01
0.001
0.0001
1
10
100
1000
Frequency
Phase (degrees)
200
150
100
50
0
-50
-100
1
10
100
1000
Frequency

```
gph14 = ctrlplot(ybclphinf(2,1));
gph14 = ctrlplot(ybclpmu(2,1),gph14);
gph14 = plot(gph14,{legend=["Khinf";"Kmu"],...
             title="perturbed closed loop:  vc actuator"})?
```


perturbed closed loop: vc actuator
Khinf
Kmu
0.015
0.01
0.005
0
-0.005
-0.01
-0.015
0
0.5
1
1.5
2

```
gph15 = ctrlplot(ybclphinf(3,1));
gph15 = ctrlplot(ybclpmu(3,1),gph15);
gph15 = plot(gph15,{legend=["Khinf";"Kmu"],...
             title="perturbed closed loop:  piezo actuator"})?
```


perturbed closed loop: piezo actuator
Khinf
Kmu
0.03
0.02
0.01
0
-0.01
-0.02
-0.03
0
0.5
1
1.5
2

Bibliography

[1] J. C. Doyle, "Lecture notes on advances in multivariable control." ONR/Honeywell Workshop, Minneapolis, MN., 1984.

[2] J. Doyle, "Structured uncertainty in control system design," in *Proc. IEEE Control Decision Conf.*, pp. 260–265, 1985.

[3] A. K. Packard, *What's new with μ: Structured Uncertainty in Multivariable Control*. PhD thesis, University of California, Berkeley, 1988.

[4] R. S. Smith, *Model Validation for Uncertain Systems*. PhD thesis, California Institute of Technology, 1990.

[5] J. Doyle and G. Stein, "Multivariable feedback design: Concepts for a classical/modern synthesis," *IEEE Trans. Auto. Control*, vol. AC-26, pp. 4–16, Feb. 1981.

[6] M. Morari and E. Zafiriou, *Robust Process Control*. New Jersey: Prentice-Hall, 1989.

[7] The MathWorks, Inc., Natick, MA, *μ-Analysis and Synthesis Toolbox (μ-Tools)*, 1991.

[8] J. Doyle, K. Lenz, and A. K. Packard, "Design examples using μ synthesis: Space shuttle lateral axis FCS during reentry," in *Proc. IEEE Control Decision Conf.*, pp. 2218–2223, dec 1986.

[9] G. J. Balas, *Robust Control of Flexible Structures, Theory and Experiment*. PhD thesis, California Institute of Technology, 1990.

[10] G. J. Balas and J. C. Doyle, "Identification of flexible structures for robust control," *IEEE Control Sys. Magazine*, vol. 10, pp. 51–58, June 1990.

[11] G. J. Balas and J. C. Doyle, "Robust control of flexible modes in the controller crossover region," in *Proc. Amer. Control Conf.*, 1989.

[12] G. J. Balas, A. K. Packard, and J. Harduvel, "Application of μ-synthesis techniques to momentum management and attitude control of the Space Station," in *AIAA Guidance, Navigation and Cont. Conf.*, 1991.

[13] R. S. Smith, C.-C. Chu, and J. L. Fanson, "The design of H_∞ controllers for an experimental non-collocated flexible structure problem," *IEEE Trans. Control Syst. Tech.*, 1993. in press.

[14] J. Fanson, C.-C. Chu, B. Lurie, and R. Smith, "Damping and structural control of the JPL phase 0 testbed structure," *J. Intell. Material Sys. & Struct.*, vol. 2, pp. 281–300, July 1991.

[15] R. S. Smith, 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.

[16] R. S. Smith and J. Doyle, "The two tank experiment: A benchmark control problem," in *Proc. Amer. Control Conf.*, vol. 3, pp. 403–415, 1988.

[17] S. Skogestad, 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.

[18] S. Skogestad, "Correction to "Robust control of ill-conditioned plants: High purity distillation"," *IEEE Trans. Auto. Control*, vol. 34, p. 672, June 1989.

[19] J. Doyle and A. K. Packard, "Uncertain multivariable systems from a state space perspective," in *Proc. Amer. Control Conf.*, vol. 3, pp. 2147–2152, 1987.

[20] A. K. Packard and J. C. Doyle, "The complex structured singular value," *Automatica*, vol. 29, no. 1, pp. 71–109, 1993.

[21] D. R. Hamburg and M. A. Shulman, "A closed loop A/F control model for internal combustion engines," *Soc. Automotive Eng.*, no. 800826, 1980.

[22] B. G. Morton and R. M. McAfoos, "A mu-test for robustness analysis of a real-parameter variation problem," in *Proc. Amer. Control Conf.*, 1985.

[23] J. Doyle and A. Packard, "Uncertain multivariable systems from a state space perspective," in *Proc. Amer. Control Conf.*, pp. 2147–2152, IEEE, 1987.

[24] D. L. Laughlin, K. G. Jordan, and M. Morari, "Internal model control and process uncertainty: mapping uncertainty regions for SISO controller design," *Int. J. of Control*, vol. 44, no. 6, pp. 1675–1698, 1986.

[25] R. S. Smith and M. Dahleh, eds., *The Modeling of Uncertainty in Control Systems: Proceedings of the 1992 Santa Barbara Workshop.* 391 pgs., Springer-Verlag, 1994.

[26] M. Gevers, "Connecting identification and robust control: A new challenge," in *Proc. IFAC Symp. on Identification & System Parameter Estimation*, vol. 1, pp. 1–10, 1991.

[27] A. Helmicki, C. Jacobson, and C. Nett, "H_∞ identification of stable lsi systems: A scheme with direct application to controller design," *Proc. Amer. Control Conf.*, pp. 1428–1434, 1989.

[28] G. Gu and P. P. Khargonekar, "Linear and nonlinear algorithms for identification in H^∞ with error bounds," in *Proc. Amer. Control Conf.*, pp. 64–69, 1991.

[29] A. J. Helmicki, C. A. Jacobson, and C. N. Nett, "Control oriented system identification: A worst-case/deterministic approach in H_∞," *IEEE Trans. Auto. Control*, pp. 1163–1176, 1991.

[30] P. Mäkilä and J. Partington, "Robust approximation and identification in H^∞," *Proc. Amer. Control Conf.*, pp. 70–76, 1991.

[31] G. Gu and P. P. Khargonekar, "Linear and nonlinear algorithms for identification in H^∞ with error bounds," in *IEEE Trans. Auto. Control*, vol. 37, pp. 953–963, 1992.

[32] G. Gu and P. P. Khargonekar, "A class of algorithms for identification in H^∞," in *Automatica*, vol. 28, pp. 299–312, 1992.

[33] G. Gu, P. P. Khargonekar, and Y. Li, "Robust convergence of two-stage nonlinear algorithms for identification in H^∞," in *Syst. and Control Letters*, vol. 18, pp. 253–263, 1992.

[34] R. G. Hakvoort, "Worst-case system identification in H_∞: error bounds and optimal models," in *Selected Topics in Identification Modelling and Control*, Delft University Press, Vol. 5 1992.

[35] E.-W. Bai, "On-line H_2, H_∞ and pointwise uncertainty bound quantification in identification of restricted complexity models," in *Proc. IEEE Control Decision Conf.*, pp. 1719–1724, 1992.

[36] G. Goodwin and M. Salgado, "Quantification of uncertainty in estimation using an embedding principle," in *Proc. Amer. Control Conf.*, 1989.

[37] R. Kosut, M. Lau, and S. Boyd, "Parameter set identification of systems with uncertain nonparametric dynamics and disturbances," in *Proc. IEEE Control Decision Conf.*, vol. 6, pp. 3162–3167, 1990.

[38] G. Goodwin, B. Ninness, and M. Salgado, "Quantification of uncertainty in estimation," in *Proc. Amer. Control Conf.*, pp. 2400–2405, 1990.

[39] B. M. Ninness and G. C. Goodwin, "Robust frequency response estimation accounting for noise and undermodeling," in *Proc. Amer. Control Conf.*, pp. 2847–2851, 1992.

[40] R. S. Smith and J. Doyle, "Model invalidation — a connection between robust control and identification," in *Proc. Amer. Control Conf.*, pp. 1435–1440, 1989.

[41] J. M. Krause, "Stability margins with real parameter uncertainty: Test data implications," in *Proc. Amer. Control Conf.*, pp. 1441–1445, 1989.

[42] R. S. Smith and J. C. Doyle, "Model validation: A connection between robust control and identification," *IEEE Trans. Auto. Control*, vol. 37, pp. 942–952, July 1992.

[43] M. Newlin and R. S. Smith, "Model validation and generalized μ," in *Proc. IEEE Control Decision Conf.*, pp. 1257–1258, 1991.

[44] K. Poolla, P. Khargonekar, A. Tikku, J. Krause, and K. Nagpal, "A time-domain approach to model validation," in *Proc. Amer. Control Conf.*, pp. 313–317, 1992.

[45] R. S. Smith, "Model validation and parameter identification for systems in H_∞ and l_1," in *Proc. Amer. Control Conf.*, pp. 2852–2856, 1992.

[46] T. Zhou and H. Kimura, "Input-output extrapolation-minimization theorem and its application to model validation and robust identification," in *The Modeling of Uncertainty in Control: Proceedings of the 1992 Santa Barbara Workshop* (R. Smith and M. Dahleh, eds.), pp. 127–137, Springer-Verlag, 1994.

[47] R. S. Smith, "Model validation for robust control: an experimental process control application," in *Proc. of the 13th IFAC World Congress*, vol. 9, pp. 61–64, July 1993.

[48] J. M. Krause and P. P. Khargonekar, "Parameter identification in the presence of non-parametric dynamic uncertainty," *Automatica*, vol. 26, pp. 113–124, 1990.

[49] R. Smith and J. Doyle, "Towards a methodology for robust parameter identification," in *Proc. Amer. Control Conf.*, vol. 3, pp. 2394–2399, 1990.

[50] J. M. Krause, P. P. Khargonekar, and G. Stein, "Robust parameter adjustment with nonparametric weighted-ball-in-H^∞ uncertainty," *IEEE Trans. Auto. Control*, vol. AC-35, pp. 225–229, 1990.

[51] R. S. Smith and J. C. Doyle, "Closed loop relay estimation of uncertainty bounds for robust control models," in *Proc. of the 13th IFAC World Congress*, vol. 9, pp. 57–60, July 1993.

[52] R. J. Schrama and P. M. V. den Hof, "An iterative scheme for identification and control design based on coprime factorizations," in *Proc. Amer. Control Conf.*, pp. 2842–2846, 1992.

[53] R. J. P. Schrama, "Accurate identification for control: the necessity of an iterative scheme," *IEEE Trans. Auto. Control*, vol. 37, pp. 991–994, July 1992.

[54] Z. Zang, R. R. Bitmead, and M. Gevers, "H_2 iterative model refinement and control robustness enhancement," in *Proc. IEEE Control Decision Conf.*, pp. 279–284, 1991.

[55] D. Bayard, Y. Yam, and E. Mettler, "A criterion for joint optimization of identification and robust control," *IEEE Trans. Auto. Control*, vol. 37, pp. 986–991, July 1992.

[56] Z. Zang, R. R. Bitmead, and M. Gevers, "Disturbance rejection: on-line refinement of controllers by closed loop modelling," in *Proc. Amer. Control Conf.*, pp. 2929–2833, 1992.

[57] B. A. Francis, *A Course in H_∞ Control Theory*, vol. 88 of *Lecture Notes in Control and Information Sciences*. Berlin: Springer-Verlag, 1987.

[58] J. Doyle, K. Glover, P. Khargonekar, and B. Francis, "State-space solutions to standard H_2 and H_∞ control problems," *IEEE Trans. Auto. Control*, vol. AC-34, pp. 831–847, 1989.

[59] K. Glover and J. Doyle, "State-space formulae for all stabilizing controllers that satisfy an H_∞ norm bound and relations to risk sensitivity," *Syst. and Control Letters*, vol. 11, pp. 167–172, Oct 1988.

[60] B. D. O. Anderson, "An algebraic solution to the spectral factorization problem," *IEEE Trans. Auto. Control*, vol. AC-12, pp. 410–414, 1967.

[61] J. C. Willems, "Least-squares stationary optimal control and the algebraic Riccati equation," *IEEE Trans. Auto. Control*, vol. AC-16, pp. 621–634, 1971.

[62] S. Boyd, V. Balakrishnan, and P. Kabamba, "On computing the h_∞ norm of a transfer matrix," *Math Contr. Signals, Syst.*, 1988.

[63] A. J. Laub, "A Schur method for solving algebraic Riccati equations," *IEEE Trans. Auto. Control*, vol. AC-24, pp. 913–921, 1979.

[64] T. Pappas, A. J. Laub, and N. R. Sandell, "On the numerical solution of the discrete-time algebraic Riccati equation," *IEEE Trans. Auto. Control*, vol. AC-25, pp. 631–641, 1980.

[65] W. F. Arnold and A. J. Laub, "Generalized eigenproblem algorithms and software for algebraic Riccati equations," *Proc. IEEE*, vol. 72, pp. 1746–1754, 1984.

[66] A. J. Laub, "Invariant subspace methods for the numerical solution of Riccati Equations," in *The Riccati Equation* (S. Bittanti, A. J. Laub, and J. C. Willems, eds.), pp. 163–196, Springer-Verlag, Berlin, 1991.

[67] G. Zames, "On the input-output stability of nonlinear time-varying feedback systems, parts I and II.," *IEEE Trans. Auto. Control*, vol. AC-11, pp. 228–238 and 465–476, 1966.

[68] J. Doyle, "Analysis of feedback systems with structured uncertainties," *IEE Proceedings, Part D*, vol. 133, pp. 45–56, Mar. 1982.

[69] M. K. H. Fan and A. L. Tits, "Characterization and efficient computation of the structured singular value," *IEEE Trans. Auto. Control*, vol. AC-31, pp. 734–743, 1986.

[70] M. K. H. Fan and A. L. Tits, "m-form numerical range and the computation of the structured singular value," *IEEE Trans. Auto. Control*, vol. AC-33, pp. 284–289, 1988.

[71] J. C. Doyle, A. K. Packard, P. M. Young, R. S. Smith, and M. P. Newlin, "The structured singular value," Tech. Rep. NASA-CR-4524, NASA, March 1992.

[72] M. G. Safonov and J. Doyle, "Minimizing conservativeness of robust singular values," in *Multivariable Control* (S. Tzafestas, ed.), New York: Reidel, 1984.

[73] M. K. H. Fan, A. L. Tits, and J. C. Doyle, "Robustness in the presence of joint parametric uncertainty and unmodeled dynamics," in *Proc. Amer. Control Conf.*, pp. 1195–1200, 1988.

[74] P. M. Young and J. C. Doyle, "Computation of the μ with real and complex uncertainties," in *Proc. IEEE Control Decision Conf.*, pp. 1230–1235, 1990.

[75] P. M. Young, M. P. Newlin, and J. C. Doyle, "μ analysis with real parametric uncertainty," in *Proc. IEEE Control Decision Conf.*, 1991.

[76] M. Dahleh, A. Tesi, and A. Vicino, "Extremal properties for the parametric robust performance problem," Tech. Rep. UCSB-ME-91-4, Univ. California, Santa Barbara, Mech. Eng., 1991. also submitted to 30th IEEE CDC.

[77] B. C. Moore, "Principal components analysis in linear systems: controllability, observability and model reduction," *IEEE Trans. Auto. Control*, vol. AC-26, pp. 17–31, 1981.

[78] D. F. Enns, *Model Reduction for Control System Design*. PhD thesis, Stanford University, 1984.

[79] K. Glover, "All optimal Hankel-norm approximations of linear multivariable systems and their L^∞-error bounds," *Int. J. of Control*, vol. 39, no. 6, pp. 1115–1193, 1984.

[80] A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*. New Jersey: Prentice-Hall, 1975.

[81] J. L. Adcock, "Curve fitter for pole-zero analysis," *Hewlett-Packard Journal*, p. 33, January 1987.

[82] M. Safonov, A. Laub, and G. Hartman, "Feedback properties of multivariable systems: The role and use of the return difference matrix," *IEEE Trans. Auto. Control*, vol. 26, no. 1, 1981.

[83] G. Hartman, M. Barrett, and C. Greene, "Control designs for an unstable vehicle," Tech. Rep. NAS 4–2578, NASA Dryden Flight Research Center, 1979.

[84] P. Merkel and R. Whitmoyer, "Development and evaluation of precision control modes for fighter aircraft," in *AIAA Guidance, Navigation and Cont. Conf.*, 1976. Paper No. 76–1950.

[85] J. T. Spanos and M. C. O'Neal, "Nanometer level optical control on the JPL Phase B testbed," in *ADPA/AIAA/ASME/SPIE Conf. Active Mat. & Adapt. Struct.*, Nov 1991.

[86] M. C. O'Neal and J. T. Spanos, "Optical pathlength control in the nanometer regime on the JPL Phase B interferometer testbed," in *SPIE Int. Symp. Optical Appl. Sci. & Eng.*, July 1991.

[87] J. T. Spanos and A. Kissil, "Modeling and identification of the JPL Phase B testbed," in *ADPA/AIAA/ASME/SPIE Conf. Active Mat. & Adapt. Struct.*, Nov 1991.

Chapter 6

Function Reference

6.1 Xμ Functions

The following pages contain descriptions of the Xμ functions. These are also available on-line via the help utility. Each description also gives an illustrative example of the function's use.

The functions are included in alphabetical order. For convenience they are cross-referenced by typical use in the following list.

System building and interconnection

Variable display and graphics

Time response calculations and PDM *functions*

Model reduction and state-space functions

Controller synthesis

μ analysis and D-K iteration

Transfer function fitting

Miscellaneous functions

balmoore

Syntax

```
[SysR,HSV,T] = balmoore(Sys,{nsr,bound})
```

Parameter List

Inputs:	Sys	Linear, stable, minimal state-space system
	nsr	(optional) If `bound` is used then a reduction will be performed which meets an error bound specified by the value in `nsr`, otherwise `nsr` is the order of the reduced system. If `nsr` is not specified, the user will be prompted for its value after the Hankel singular values are displayed.
Keywords:	bound	Boolean; meet an upper bound on the error (see nsr).
Outputs:	SysR	Internally balanced reduced order system; dynamic system object.
	HSV	A column vector containing the Hankel singular values of the system, Sys.
	T	Square matrix containing the balancing transformation, i.e. xbal = Tx.

Description

Computes the balanced form of the system Sys which can be continuous or discrete, then optionally truncates to the desired order, nsr, via B.C. Moore's algorithm.

The user must ensure that the input system is minimal. Any initial state values or state names associated with Sys are assigned to SysR. Input and Output names are also maintained.

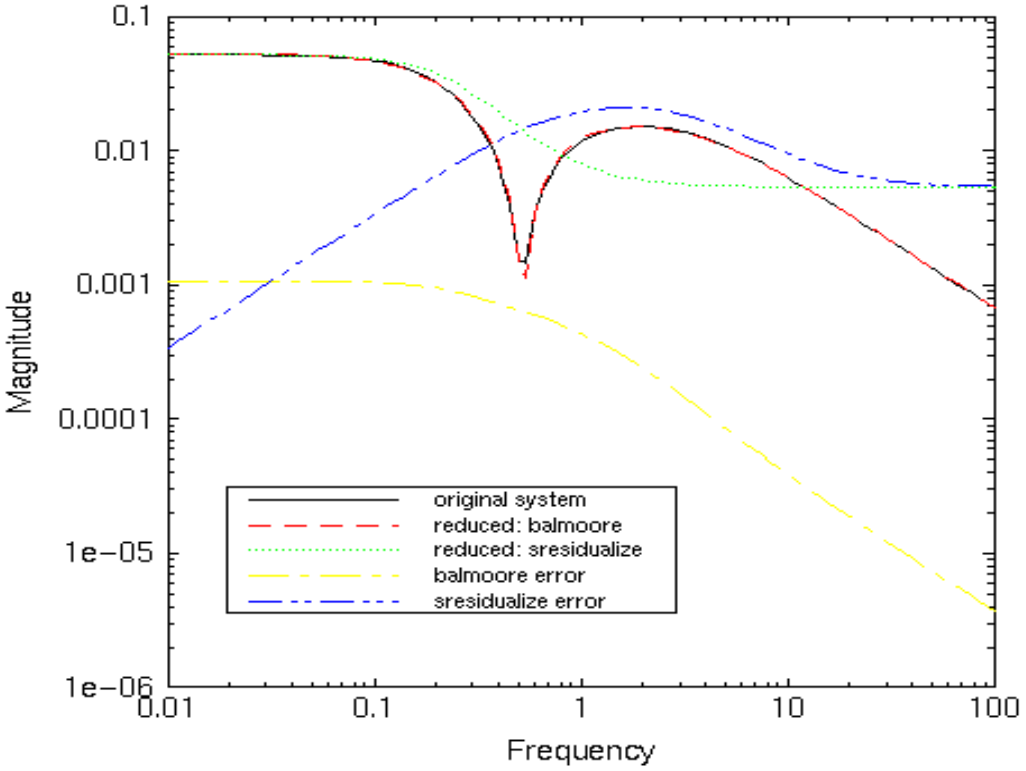
This function is cross-licensed from the Model Reduction Module.

Reference

B.C. Moore, *"Principal Component Analysis in Linear Systems: Controllability, Observability and Model Reduction,"* IEEE Trans. Auto. Ctrl., Vol. 26, No. 1, pp. 17–32, Feb. 1981.

Example

```
# Create a five state system for reduction.

a = daug(-0.891334,[-1.20857,0.799042;-0.799042,-1.20857],...
           -4.74685,-21.3013)
b = [0.0262569;-0.189601;-0.113729;0.211465;-0.538239]
c = [0.120725,-0.336942,0.397198,-0.700524,-1.02235]
d = 0
sys1 = system(a,b,c,d)
fHz = logspace(0.01,100,100)
sys1g = freq(sys1,fHz)

# Reduce to 3 states by balanced truncation.

[sysout1,hsv] = balmoore(sys1,{nsr=3})
sysout1g = freq(sysout1,fHz)
balerr = sys1g - sysout1g

# Reduce to a 3 state system by residualization
# for comparison purposes.

sysout2 = sresidualize(sys1,3)
sysout2g = freq(sysout2,fHz)
residerror = sys1g - sysout2g

gph1 = ctrlplot([sys1g,sysout1g,sysout2g,...
               balerr,residerror],{logmagplot});
gph1 = plot(gph1,{!grid,legend=["original system";...
        "reduced:  balmoore";"reduced:  sresidualize";...
        "balmoore error";"sresidualize error"]})?
```



See Also:

`minimal`, `ophank`.

blknorm

Syntax

```
normM = blknorm(M,blk,p,Frobenius)
```

Parameter List

Inputs:	M	Matrix (or PDM).
	blk	Block structure. See mu section of the manual for a description of the syntax.
	p	Scalar valued. Specifies the Holder "p" norm to be used, where $1 \leq p \leq$ inf. Optional. The default is $p = 2$.
Keywords:	Frobenius	The Frobenius norm is used.
Outputs:	normM	Matrix (or PDM) norms of each block of M.

Description

M is partitioned according the input/output partitions determined by blk. The maximum singular value of each partition is calculated and normA is the matrix (PDM) of norms.

For example, if $p = 2$ (the default case) and blk represents a single perturbation (with compatible input/output dimensions) then normA is simply the maximum singular value. If blk consists of entirely 1×1 perturbations, then normA is equal to abs(A).

Repeated scalar blocks are taken as full blocks. This neglects the assumed equivalences between parts of Delta.

The Xmath function `norm` is used for the calculations. For further detail, refer to `norm`

Examples

```
A = random(3,3)-0.5*ones(3,3)?

A (a square matrix) =

   0.0618661    0.0896177      0.185398
   0.390622     0.00422128    -0.150638
  -0.112622     0.42229        0.448818

blknorm(A,[1,1; 1,1; 1,1])

ans (a square matrix) =

  0.0618661    0.0896177     0.185398
  0.390622     0.00422128    0.150638
  0.112622     0.42229       0.448818

blknorm(A,[3,3])

ans (a scalar) = 0.682429

# compare to the following:
max(svd(A))

ans (a scalar) = 0.682429


B = [1,2,3,4; 5,6,7,-8; 9,10,-11,12]?

B (a rectangular matrix) =

  1     2      3     4
  5     6      7    -8
  9    10    -11    12

blk = [2,1;1,1;1,1]
# examine 1,1 entry of blknorm result
blknorm(B,blk)

ans (a square matrix) =
```

```
   2.23607     3     4
   7.81025     7     8
  13.4536     11    12

# and compare to
norm(B(1,1:2))

ans (a scalar) = 2.23607
```

See Also

norm

conpdm

Syntax

```
outpdm = conpdm(mat,domain,{skipChks})
```

Parameter List

Inputs:	mat	constant matrix
	domain	domain over which outsys will be defined.
Keywords:	skipChks	Boolean specifying that syntax checking is to be skipped.
Outputs:	outpdm	PDM

Description

Creates a PDM data object from a constant matrix. Outpdm represents a constant gain; its value is repeated at every instance of the domain. It is equivalent to,

```
outpdm = pdm(kronecker(ones(length(dom),1),mat),dom)
```

and is a useful shorthand for including constant values in a PDM plot.

This function may be superceded by redefining the augmentation operators and other functions in a later release of Xmath. To maintain future upwards compatibility avoid using this function when developing derivative software.

consys

Syntax

```
outsys = consys(mat,{skipChks})
```

Parameter List

Inputs:	mat	constant matrix
Keywords:	skipChks	Boolean specifying that syntax checking is to be skipped.
Outputs:	outsys	Dynamic System

Description

Creates a Dynamic System object from a constant matrix. Outsys represents a constant gain; the A, B and C matrices are empty. It is equivalent to,

```
outsys = system([],[],[],mat)
```

and can be useful in specifying constant inputs to `freq`.

This function may be superceded by redefining the augmentation operators and other functions in a later release of Xmath. To maintain future upwards compatibility avoid using this function when developing derivative software.

csum

Syntax

```
[outpdm] = csum(inpdm, {channels})
```

Parameter List

Inputs:	inpdm	real or complex valued PDM or constant matrix
Keywords:	channels	Sum over channels. outpdm has the same dimensions as inpdm.
Outputs:	outpdm	output PDM

Description

Perform a cumulative sum over the rows of a matrix or a PDM. If channels is specified then the sum is performed over the domain of the PDM.

Examples

```
A = [ones(6,1),random(6,1)]?

A (a rectangular matrix) =

  1    0.608453
  1    0.854421
  1    0.0642647
  1    0.827908
  1    0.926234
  1    0.566721

csum(A)
```

```
ans (a rectangular matrix) =

  1    0.608453
  2    1.46287
  3    1.52714
  4    2.35505
  5    3.28128
  6    3.848


pdmA = pdm(A,[1,2,3])?
pdmA (a pdm) =
domain |        Col 1  Col 2
-------+-----------------------
     1 | Row 1  1      0.608453
       | Row 2  1      0.854421
-------+-----------------------
     2 | Row 1  1      0.0642647
       | Row 2  1      0.827908
-------+-----------------------
     3 | Row 1  1      0.926234
       | Row 2  1      0.566721
-------+-----------------------

csum(pdmA)
ans (a pdm) =
domain |        Col 1  Col 2
-------+-----------------------
     1 | Row 1  1      0.608453
       | Row 2  2      1.46287
-------+-----------------------
     2 | Row 1  1      0.0642647
       | Row 2  2      0.892173
-------+-----------------------
     3 | Row 1  1      0.926234
       | Row 2  2      1.49296
-------+-----------------------

csum(pdmA,channels)
```

```
ans (a pdm) =
domain |        Col 1  Col 2
-------+-----------------------
     1 | Row 1  1      0.608453
       | Row 2  1      0.854421
-------+-----------------------
     2 | Row 1  2      0.672717
       | Row 2  2      1.68233
-------+-----------------------
     3 | Row 1  3      1.59895
       | Row 2  3      2.24905
-------+-----------------------
```

ctrlplot

Syntax

```
graph = ctrlplot(pdm,old_graph,{keywords})
```

Parameter List

Inputs:	pdm	Pdm (or matrix) containing the data to be plotted.
	old_graph	(optional) Graphical object to which data is added. Conceptually the same as `plot(pdm,{keep=old_graph})`.

Keywords:

The following keywords specify the basic plot format. Only one can be selected.

timeresp	(default) real(pdm) vs. domain. This is the same as the default `plot` function and is suitable for time domain responses
bode	Two subplots are generated: log magnitude vs. domain and angle vs. domain. They are positioned one above the other. If present, old_graph must also be in this format.
nyquist	imag(pdm) vs. real(pdm). Standard Nyquist plot.
nichols	log magnitude vs. angle. Standard Nichols chart.
logmagplot	log magnitude vs. domain.
phaseplot	angle vs. domain.

The following keywords specify whether the domain is log or linear scale. This is not applicable to the Nyquist or Nichols plots. The defaults depend upon which of the above control plot types has been selected.

	linear	linear domain. Default = 1 for timeresp keyword. Default = 0 for bode keyword.
	log	logarithmic domain. Default = 1 for bode keyword. Default = 0 for timeresp keyword.

Default units can be supplied for the magnitude and phase plots (bode, nichols, logmagplot and phaseplot keywords) with the following keywords.

	degrees	Angles are specified in degrees (default = 1)
	db	log magnitudes are specified in decibels (default = 0).

The following keywords behave identically to those in `plot`. They relate directly to the line/marker specifications and must be associated with a particular pdm.

	line	(default = 1). Plot as a line type.
	marker	(default = 0). Plot markers.
	line_style	refer to `plot`
	line_width	refer to `plot`
	marker_style	refer to `plot`
	marker_size	refer to `plot`

Some preprocessing of the data is performed by the following keyword.

	unwrap	Applicable to bode, nichols and phaseplot keywords. Phase changes of 2π are unwrapped, rather than being graphed between $-\pi$ and π. Default = 1.
Outputs:	graph	Resulting graphical object.

Description

This function performs some common control system related plotting. The user can use ctrlplot to set up a basic plot and perform some preprocessing of the data. This generates a graphical object containing the data and the user can perform subsequent calls to `plot` to add things like text, labels, gridding etc. The second argument (optional) is a graphical object to which the pdm will be added.

Plots over a domain (bode, timeresp, logmagplot, phaseplot) must be called with a pdm. A matrix or scalar can be turned into a suitable pdm with the `conpdm` function. Nyquist and Nichols plots can also plot scalars. Points such as -1 are often useful. The user should specify a marker so that such points show up. When mixing scalar and pdm data on a Nyquist or Nichols plot, plot the pdm data first to get a reasonable choice of axes.

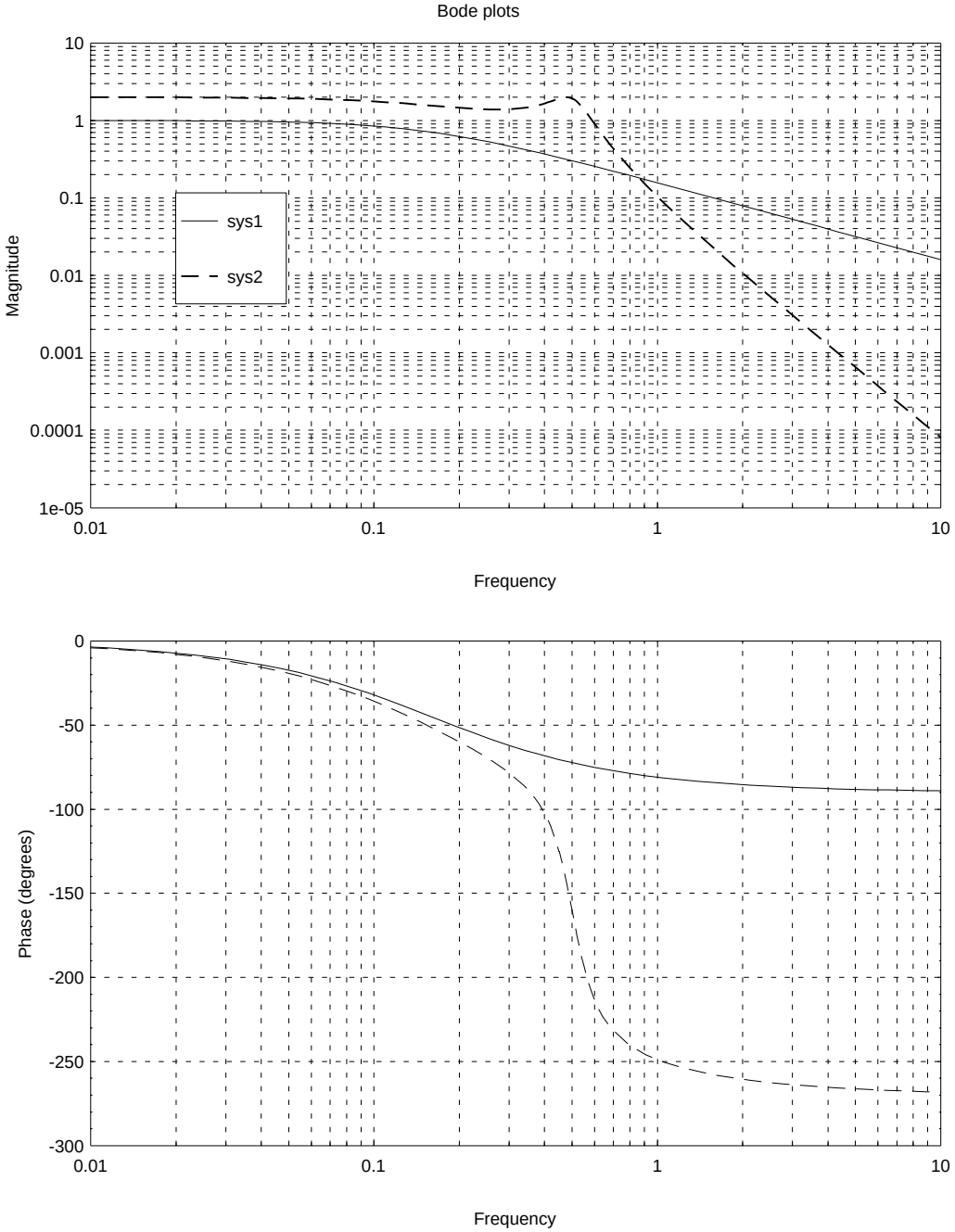
For Bode plots, two graphs are created — the magnitude and the phase plots — and positioned one above the other. The result is returned as a single graphical object. Similarly, when adding data to Bode plots, the existing graphical object must also contain two subplots.

Default labels, corresponding to the most common use of the particular invocation, are put on the axes. These can be overwritten with subsequent calls to `plot`.

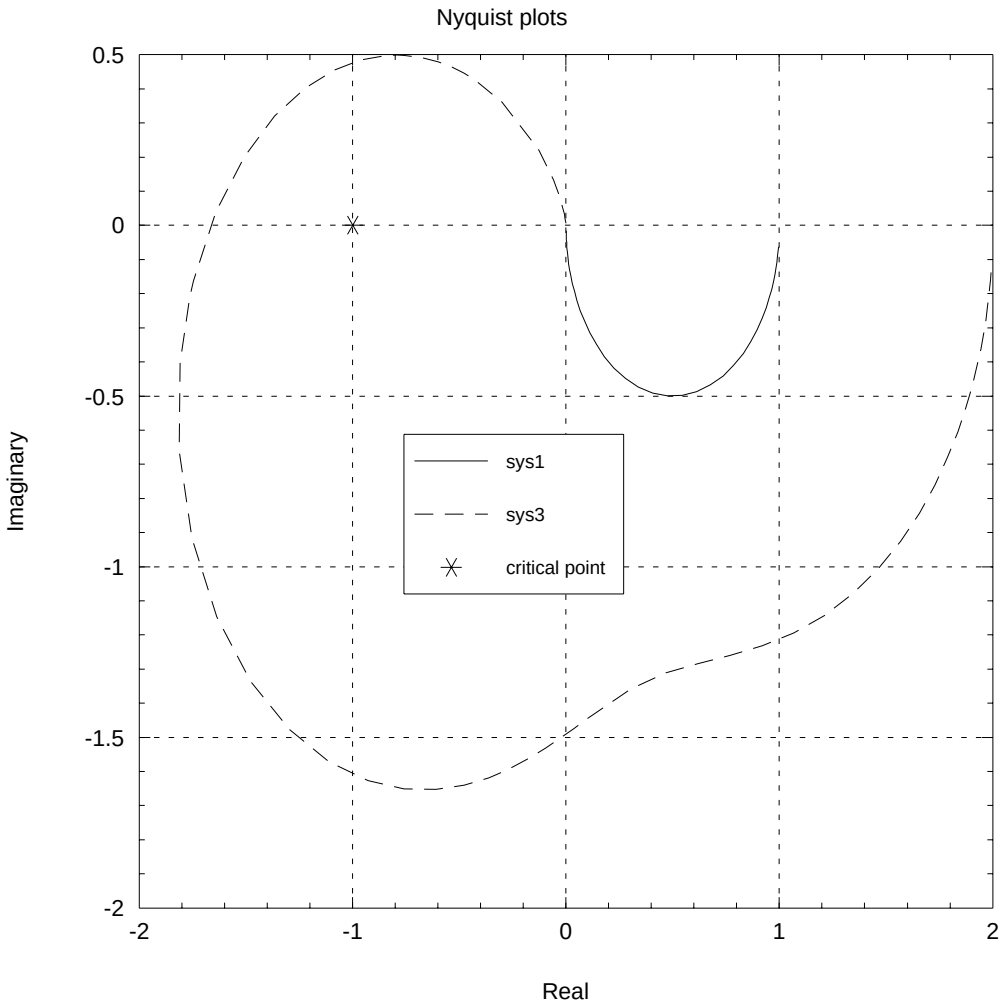
Examples:

```
# Create 2 systems.

sys1 = 1/makepoly([1,1],"s")
sys2 = 2*sys1*10/makepoly([1,1,10],"s")

w1 = logspace(0.01,10,50)'
w2 = sort([w1;[0.35:0.01:0.65]'])
sys1g = freq(sys1,w1)
sys2g = freq(sys2,w2)

# Bode plots

g1 = ctrlplot(sys1g,{bode});
g1 = ctrlplot(sys2g,g1,{bode});
g1 = plot({keep=g1,title = "Bode plots",...
```

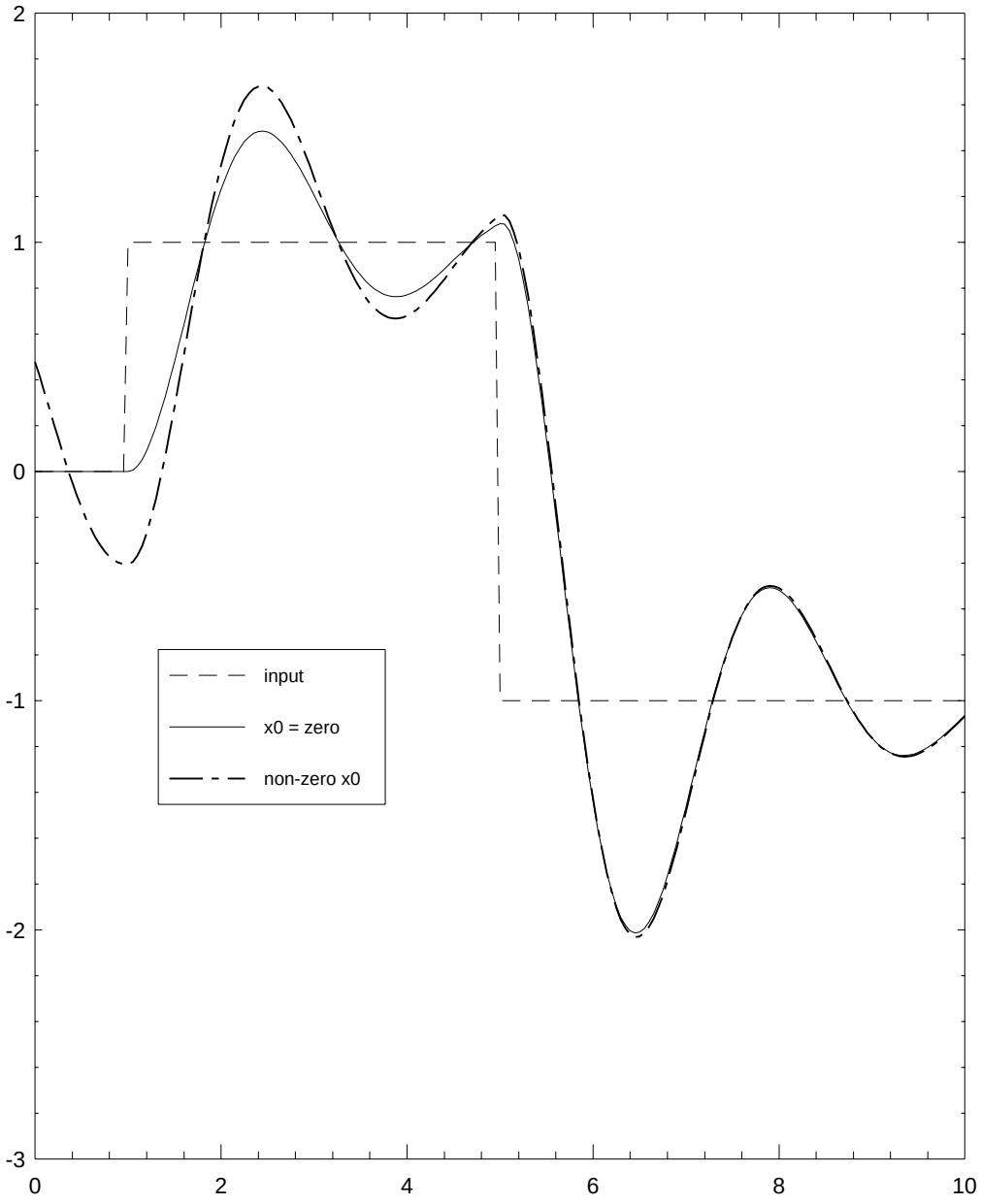
```
legend = ["sys1","sys2"]})?
```


Bode plots
Magnitude
Frequency
sys1
sys2
10
1
0.1
0.01
0.001
0.0001
1e-05
0.01
0.1
1
10
Phase (degrees)
Frequency
0
-50
-100
-150
-200
-250
-300

```
# Nyquist plots
g2 = ctrlplot(sys1g,{nyquist});
g2 = ctrlplot(sys2g,g2,{nyquist});
g2 = ctrlplot(-1,g2,{nyquist,marker=1,line=0});
g2 = plot(g2,{projection="orthographic",...
     legend=["sys1","sys3","critical point"],title="Nyquist plots"})?
```


Nyquist plots
0.5
0
-0.5
-1
-1.5
-2
-2
-1
0
1
2
Imaginary
Real
sys1
sys3
critical point

```
# Create a second order lightly damped system to illustrate
# time response plotting.  The calculation is repeated with
# a non-zero initial condition.

sys = 5/makepoly([1,1,5],"s")
u = gstep([0:0.05:10],[0;1;5],[0;1;-1])
y0 = sys*u
[a,b,c,d] = abcd(sys)
sys = system(a,b,c,d)
y1 = system(sys,{X0=[-1;0]})*u

# Now plot the result

g1 = ctrlplot(u,{line_style=2});
g1 = ctrlplot(y0,g1,{line_style=1});
g1 = ctrlplot(y1,g1,{line_style=4});
g1 = plot(g1,{!grid,legend=["input";"x0 = zero";"non-zero x0"]})?
```


2
1
0
-1
-2
-3
0
2
4
6
8
10
input
x0 = zero
non-zero x0

See Also:

`plot`.

daug

Syntax

```
out = daug (sys1,sys2,...)
```

Parameter List

Inputs:	sys1	Input systems. These can be dynamical systems and constants, or pdms and constants.
	⋮	"
Outputs:	out	output system.

Description

Diagonal augmentation of dynamical system/pdm/constant, matrices.

$$\texttt{out} = \begin{bmatrix} \texttt{sys1} & 0 & \dots & 0 \\ 0 & \texttt{sys2} & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \texttt{sysN} \end{bmatrix}$$

Limitations

Only 21 systems can be augmented with a single function invocation.

Examples

```
daug([1,1],[2;2],inf)
```

```
ans (a square matrix) =

  1    1    0      0
  0    0    2      0
  0    0    2      0
  0    0    0    Inf

sys1 = randsys(1,1,2,{stable})
sys1 = system(sys1,{statenames="sys1state"})?

sys1 (a state space system) =

  A
  -0.886949

  B
  0.853282    0.012459

  C
  0.186754

  D
  0.492058    0.748961

  X0
  0

  State Names
  -----------
  sys1state

  System is continuous

sys2 = randsys(1,2,1,{stable})
sys2 = system(sys2,{statenames="sys2state",x0=1})?

sys2 (a state space system) =

  A
```

```
  -1.67106

  B
  0.579502

  C
  0.262815
  0.436099

  D
  0.911055
  0.808267

  X0
  1

  State Names
  -----------
  sys2state

  System is continuous

# Note the effect of the constant in the following
daug(sys1,10,sys2)

ans (a state space system) =

  A
  -0.886949     0
   0           -1.67106

  B
  0.853282    0.012459    0    0
  0           0           0    0.579502

  C
  0.186754    0
  0           0
  0           0.262815
  0           0.436099
```

```
  D
  0.492058    0.748961     0    0
  0           0           10    0
  0           0            0    0.911055
  0           0            0    0.808267

  X0
  0
  1

  State Names
  -----------
  sys1state    sys2state

  System is continuous


pdm1 = randpdm(3,2,2)?

pdm1 (a pdm) =

domain |        Col 1     Col 2
-------+--------------------------
     0 | Row 1  0.810265  0.259043
       | Row 2  0.413909  0.359993
-------+--------------------------
     1 | Row 1  0.691279  0.765686
       | Row 2  0.357265  0.76934
-------+--------------------------
     2 | Row 1  0.547763  0.0962289
       | Row 2  0.956117  0.220741
-------+--------------------------

pdm2 = pdm(10*ones(3,1),domain(pdm1))?

pdm2 (a pdm) =

domain |
-------+-----
```

```
     0 |   10
-------+-----
     1 |   10
-------+-----
     2 |   10
-------+-----
```

```
daug(pdm1,pdm2)

ans (a pdm) =
```

domain		Col 1	Col 2	Col 3
0	Row 1	0.810265	0.259043	0
	Row 2	0.413909	0.359993	0
	Row 3	0	0	10
1	Row 1	0.691279	0.765686	0
	Row 2	0.357265	0.76934	0
	Row 3	0	0	10
2	Row 1	0.547763	0.0962289	0
	Row 2	0.956117	0.220741	0
	Row 3	0	0	10

delsubstr

Syntax

```
[outstr] = delsubstr(str,charstr)
```

Parameter List

Inputs:	str	String or vector of strings.
	charstr	String
Outputs:	outstr	String or vector of strings.

Description

All occurences of the substring, `charstr`, within `str` are deleted.

If, by deleting `charstr`, another occurence of charstr in created, it will not be deleted. Examine the second example closely to see the effect of this.

Unless `str` is a scalar string, deleting a whole string element will cause an error.

Example

```
strvec = ["string one";"aaa";"xxyy"]
out1 = delsubstr(strvec,"g o")?

out1 (a column vector of strings) =

  strinne
  aaa
  xxyy

out2 = delsubstr(strvec,"xy")?
```

```
out2 (a column vector of strings) =

  string one
  aaa
  xy

# If executed, the following would give an error
# out3 = delsubstr(strvec,"a")?
```

fitsys

Syntax

```
[sys] = fitsys(data,npoles,nzeros,weight, {skipchks,Hertz})
```

Parameter List

Inputs:	data	Complex valued data (PDM).
	npoles	Order of requested fit. (optional, default = 0).
	nzeros	Number of zeros in transfer function. (optional, default = npoles)
	weight	Weighting function. (scalar, PDM, or DYNAMIC SYSTEM) (optional, default = 1).
Keywords:	Hertz	Boolean. This keyword is mandatory as the function must know whether the domain is in Hertz or radians/second (specified by !Hertz) to fit correctly. Note that the Xmath function `freq` assumes that the frequency range is specified in Hertz.
	skipchks	Boolean. Skip the error checking. (Default = 0)
Outputs:	sys	Dynamic system, order = npoles.

Description

Fits a transfer function to complex valued data. npoles and nzeros specifies the number of poles and zeros.

The optional argument weight, specifies a weighting for the fit. If weight is a PDM it must be over the same domain as data. It may also be a DYNAMIC SYSTEM, in which case the magnitude of its frequency response is the weight. A scalar weight may also be specified although this will have no effect. For logscale frequency data a weight of close to $1/s$ is strongly recommended.

The primary use of this routine is the fitting of D scale weights for mu synthesis iterations.

Chebyshev polynomials are used as basis functions for both the numerator and denominator polynomials.

WARNING: This routine uses iterative polynomial calculations which are not well conditioned for high order (> 6) fits.

Reference

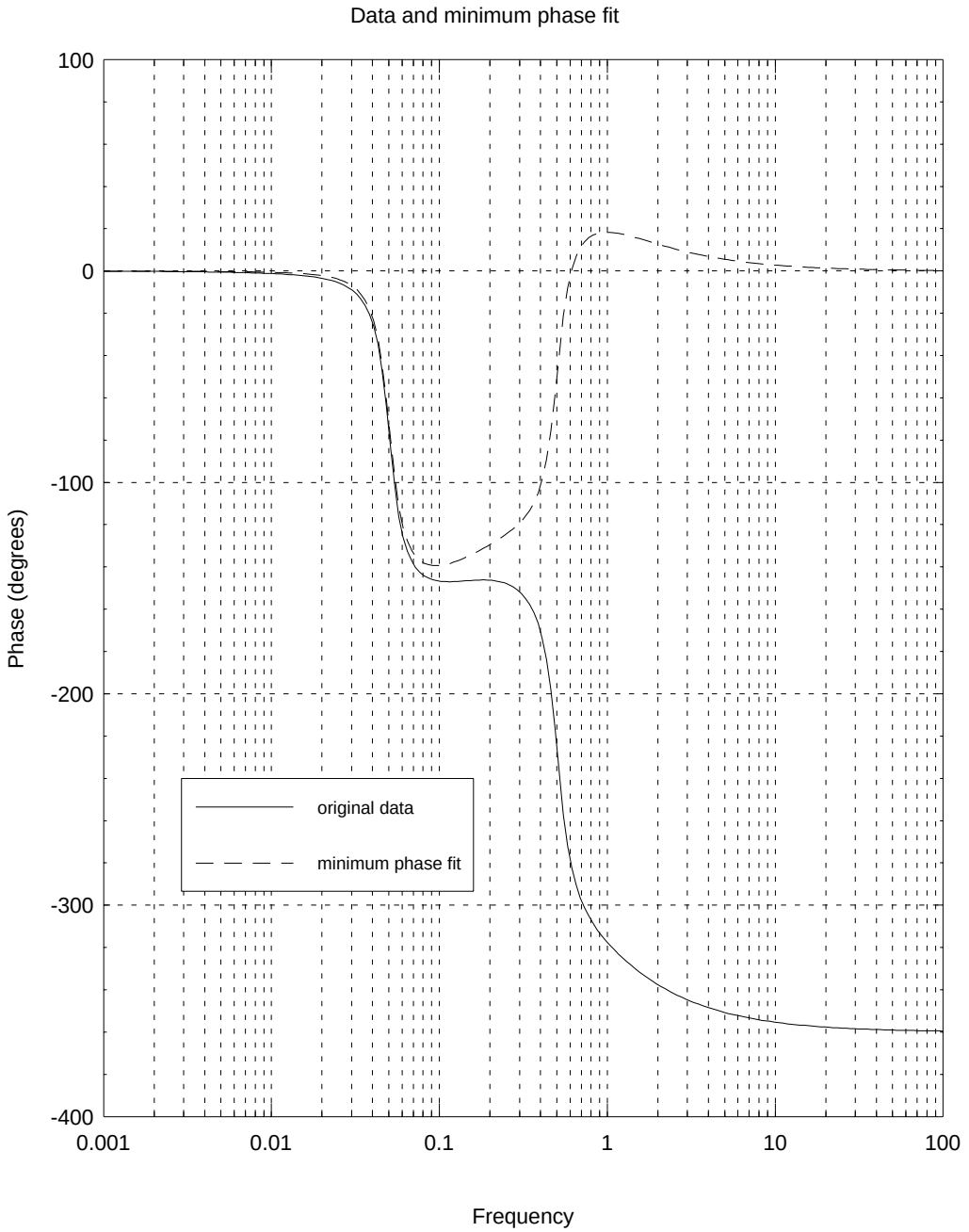
For further information see: *"Curve Fitter for Pole-Zero Analysis,"* J.L. Adcock, Hewlett-Packard Journal, p. 33, January 1987.

Example

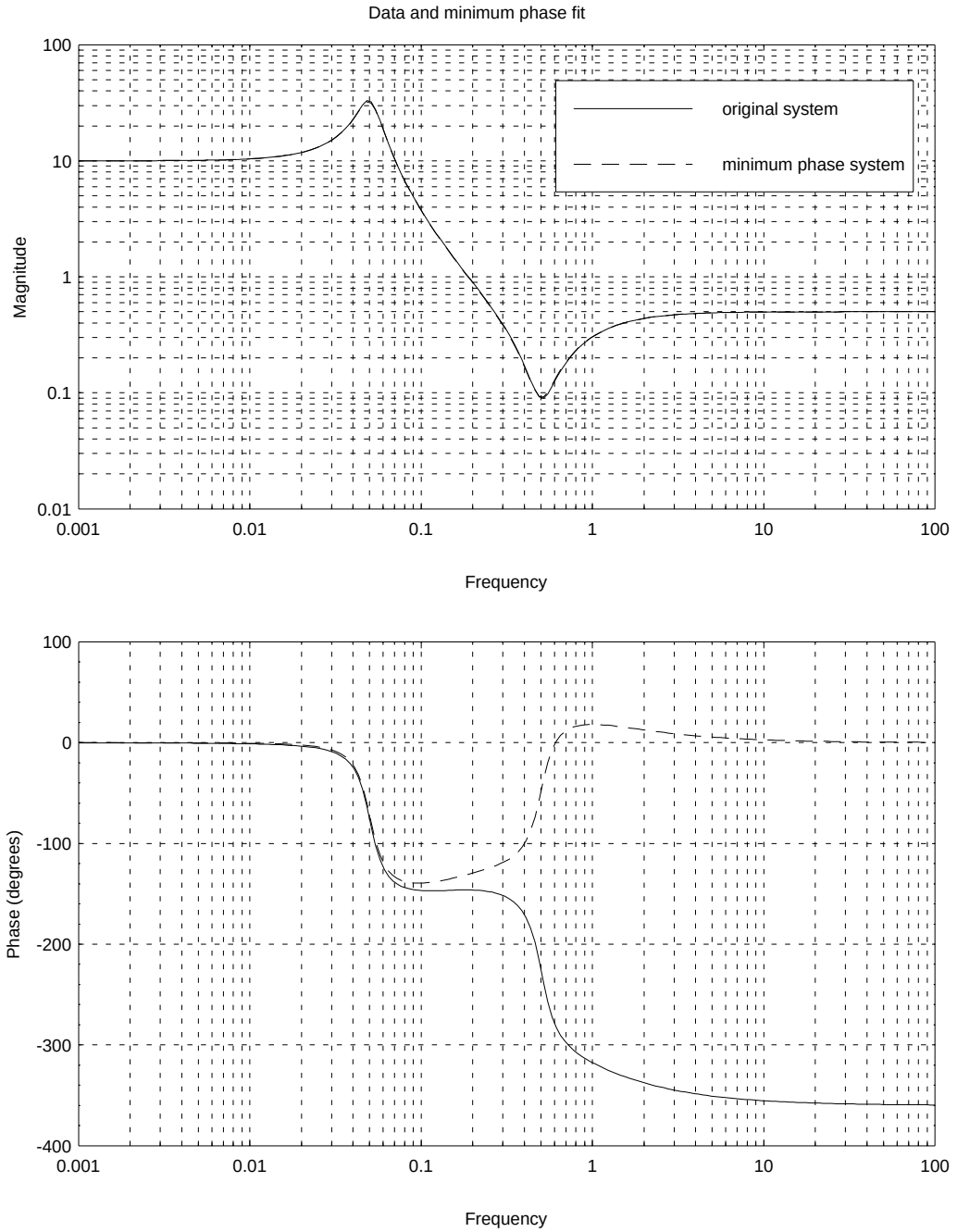
```
# Set up a plant to generated data for the fitting
# problem.

plant = makepoly([0.1,-0.1,1],"s")*makepoly([1,1],"s")...
       /(makepoly([1,0.1,.1],"s")*(makepoly([0.2,1],"s")))

# Note that plant has right half plane zeros

rifd(plant)

Poles:

      real        imaginary      frequency      damping
                                 (rad/sec)       ratio

  -5.0000e-02     3.1225e-01     3.1623e-01     0.1581
  -5.0000e-02    -3.1225e-01     3.1623e-01     0.1581
  -5.0000e+00     0.0000e+00     5.0000e+00     1.0000


Zeros:
```

```
      real        imaginary      frequency      damping
                                 (rad/sec)       ratio

  -1.0000e+00     0.0000e+00     1.0000e+00     1.0000
   5.0000e-01     3.1225e+00     3.1623e+00    -0.1581
   5.0000e-01    -3.1225e+00     3.1623e+00    -0.1581
```

```
omega = logspace(0.001,100,200)
plantg = freq(plant,omega)

# Use complex cepstrum to fit minimum phase equivalent
# to the magnitude of the data.  One of the principle
# uses of the fitsys function is fitting approximations
# to noisy data.  To illustrate the concepts, no noise
# is added here.

cdata = mkphase(abs(plantg))

gph1 = ctrlplot([plantg,cdata],{phaseplot});
gph1 = plot(gph1,{title="Data and minimum phase fit",...
            legend=["original data";"minimum phase fit"]})?
```


Data and minimum phase fit
100
0
-100
-200
-300
-400
Phase (degrees)
0.001
0.01
0.1
1
10
100
Frequency
original data
minimum phase fit

```
# Create fitting weight.  1/s works well for logspaced
# data.

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

# Fit new system and compare pole location with
# the original.  Note that it is minimum phase.

nsys = fitsys(cdata,3,3,wght)
rifd(nsys)


Poles:

      real        imaginary      frequency      damping
                                 (rad/sec)       ratio

  -5.1043e-02    -3.1249e-01     3.1663e-01     0.1612
  -5.1043e-02     3.1249e-01     3.1663e-01     0.1612
  -5.0107e+00     0.0000e+00     5.0107e+00     1.0000


Zeros:

      real        imaginary      frequency      damping
                                 (rad/sec)       ratio

  -1.0189e+00     0.0000e+00     1.0189e+00     1.0000
  -5.0906e-01    -3.0989e+00     3.1405e+00     0.1621
  -5.0906e-01     3.0989e+00     3.1405e+00     0.1621


nsysg = freq(nsys,omega)
gph2 = ctrlplot([plantg,nsysg],{bode});
gph2 = plot(gph2,{title="Data and minimum phase fit",...
          legend=["original system";"minimum phase system"]})?
```


Data and minimum phase fit
original system
minimum phase system
Magnitude
Frequency
Phase (degrees)
Frequency

Limitations

Limited to SISO systems.

See Also

`tfid`

gstep

Syntax

```
gPdm = gstep (ytime,timespec,valspec, {skipChks})
```

Parameter List

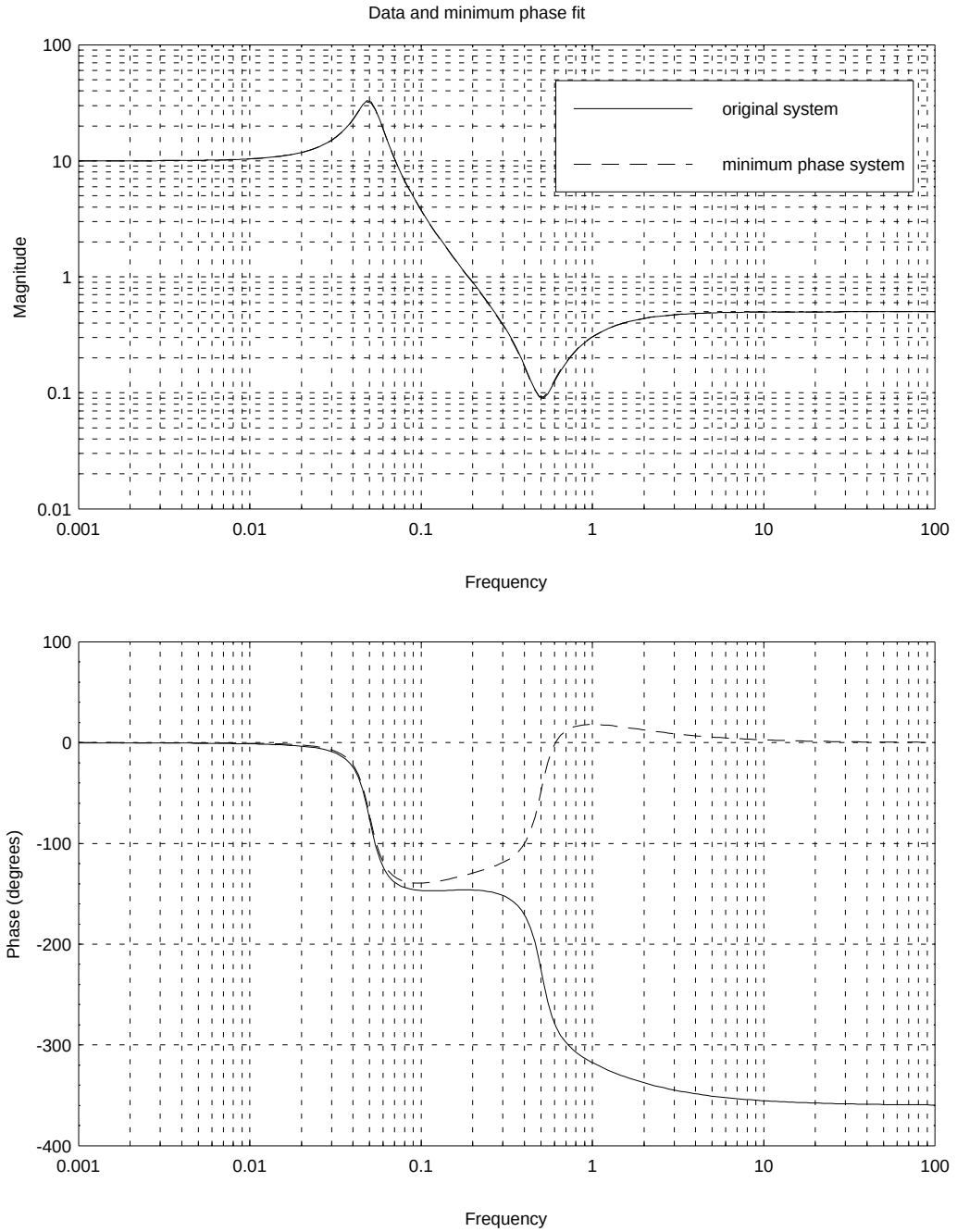
Inputs:	ytime	output time vector (seconds).
	timespec	times for specified step data (optional)
	valspec	value for specified step data (optional)
Keywords:	skipChks	Boolean specifying that syntax checking is to be skipped.
Outputs:	gPdm	PDM containing the step values as a function of time.

Description

This function creates a PDM over the domain: ytime. At ytime = timespec(i) the output steps to value: valspec(i) and maintains that value until the next specified step or the end of the domain.

Example

```
time = [0:100]
steptimes = [5,25,30,65,90]
stepvalues = [-1,2,1,-1.5,1.5]
out = gstep(time,steptimes,stepvalues)
gph1 = ctrlplot(out);
gph1 = plot(gph1,{title="gstep example"})?
```


Data and minimum phase fit
100
10
1
0.1
0.01
Magnitude
original system
minimum phase system
0.001
0.01
0.1
1
10
100
Frequency
100
0
-100
-200
-300
-400
Phase (degrees)
0.001
0.01
0.1
1
10
100
Frequency

See Also

randpdm, gcos, gsin, gpulse, gsawtooth, gsquarewave

hinfnorm

Syntax

```
[out,omega] = hinfnorm(sys,tol,{imag_eps,max_it})
```

Parameter List

Inputs:	sys	DYNAMIC SYSTEM, frequency response (PDM), or constant gain (matrix).
	sys	Specifies the relative tolerance of the answer when the input is a DYNAMIC SYSTEMDefault = 0.001.
Keywords:	imag_eps	Epsilon value for determining imaginary eigenvalues of the Hamiltonian. Default = sqrt(eps).
	max_it	Maximum number of iterations. Default = 100.
Outputs:	out	H_∞ norm of the input system. Scalar or pdm or matrix input systems. 2×1 vector for DYNAMIC SYSTEM inputs. An upper bound of inf indicates that the maximum number of iterations was exceeded.
	omega	Frequency (Hz) where the norm is achieved.

Description

The H_∞ norm is defined to be the supremum, over frequency, of the maximum singular value of the system's frequency response.

If the input system is a PDM, it is assumed to be a frequency response and the norm is calculated only from the frequencies provided.

If the inputs system is a matrix, it represents a constant gain and the maximum singular value of the matrix is returned. Note that this is NOT the same as the result of norm(sys,inf).

Stable DYNAMIC SYSTEM norms are calculated by an iterative Hamiltonian method. In this case out is a 2×1 vector with upper and lower bounds for the norm.

Example

```
# Set up a simple closed loop problem.
# This example is given in more detail in the
# hinfsyn online help.

plant = makepoly([0.1,-0.1,1],"s")*makepoly([1,1],"s")...
        /(makepoly([1,0.1,.1],"s")*(makepoly([0.2,1],"s")))

# Create weights

Wperf = 100/makepoly([100,1],"s")
Wact = makepoly([0.5,0.05],"s")/makepoly([0.05,1],"s")

# Form the weighted interconnection structure

sysnames = ["plant";"Wperf";"Wact"]
sysinp = ["ref";"control"]
sysout = ["Wperf"; "Wact"; "ref-plant"]
syscnx = ["control"; ...          # input to plant
          "ref-plant"; ...        # input to Wperf
          "control"]              # input to Wact

wghtic = sysic(sysnames,sysinp,sysout,syscnx,plant,...
               Wperf,Wact)

# Design Hinf controller

nctrls = 1
nmeas = 1
gmax = 25
gmin = 0
Kinf = hinfsyn(wghtic,nmeas,nctrls,[gmax;gmin])

Test bounds:      0.0000  <  gamma  <=     25.0000
  gamma      Hx_eig     X_eig    Hy_eig      Y_eig   nrho_xy   p/f
```

```
   25.000     5.2e-01   1.7e-03   1.0e-02   0.0e+00   0.0000    p
   12.500     5.2e-01   1.7e-03   1.0e-02   0.0e+00   0.0000    p
    6.250     5.2e-01   1.7e-03   1.0e-02   0.0e+00   0.0000    p
    3.125     5.1e-01   1.7e-03   1.0e-02   0.0e+00   0.0000    p
    1.562     5.0e-01   1.7e-03   1.0e-02   0.0e+00   0.0000    p
    0.781     3.9e-01  -9.8e+01   1.0e-02   0.0e+00   0.0000    f
    1.172     4.8e-01   1.8e-03   1.0e-02   0.0e+00   0.0000    p

 Gamma value achieved:     1.1719

# form weighted closed loop system

clpinf = starp(wghtic,Kinf)

# Compare hinfnorm to gamma value.  The H_infinity
# norm should be less than or equal to gamma.

hinfnorm(clpinf)?

ans (a column vector) =

  1.1701
  1.16893
```

See Also

h2norm, hinfsyn, h2syn.

h2norm

Syntax

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

Parameter List

Inputs:	sys	Continuous time Dynamic System
Outputs:	out	H_2 norm of the input system

Description

The H_2 norm of a stable, strictly proper system is calculated. This is given by

$$out = \text{trace}(CXC'),$$

where X is the controllability grammian, solving the Lyapunov equation,

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

Example

```
# Set up a simple closed loop problem.
# This example is also studied in the
# hinfsyn on-line help.

plant = makepoly([0.1,-0.1,1],"s")*makepoly([1,1],"s")...
        /(makepoly([1,0.1,.1],"s")*(makepoly([0.2,1],"s")))

# Create weights
```

```
Wperf = 100/makepoly([100,1],"s")
Wact = makepoly([0.5,0.05],"s")/makepoly([0.05,1],"s")

# Form the weighted interconnection structure

sysnames = ["plant";"Wperf";"Wact"]
sysinp = ["ref";"control"]
sysout = ["Wperf"; "Wact"; "ref-plant"]
syscnx = ["control"; ...        # input to plant
          "ref-plant"; ...      # input to Wperf
          "control"]            # input to Wact

wghtic = sysic(sysnames,sysinp,sysout,syscnx,plant,...
                Wperf,Wact)

# Design H2 controller

nctrls = 1
nmeas = 1
K2 = h2syn(wghtic,nmeas,nctrls)

# Form the weighted closed loop system and calculate
# its H2 norm.

wghtclp2 = starp(wghtic,K2)
h2norm(wghtclp2)?

ans (a scalar) =   2.78655
```

See Also

h2syn, hinfsyn, hinfnorm

hinfsyn

Syntax

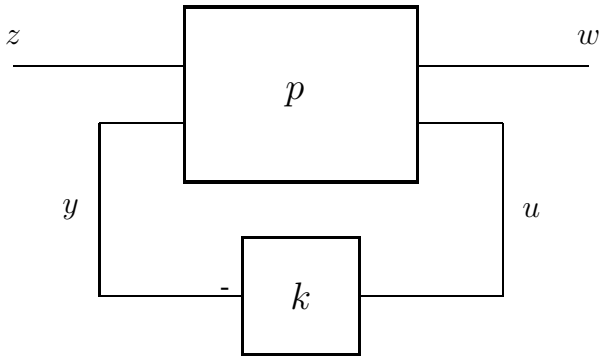
```
[k,gfin,stat] = hinfsyn(p,nmeas,ncon,gamma,{keywords})
```

Parameter List

Inputs:	p	Generalized interconnection structure (DYNAMIC SYSTEM)
	nmeas	measurement vector dimension.
	ncon	control vector dimension.
	gamma	H_∞ norm bound of controller. For a bisection search specify gamma = [gamma_min;gamma_max].
Keywords:	schur_solution	real Schur decomposition for Riccati solution (default)
	eig_solution	eigendecomposition for Riccati solution.
	tol	tol=value. Specifies the relative tolerance for stopping a bisection fit. Default = (gamma_max - gamma_min)/50
	epr	epr = value. Tolerance for determining when the Hamiltonian eigenvalues lie on the $j\omega$ axis. Default = 0.5*sqrt(eps)
	epp	epp = value. Tolerance for determining that a Riccati equation solution is positive definite. Default = 1e-6
	maxit	maxit = value. Maximum number of bisection iterations. Default = inf.
Outputs:	k	Central H_∞ optimal controller.
	gfin	H_∞ norm achieved for the returned controller.
	stat	return status 0 Controller calculated -1 no controller exists for specified gamma value.

Description

The H_∞ (sub)optimal controller for the interconnection, p, is calculated. The resulting closed loop system is illustrated below.



The variables `ncon` and `nmeas` are used to specify the dimensions of u and y in the above diagram ($ncon = \dim(u)$ and $nmeas = \dim(y)$). The objective is to design a stabilizing controller, k, which minimizes the H_2 norm of the closed loop system between w and z.

The closed loop system can be formed with the command,

`clpsys` = `starp`(`p`, `k`).

p is a state-space system, which can be partitioned with respect to $[z;\ y]$ and $[w;\ u]$ in the following way.

$$p = \left[\begin{array}{c|cc} a & b_1 & b_2 \\ \hline c_1 & d_{11} & d_{12} \\ c_2 & d_{21} & d_{22} \end{array}\right].$$

The following assumptions must hold:

1. (a, b_2, c_2) is stabilizable and detectable
2. d_{12} and d_{21} have full rank
3. The matrix $[a - j\omega I, b_2; c_1, d_{12}]$ has full column rank for all ω
4. The matrix $[a - j\omega I, b_1; c_2, d_{21}]$ has full row rank for all ω

Reference

This function uses the state-space formulae given in:
"State-space formulae for all stabilizing controllers that satisfy an H_∞ norm bound and relations to risk sensitivity," Keith Glover and John Doyle, Systems & Control Letters **11**, pp. 167–172., Oct, 1988.

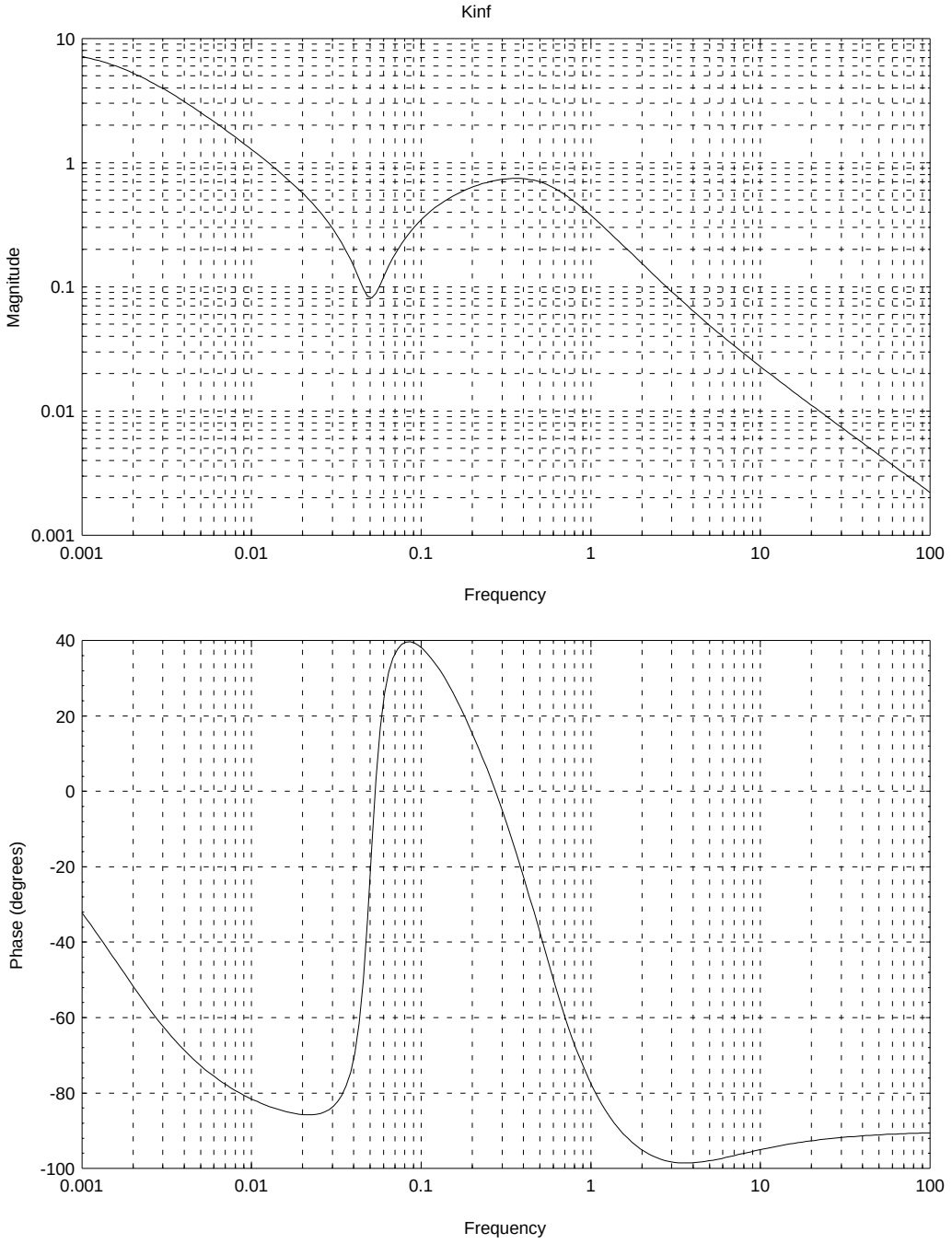
Example

```
# Set up a simple closed loop problem.
# A tracking problem is chosen.  Weights are used to
# trade off between tracking performance and actuator
# effort.

plant = makepoly([0.1,-0.1,1],"s")*makepoly([1,1],"s")...
       /(makepoly([1,0.1,.1],"s")*(makepoly([0.2,1],"s")))

# Create weights (performance & actuator)

Wperf = 100/makepoly([100,1],"s")
Wact = makepoly([0.5,0.05],"s")/makepoly([0.05,1],"s")

# Form the weighted interconnection structure

sysnames = ["plant";"Wperf";"Wact"]
sysinp = ["ref";"control"]
sysout = ["Wperf"; "Wact"; "ref-plant"]
syscnx = ["control"; ...      # input to plant
         "ref-plant"; ...     # input to Wperf
         "control"]           # input to Wact
```

```
wghtic = sysic(sysnames,sysinp,sysout,syscnx,plant,...
              Wperf,Wact)

# Design Hinf controller

nctrls = 1
nmeas = 1
gmax = 25
gmin = 0
Kinf = hinfsyn(wghtic,nmeas,nctrls,[gmax;gmin])

Test bounds:      0.0000  <  gamma  <=     25.0000
  gamma      Hx_eig     X_eig    Hy_eig      Y_eig   nrho_xy   p/f
   25.000     5.2e-01   1.7e-03   1.0e-02   0.0e+00   0.0000    p
   12.500     5.2e-01   1.7e-03   1.0e-02   0.0e+00   0.0000    p
    6.250     5.2e-01   1.7e-03   1.0e-02   0.0e+00   0.0000    p
    3.125     5.1e-01   1.7e-03   1.0e-02   0.0e+00   0.0000    p
    1.562     5.0e-01   1.7e-03   1.0e-02   0.0e+00   0.0000    p
    0.781     3.9e-01  -9.8e+01   1.0e-02   0.0e+00   0.0000    f
    1.172     4.8e-01   1.8e-03   1.0e-02   0.0e+00   0.0000    p

 Gamma value achieved:     1.1719

rifd(Kinf)

Poles:

      real         imaginary      frequency      damping
                                  (rad/sec)       ratio

  -1.0000e-02     0.0000e+00     1.0000e-02     1.0000
  -1.0347e+00     0.0000e+00     1.0347e+00     1.0000
  -2.6846e+00     2.3017e+00     3.5362e+00     0.7592
  -2.6846e+00    -2.3017e+00     3.5362e+00     0.7592
  -1.2692e+01     0.0000e+00     1.2692e+01     1.0000


Zeros:
```

```
      real        imaginary      frequency      damping
                                 (rad/sec)       ratio

  -5.0000e-02    -3.1225e-01     3.1623e-01     0.1581
  -5.0000e-02     3.1225e-01     3.1623e-01     0.1581
  -5.0000e+00     0.0000e+00     5.0000e+00     1.0000
  -2.0000e+01     0.0000e+00     2.0000e+01     1.0000
```

```
omega = logspace(0.001,100,200)
Kinfg = freq(Kinf,omega)
gph1 = ctrlplot(Kinfg,{bode});
gph1 = plot(gph1,{title="Kinf"})?
```


Kinf
Magnitude
10
1
0.1
0.01
0.001
0.001
0.01
0.1
1
10
100
Frequency
Phase (degrees)
40
20
0
-20
-40
-60
-80
-100
0.001
0.01
0.1
1
10
100
Frequency

```
# Use sysic to create unweighted interconnection

ic = sysic("plant",["ref";"ctrl"],["plant";"ref-plant"],...
             "ctrl",plant)
clpinf = starp(ic,Kinf)
rifd(clpinf)
```

Poles:

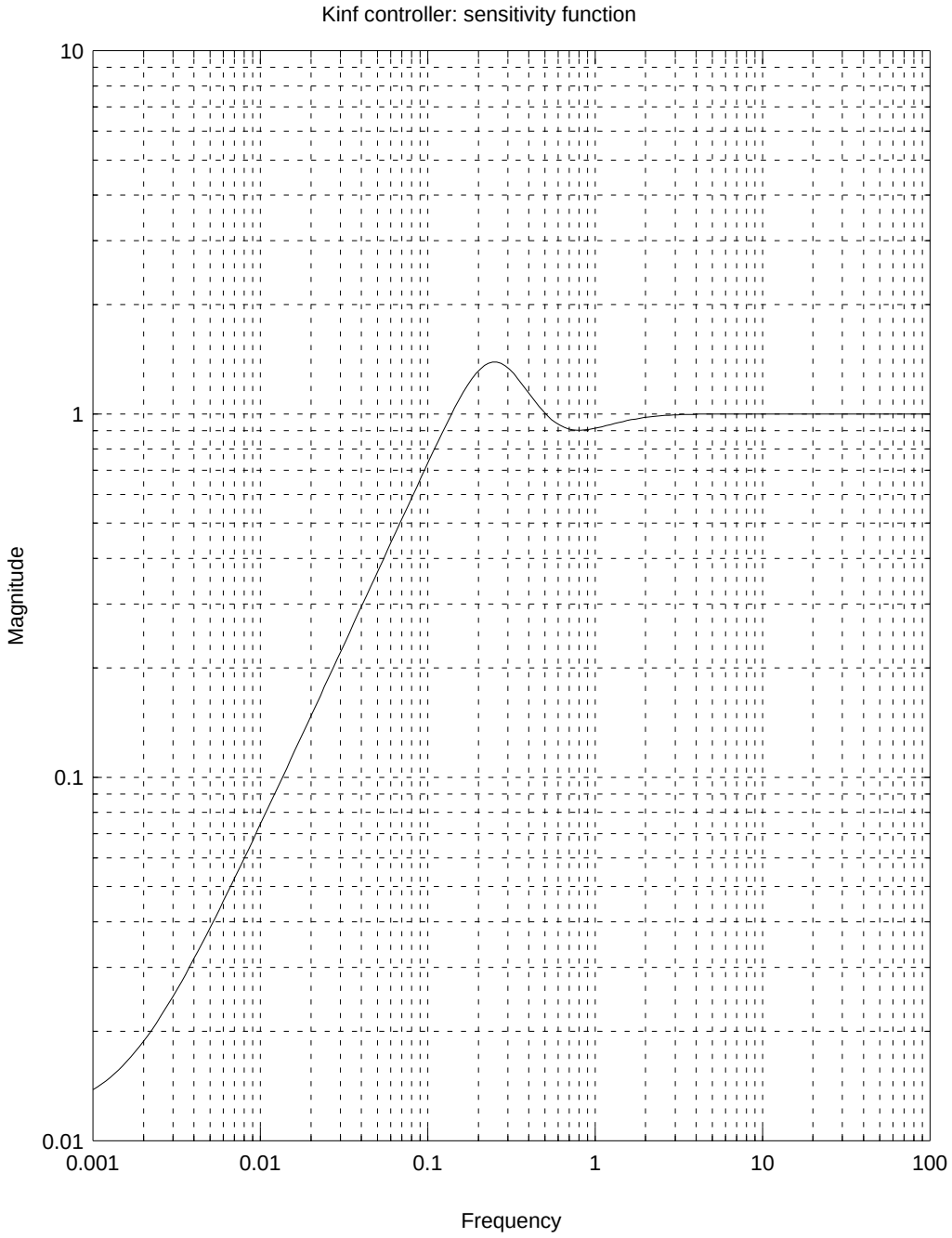
real	imaginary	frequency (rad/sec)	damping ratio
-5.0000e-02	3.1225e-01	3.1623e-01	0.1581
-5.0000e-02	-3.1225e-01	3.1623e-01	0.1581
-9.5105e-01	0.0000e+00	9.5105e-01	1.0000
-9.7350e-01	-1.2285e+00	1.5675e+00	0.6211
-9.7350e-01	1.2285e+00	1.5675e+00	0.6211
-5.0000e+00	0.0000e+00	5.0000e+00	1.0000
-5.0901e+00	0.0000e+00	5.0901e+00	1.0000
-1.1812e+01	0.0000e+00	1.1812e+01	1.0000

Zeros:

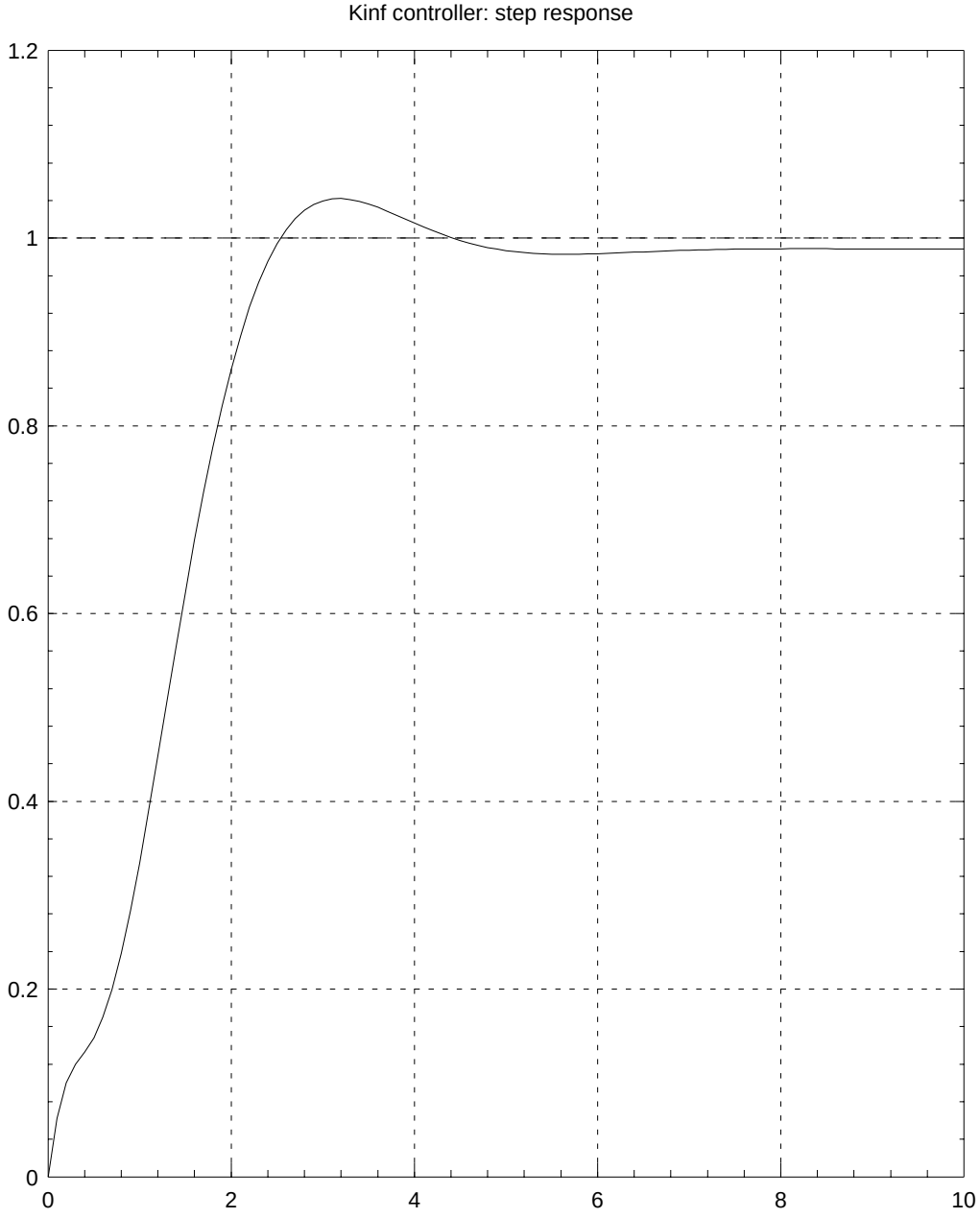
real	imaginary	frequency (rad/sec)	damping ratio
-5.0000e-02	3.1225e-01	3.1623e-01	0.1581
-5.0000e-02	-3.1225e-01	3.1623e-01	0.1581
-1.0000e+00	0.0000e+00	1.0000e+00	1.0000
5.0000e-01	3.1225e+00	3.1623e+00	-0.1581
5.0000e-01	-3.1225e+00	3.1623e+00	-0.1581
-5.0000e+00	0.0000e+00	5.0000e+00	1.0000
-2.0000e+01	0.0000e+00	2.0000e+01	1.0000

```
# Examine sensitivity function

sens = inv(1 + plant*Kinf)
sensg = freq(sens,omega)
```

```
gph2 = ctrlplot(sensg,{logmagplot});
gph2 = plot(gph2,{title="Kinf controller:  sensitivity function"})?
```


Kinf controller: sensitivity function
10
1
0.1
0.01
Magnitude
0.001
0.01
0.1
1
10
100
Frequency

```
# Examine step response

step = gstep([0:0.1:10],0,1)
y = clpinf*step
gph3 = ctrlplot([y,step]);
gph3 = plot(gph3,{title="Kinf controller:  step response"})?
```


Kinf controller: step response
1.2
1
0.8
0.6
0.4
0.2
0
0
2
4
6
8
10

See also

`hinfsyn`, `hinfnorm`, `h2norm`

h2syn

Syntax

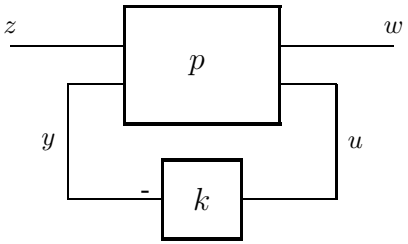
```
k = h2syn(p,nmeas,ncon,{keywords})
```

Parameter List

Inputs:	p	Generalized interconnection structure (DYNAMIC SYSTEM)
	nmeas	measurement vector dimension.
	ncon	control vector dimension.
Keywords:	schur_solution	real Schur decomposition for Riccati solution (default)
	eig_solution	eigendecomposition for Riccati solution.
	epr	epr = value. Tolerance for determining when the Hamiltonian eigenvalues lie on the $j\omega$ axis. Default = 0.5*sqrt(eps)
Outputs:	k	H_2 optimal controller.

Description

The H_2 optimal controller for the interconnection, p, is calculated. The resulting closed loop system is illustrated below.



The variables `ncon` and `nmeas` are used to specify the dimensions of u and y in the above diagram ($ncon = \dim(u)$ and $nmeas = \dim(y)$). The objective is to design a stabilizing controller, k, which minimizes the H_2 norm of the closed loop system between w and z.

p is a state-space system, which can be partitioned with respect to $[z;\ y]$ and $[w;\ u]$ in the following way.

$$p = \left[\begin{array}{c|cc} a & b_1 & b_2 \\ \hline c_1 & d_{11} & d_{12} \\ c_2 & d_{21} & d_{22} \end{array}\right].$$

The following assumptions must hold:

1. (a, b_2, c_2) is stabilizable and detectable
2. d_{12} and d_{21} have full rank
3. $d_{11} = 0$
4. The matrix $[a - j\omega I, b_2; c_1, d_{12}]$ has full column rank for all ω
5. The matrix $[a - j\omega I, b_1; c_2, d_{21}]$ has full row rank for all ω

In theory the Riccati equations should always have a solution. However, Hamiltonian eigenvalues which are close to the imaginary axis will give problems. `epr` specifies a tolerance for how close the calculated eigenvalues can be to the $j\omega$ axis.

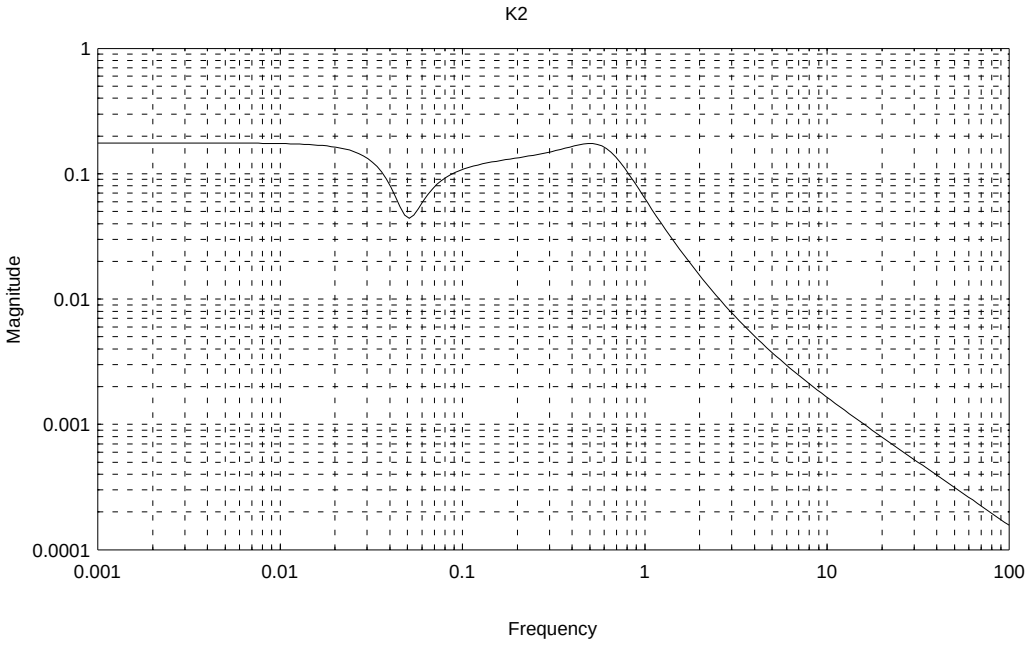
Reference

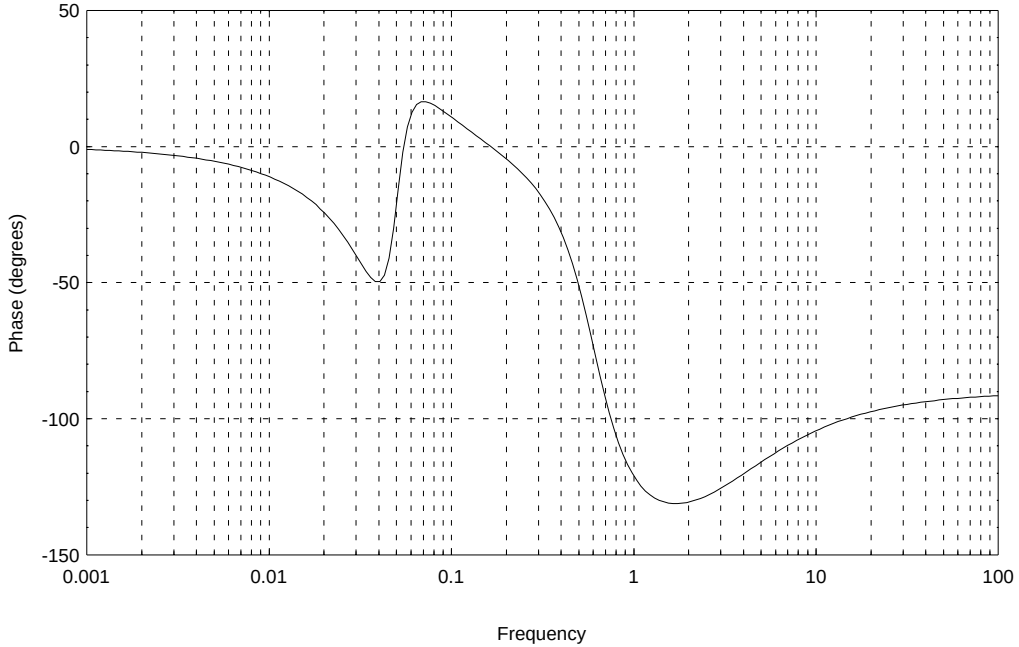
This function uses the state-space formulae given in:
"State-space formulae for all stabilizing controllers that satisfy an H_∞ norm bound and relations to risk sensitivity," Keith Glover and John Doyle, Systems & Control Letters **11**, pp. 167–172., Oct, 1988.

Example

```
# Set up a simple closed loop problem.
# This example is also studied in the
# hinfsyn on-line help.

plant = makepoly([0.1,-0.1,1],"s")*makepoly([1,1],"s")...
        /(makepoly([1,0.1,.1],"s")*(makepoly([0.2,1],"s")))

# Create weights.  These are definitely not the best
# for an H2 design - they are chosen to match with
# the Hinf design example for comparison purposes.

Wperf = 100/makepoly([100,1],"s")
Wact = makepoly([0.5,0.05],"s")/makepoly([0.05,1],"s")

# Form the weighted interconnection structure

sysnames = ["plant";"Wperf";"Wact"]
sysinp = ["ref";"control"]
sysout = ["Wperf"; "Wact"; "ref-plant"]
syscnx = ["control"; ...        # input to plant
          "ref-plant"; ...      # input to Wperf
          "control"]            # input to Wact

wghtic = sysic(sysnames,sysinp,sysout,syscnx,plant,...
                Wperf,Wact)

# Design H2 controller

nctrls = 1
nmeas = 1
```

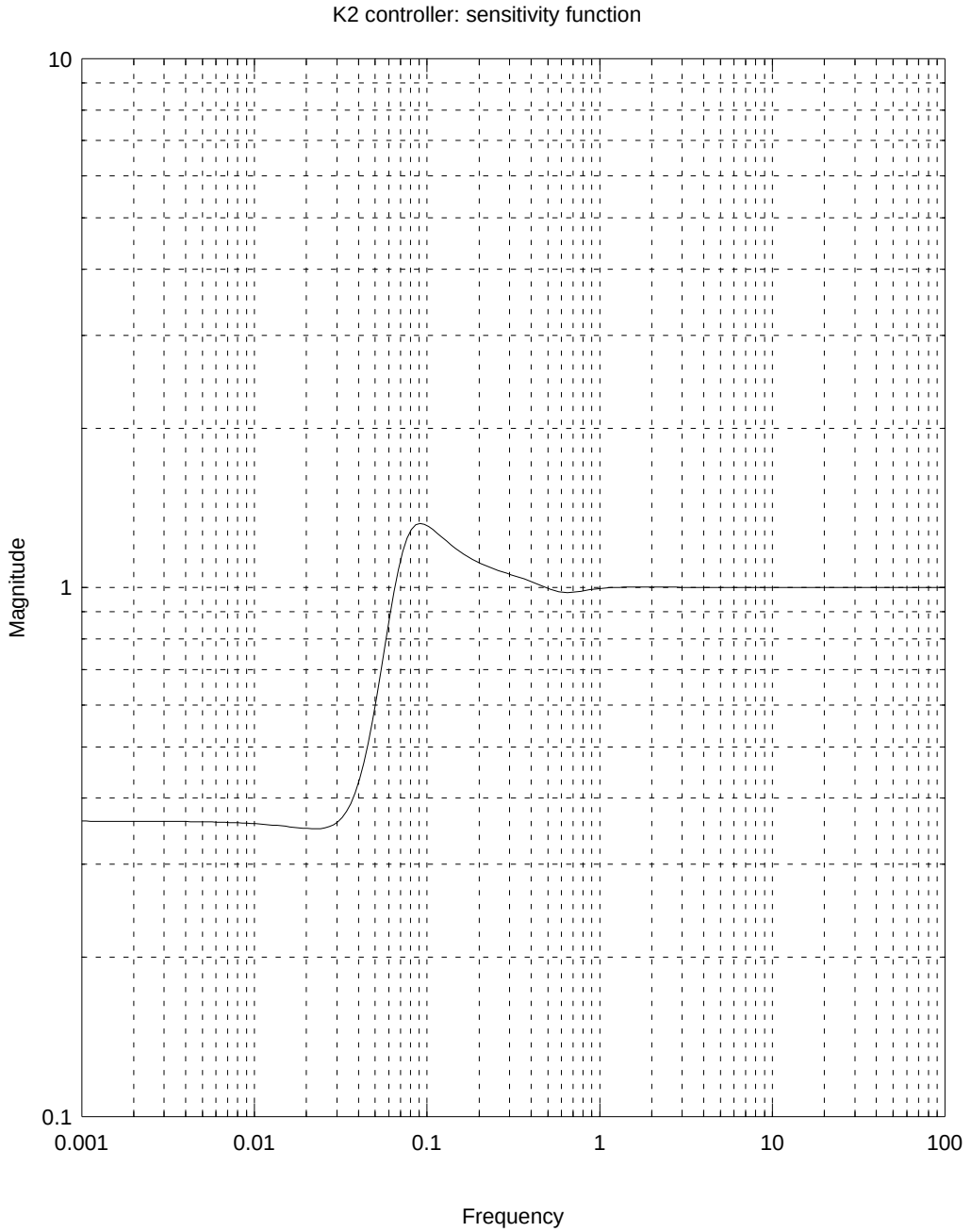
```
K2 = h2syn(wghtic,nmeas,nctrls)
rifd(K2)


Poles:

      real         imaginary      frequency      damping
                                  (rad/sec)       ratio

  -1.4046e-01    -2.3161e-01     2.7087e-01     0.5186
  -1.4046e-01     2.3161e-01     2.7087e-01     0.5186
  -1.5863e+00     3.4754e+00     3.8203e+00     0.4152
  -1.5863e+00    -3.4754e+00     3.8203e+00     0.4152
  -5.2060e+00     0.0000e+00     5.2060e+00     1.0000


Zeros:

      real         imaginary      frequency      damping
                                  (rad/sec)       ratio

  -5.0000e-02    -3.1225e-01     3.1623e-01     0.1581
  -5.0000e-02     3.1225e-01     3.1623e-01     0.1581
  -5.0000e+00     0.0000e+00     5.0000e+00     1.0000
  -2.0000e+01     0.0000e+00     2.0000e+01     1.0000


omega = logspace(0.001,100,200)
K2g = freq(K2,omega)
gph1 = ctrlplot(K2g,{bode});
gph1 = plot(gph1,{title="K2"})?
```


K2
Magnitude
1
0.1
0.01
0.001
0.0001
0.001
0.01
0.1
1
10
100
Frequency

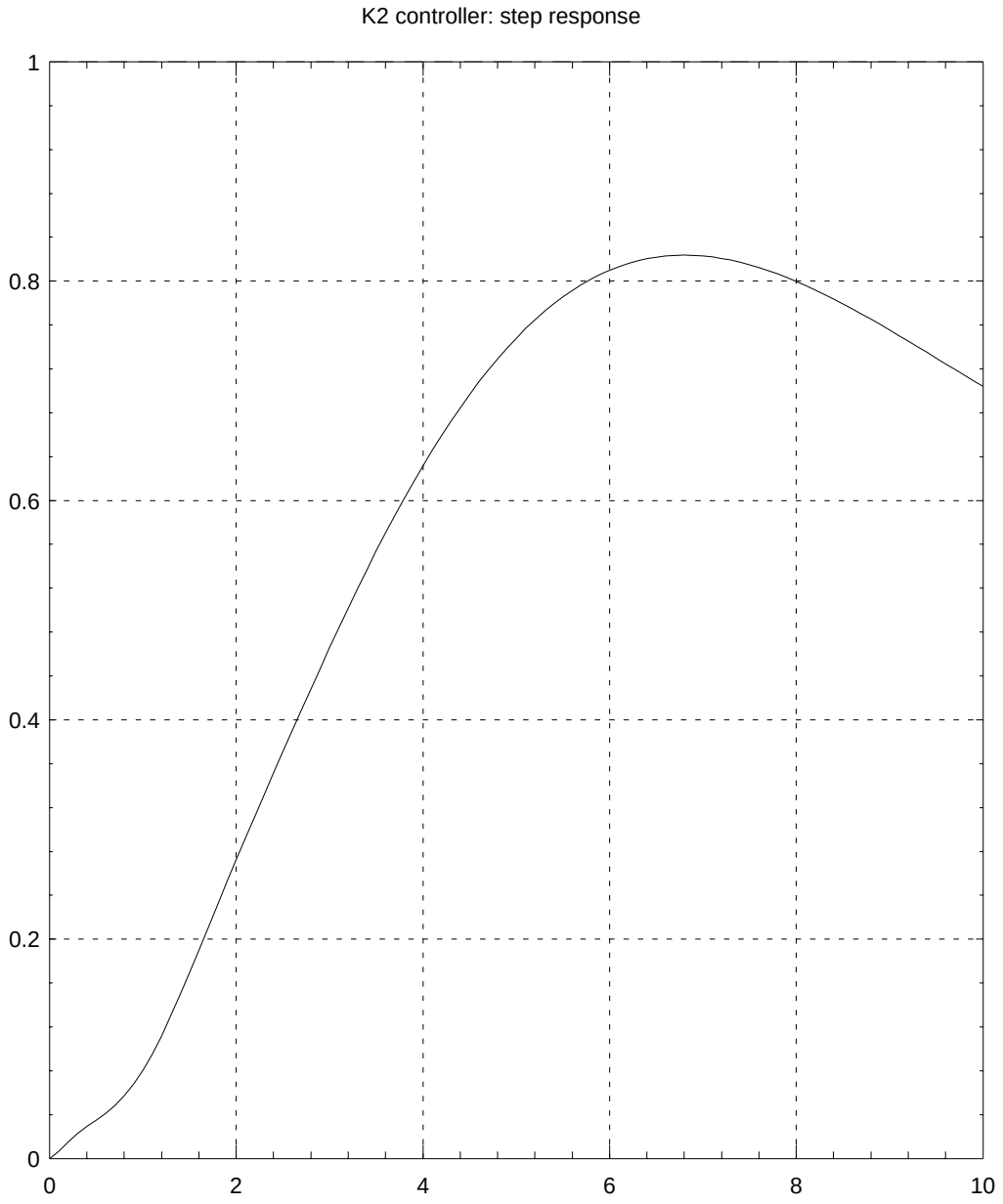

Phase (degrees)
50
0
-50
-100
-150
0.001
0.01
0.1
1
10
100
Frequency

```
# Use sysic to create unweighted interconnection

ic = sysic("plant",["ref";"ctrl"],["plant";"ref-plant"],...
               "ctrl",plant)
clp2 = starp(ic,K2)
rifd(clp2)


Poles:

      real        imaginary      frequency      damping
                                 (rad/sec)       ratio

  -5.0000e-02     3.1225e-01     3.1623e-01     0.1581
  -5.0000e-02    -3.1225e-01     3.1623e-01     0.1581
  -1.8120e-01    -4.2333e-01     4.6048e-01     0.3935
  -1.8120e-01     4.2333e-01     4.6048e-01     0.3935
  -1.6753e+00     3.4263e+00     3.8139e+00     0.4393
  -1.6753e+00    -3.4263e+00     3.8139e+00     0.4393
  -4.9957e+00     0.0000e+00     4.9957e+00     1.0000
  -5.0000e+00     0.0000e+00     5.0000e+00     1.0000


Zeros:

      real        imaginary      frequency      damping
                                 (rad/sec)       ratio

  -5.0000e-02     3.1225e-01     3.1623e-01     0.1581
  -5.0000e-02    -3.1225e-01     3.1623e-01     0.1581
  -1.0000e+00     0.0000e+00     1.0000e+00     1.0000
   5.0000e-01     3.1225e+00     3.1623e+00    -0.1581
   5.0000e-01    -3.1225e+00     3.1623e+00    -0.1581
  -5.0000e+00     0.0000e+00     5.0000e+00     1.0000
  -2.0000e+01     0.0000e+00     2.0000e+01     1.0000


# Examine sensitivity function

sens = inv(1 + plant*K2)
```

```
sensg = freq(sens,omega)

gph2 = ctrlplot(sensg,{logmagplot});
gph2 = plot(gph2,{title="K2 controller:  sensitivity function"})?
```


K2 controller: sensitivity function
10
1
0.1
Magnitude
0.001
0.01
0.1
1
10
100
Frequency

```
# Examine step response

step = gstep([0:0.1:10],0,1)
y = clp2*step
gph3 = ctrlplot([y,step]);
gph3 = plot(gph3,{title="K2 controller:  step response"})?
```


K2 controller: step response
1
0.8
0.6
0.4
0.2
0
0
2
4
6
8
10

See also

`hinfsyn`, `h2norm`, `hinfnorm`

interp

Syntax

```
outpdm = interp(inpdm,stepsize,final {keywords})

outpdm = interp(inpdm,domspec, {keywords})
```

Parameter List

Inputs:	inpdm	Input pdm.
	stepsize	Increment in outpdm domain.
	final	Last value in outpdm domain. Optional: default = max(domain(inpdm)).
	domspec	Regular vector or PDM used to specify the domain of outpdm.
Keywords:	order	Interpolation order. Values are: 0 zero order hold (default) 1 linear interpolation
Outputs:	outpdm	interpolated pdm

Description

m is interpolated to give outpdm. Two syntaxes are available for specifying the domain of outpdm.

In the first the domain is,

[min(domain(inpdm)):final:stepsize].

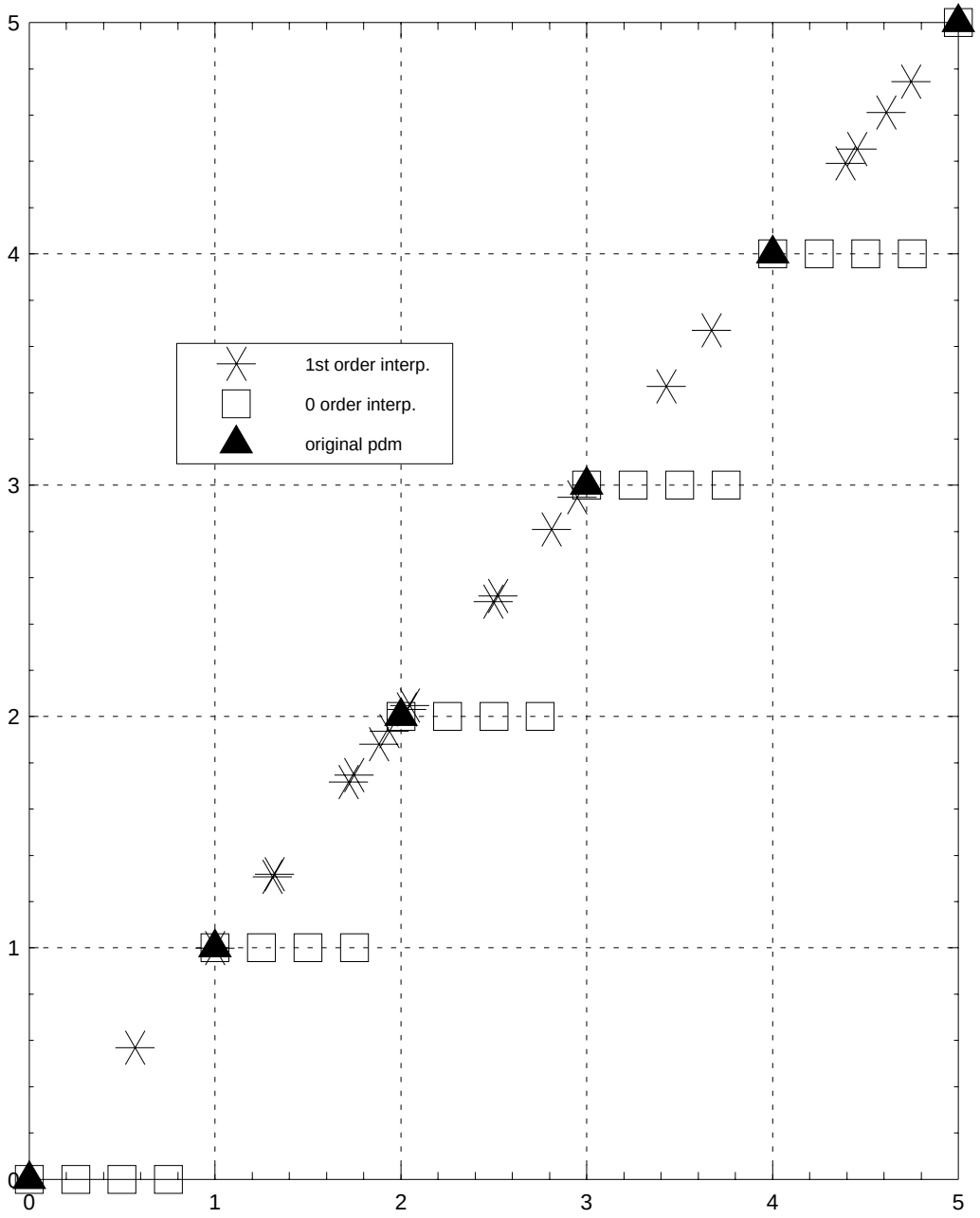
If final is not specified max(domain(inpdm)) is used.

The second syntax is, domain(domspec) if domspec is a pdm or domspec if domspec is a vector.

This function differs from the `interpolate` function in that it can handle zero order hold type interpolation and deal with irregularly spaced input pdms. Irregularly spaced output pdms can be generated with the domspec syntax. These features are often useful when dealing with data generated from experiments.

Example

```
time = [0:1:5]
u = gstep(time,time,time)

u1 = interp(u,0.25,{order=0})

time2 = sort(5*random(20,1))
u2 = interp(u,time2,{order=1})

gph1 = ctrlplot(u2,{marker=1,marker_style=1,...
            marker_size=1,line=0}); gph1 =
plot(u1,gph1,{marker=1,marker_style=6,...
            marker_size=1,line=0}); gph1 =
plot(u,gph1,{marker=1,marker_style=9,...
            marker_size=1,line=0}); gph1 = plot(gph1,{legend=["1st
order interp.";...
            "0 order interp.";"original pdm"]})?
```


1st order interp.
0 order interp.
original pdm
0
1
2
3
4
5

See Also

`interpolate`

mergeseg

Syntax

```
outpdm = mergeseg(pdm1,pdm2, {keywords})
```

Parameter List

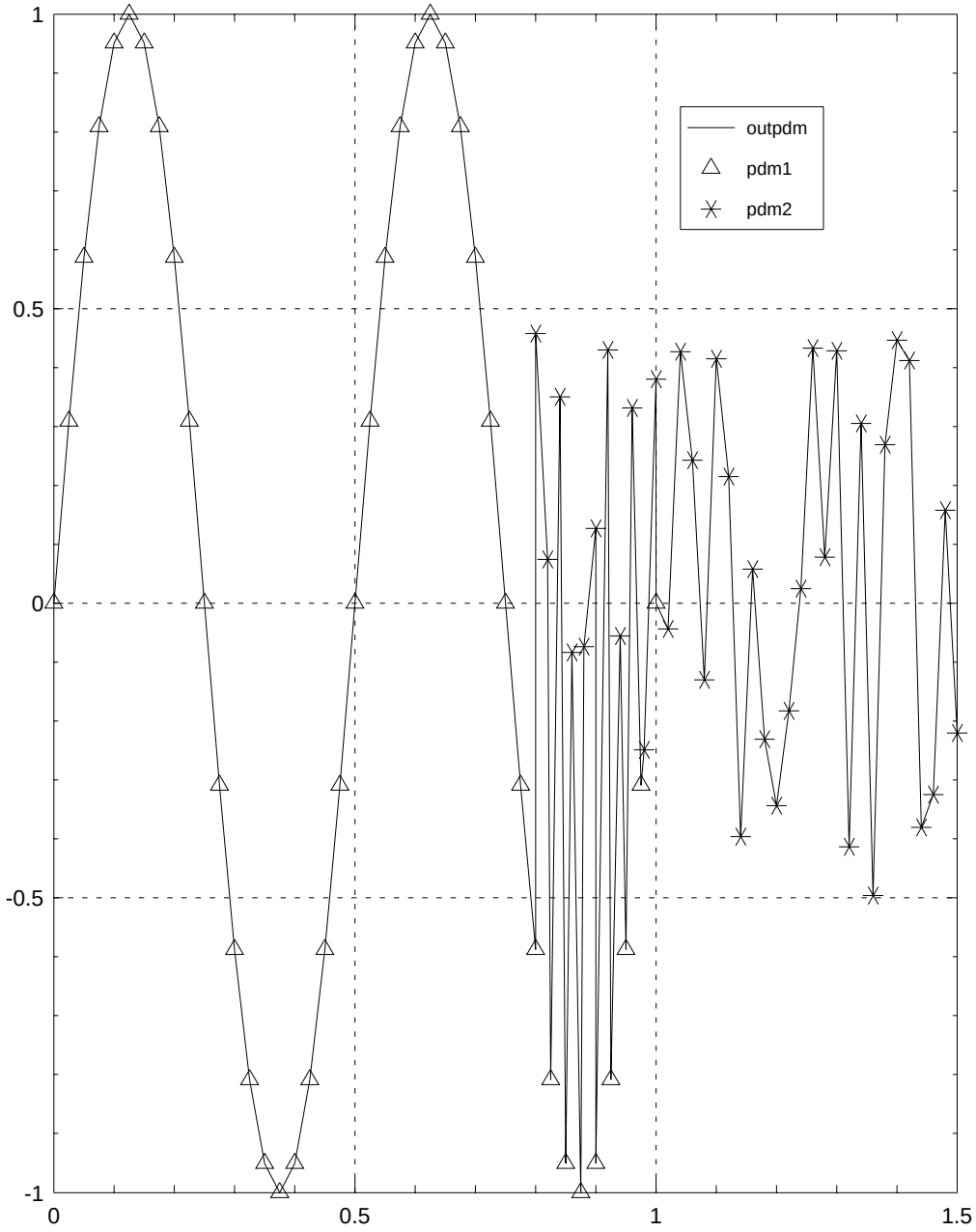
Inputs:	pdm1	input PDM
	pdm2	input pdm
Keywords:	domsort	Sort the result of merging the PDMs. If !domsort then pdm2 is simply concatenated onto pdm1. Boolean. Default = 1.
	increasing	Sort in increasing order. Boolean. Default = 1.
	decreasing	Sort in decreasing order. Boolean. Default = 0.
	duplicates	Leave duplicated domain values in outpdm. If !duplicates is specified then only the matrices from pdm1 associated with the duplicate domain values are included in outpdm. The value of tol determines what constitutes equality in the domains. Boolean. Default = 1.
	tol	Tolerance in determining duplicates in the domains. Default = eps*max([domain(pdm1);domain(pdm2)])
Outputs:	outpdm	output pdm

Description

This function concatenates two pdms and then sorts them according to their domain. This is useful for merging experimental data taken over different frequency or time ranges. A regular domain is not required. The pdms must have equal row and column dimensions.

Example

```
time1 = [0:0.025:1]
pdm1 = gsin(time1,{frequency=2})
time2 = [0.8:0.02:1.5]
pdm2 = randpdm([],1,1,{dom=time2,zeromean})

outpdm = mergeseg(pdm1,pdm2)

gph1 = ctrlplot(outpdm);
gph1 = plot(pdm1,gph1,{marker=1,marker_style=8,...
                       line=0});
gph1 = plot(pdm2,gph1,{marker=1,marker_style=1,...
           line=0,legend=["outpdm";"pdm1";"pdm2"]})?
```


outpdm
pdm1
pdm2
1
0.5
0
-0.5
-1
0
0.5
1
1.5

mkpert

Syntax

```
[pertsys] = mkpert(Delta,blk,mubnds,{fselect,pnorm,Hertz})
```

Parameter List

Inputs:	Delta	Lower bound perturbation data from mu calculation. (PDM)
	blk	block structure (refer to mu function documentation). (matrix)
	mubnds	Calculated mu bounds (PDM).
Keywords:	fselect	Scalar valued. Specifies the frequency for interpolation, overriding that obtained from mubnds.
	pnorm	Scalar valued. Specifies the norm of the resulting system, overriding that calculated from mubnds.
	Hertz	Boolean. Domain of PDMs are in units of Hertz. This keyword is mandatory. To specify radians/sec use !Hertz.
Outputs:	pertsys	Constructed perturbation. (DYNAMIC SYSTEM)

Description

Creates the rational, stable, system which interpolates the perturbation, Delta, at a specified frequency. The frequency chosen is the one where the lower bound to mu is maximum. This corresponds to the smallest destabilizing perturbation. The system has the assumed structure given by blk.

This function may also be called with a single pdm argument. The result will interpolate the argument at its first domain value.

This function is used to construct a bad perturbation, the effects of which can be studied by simulation.

The bounds, mubnds, are used to determine the "worst-case" frequency for the interpolation. The norm of pertsys is 1/(norm(Delta($j\omega$))) where ω is the chosen frequency. This is the smallest destabilizing perturbation at that frequency.

Both the interpolation frequency and the norm of pertsys can be specified via keywords. The user may be interested only in certain frequency ranges and may have an assumed norm bound on the perturbation (typically unity).

Components of pertsys (or all of it if appropriate) may be real valued gains rather than dynamic systems.

Example

The use of `mkpert` is studied in context in the on-line help for `musynfit` and the manual documentation for `musynfit` (page 299).

mkphase

Syntax

```
[cdata] = mkphase(magdata, {skipchks,Hertz})
```

Parameter List

Inputs:	magdata	Magnitude data (PDM)
Keywords:	skipchks	Boolean. Skip the error checking. (Default = 0)
	Hertz	The domain of the PDM is in Hertz. This is the default. !Hertz specifies a domain in rad/sec.
Outputs:	cdata	Complex valued data corresponding to a minimum phase transfer function.

Description

Fits phase data to magnitude data. The phase is equivalent to that produced by minimum phase system. A complex cepstrum method is used.

Reference

For further details see: *"Digital Signal Processing,"* A.V. Oppenheim & R.W. Schafer, p.501, Prentice-Hall, 1975.

Example

The on-line help example is the same as that for `fitsys`. Refer to page 239.

Limitations

Limited to SISO systems.

See Also

`fitsys`, `ccepstrum`

modalstate

Syntax

```
outsys = modalstate(sys, {keywords})
```

Parameter List

Inputs:	sys	Input Dynamic System
Keywords:	increasing	Boolean. Order in terms of increasing magnitude (continuous) or angle (discrete). Default = 1
	decreasing	Boolean. Order in terms of decreasing magnitude (continuous) or angle (discrete). Default = 0
	splitstable	Boolean. Separate the stable from the unstable modes and order separately. Default = 0.
Outputs:	outsys	output Dynamic System

Description

Transform the system to give a block diagonal A matrix, with complex eigenvalues in 2×2 blocks and real eigenvalues in 1×1 blocks. The system must be diagonalizable to avoid introducing errors. The initial condition is also transformed. The state names are deleted, however the input and output names, and period (if discrete) are preserved.

Example

```
sys = randsys(6,1,1,{stable})
rifd(sys)
Poles:

      real        imaginary      frequency      damping
                                 (rad/sec)       ratio
```

```
  -8.1602e-01     1.3353e+00     1.5649e+00     0.5215
  -8.1602e-01    -1.3353e+00     1.5649e+00     0.5215
  -4.6142e+00    -1.6648e+01     1.7275e+01     0.2671
  -4.6142e+00     1.6648e+01     1.7275e+01     0.2671
  -2.2478e+01     0.0000e+00     2.2478e+01     1.0000
  -3.9525e+01     0.0000e+00     3.9525e+01     1.0000


Zeros:

      real        imaginary      frequency      damping
                                 (rad/sec)       ratio

  -1.0270e+00    -1.5086e+00     1.8250e+00     0.5628
  -1.0270e+00     1.5086e+00     1.8250e+00     0.5628
  -4.5529e+00    -1.6621e+01     1.7233e+01     0.2642
  -4.5529e+00     1.6621e+01     1.7233e+01     0.2642
  -2.4416e+01     0.0000e+00     2.4416e+01     1.0000
  -3.9534e+01     0.0000e+00     3.9534e+01     1.0000

sys1 = modalstate(sys)
[a1,,,] = abcd(sys1)
# compare a1 to the poles of sys
a1?

a1 (a square matrix) =

 -0.816024    -1.33529      0          0          0          0
  1.33529     -0.816024     0          0          0          0
  0            0           -4.61417   16.6478    0          0
  0            0          -16.6478    -4.61417   0          0
  0            0            0          0        -22.4779     0
  0            0            0          0          0        -39.5252
```

mu

Syntax

```
[mubnds,D,Dinv,Delta,sens] = mu(M,blk)
```

Parameter List

Inputs:	M	Matrix or pdm.
	blk	Block structure defined by a matrix of dimension: number of blocks $\times$ 2. If the ith block has c outputs and r inputs, then blk(i,:) = [r,c]. The default is equivalent to 1x1 blocks (M must be square).
Outputs:	mubnds	Upper and lower bounds (in vector form) for mu(M).
	D,Dinv	D-scale matrices giving the calculated upper-bound. mu(M) $\leq$ msv(D*M*Dinv)
	Delta	Perturbation achieving the lower bound.
	sens	Sensitivity of the upper bound with respect to the values in D & Dinv

Description

Calculates the upper and lower bounds of the structured singular value of M, with block structure: blk. The upper bound scaling matrices and the lower bound destabilizing perturbation also returned.

The Osborne method is used to calculate the upper bound (for small matrices this is enhanced by a Perron Frobenius method) and a power iteration is used for the lower bound.

Example

```
# The following is the classic example showing
```

```
# that mu is not equal to its upper bound for
# more than three full blocks.

gamma = 3 + sqrt(3); beta = sqrt(3) -1
a = sqrt(2/gamma); b = 1/sqrt(gamma)
c = 1/sqrt(gamma); d = -sqrt(beta/gamma)
f = (1+jay)*sqrt(1/(gamma*beta))
psi1 = -pi/2; psi2 = pi

U = [a,0; b,b; c,jay*c; d,f]
V = [0,a; b,-b; c,-jay*c; f*exp(jay*psi1), d*exp(jay*psi2)]
scl = diagonal(random(4,1)+0.1*ones(4,1))
M = scl*U*V*'*inv(scl)

# Consider 4 1x1 blocks

blk1 = [1,1; 1,1; 1,1; 1,1]
[mubnds1,D1,Dinv1,Delta1] = mu(M,blk1)

max(svd(M))?
ans (a scalar) =   2.81264

max(svd(D1*M*Dinv1))?
ans (a scalar) =   1

mubnds1?
mubnds1 (a column vector) =

  1
  0.860682

# Consider 1 4x4 block (equivalent to max sing.  val.)

blk2 = [4,4]
[mubnds2,D2,Dinv2,Delta2] = mu(M,blk2)

# Note that the perturbation is such that
# det(I-M Delta) = 0.

max(svd(D2*M*Dinv2))?
```

```
ans (a scalar) =   2.81264

mubnds2?
mubnds2 (a column vector) =

  2.81264
  2.81264

det(eye(4,4) - M*Delta2)?
ans (a scalar) =   -2.53156e-16 + 4.3828e-17 j
```

For an example of how mu is used for system robustness analysis, refer to the on-line help for musynfit (page 299).

Limitations

This version of the software cannot handle repeated blocks or real valued blocks.

musynfit

Syntax

```
[Dsys,Dinvsys] = musynfit(Dmag,blk,nmeas,nctrls,..
                          weight,M,order,{keywords})
```

Parameter List

Inputs:	Dmag	New D matrix from mu calculation (magnitude data only). The domain is assumed to be in Hertz.
	blk	block structure (refer to `mu` function documentation).
	nmeas	scalar: number of measurements
	nctrls	scalar: number of controls
	weight	(optional) weighting function for the fit. Typically the sensitivity output of mu is used. Must be a pdm of dimension: number of blocks $\times$ 1.
	M	(optional) Matrix (PDM) of interest in the μ calculation. I.e. $\mu(M) \leq \overline{\sigma}(DMD^{-1})$. If M is provided a second plot compares the $\mu(M)$ upper bound, based on Dmag, to the $mu(M)$ upper bound using a frequency response of the D scale transfer function approximation.
	order	(optional) A scalar (or vector of dimension nblks-1) specifying the order of the fit to be used. If this argument is present, the function runs without requiring user interaction and does not graph the results.

Keywords:	Hertz	Boolean. This keyword is mandatory as the function must know whether the domain is in Hertz or radians/second (specified by !Hertz) to fit correctly. Note that the Xmath function `freq` assumes that the frequency range is specified in Hertz.
	plotweight	Boolean, default = 0. This will generate a second plot showing the weighting function. This can be useful is assessing the quality of a given transfer function fit. If the variable M is also specified, the weighting function plot takes priority over the upper bound comparison plot.
	fit	Integer specifying the routine to be used in fitting the data. This sets the default routine which may be changed interactively by the user. The choices are: fit = 1 Xmath function: tfid fit = 2 Xmu function: fitsys The default is fit = 2.
Outputs:	Dsys	New left D scale system.
	Dinvsys	New D inverse system (to be multiplied to M on the right).

Description

Fits stable, minimum phase, transfer functions to each block of the D-scale matrix. A complex cepstrum technique is used to generate the phase response from the provided magnitude data.

The user may interactively select a fitting order and compare the result with the data. A choice of fitting routines is provided. If the order is specified in the command line the function is not interactive.

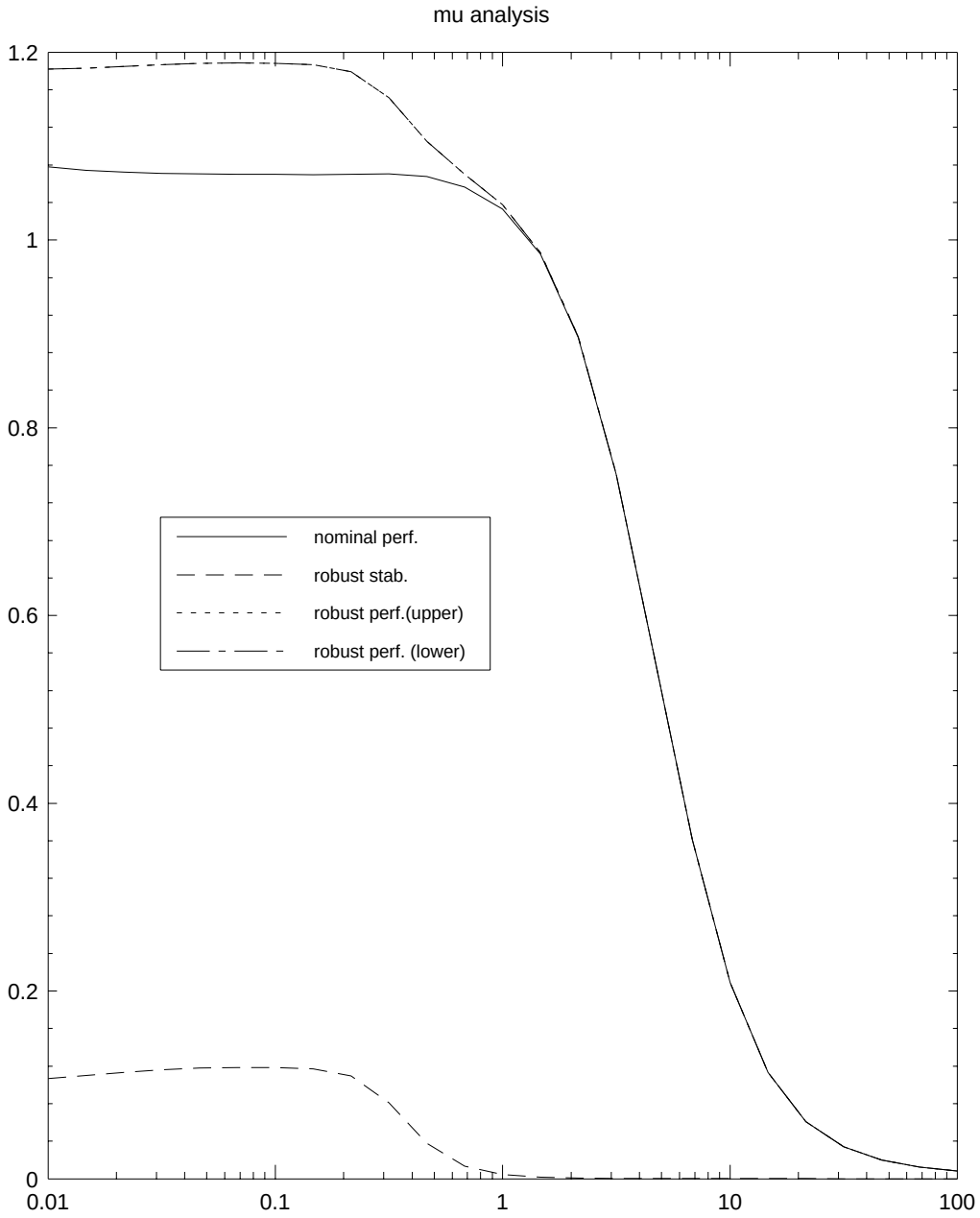
Example

```
# The nominal plant is a double integrator.
# A multiplicative perturbation weight reflects
# increased uncertainty at high frequencies
```

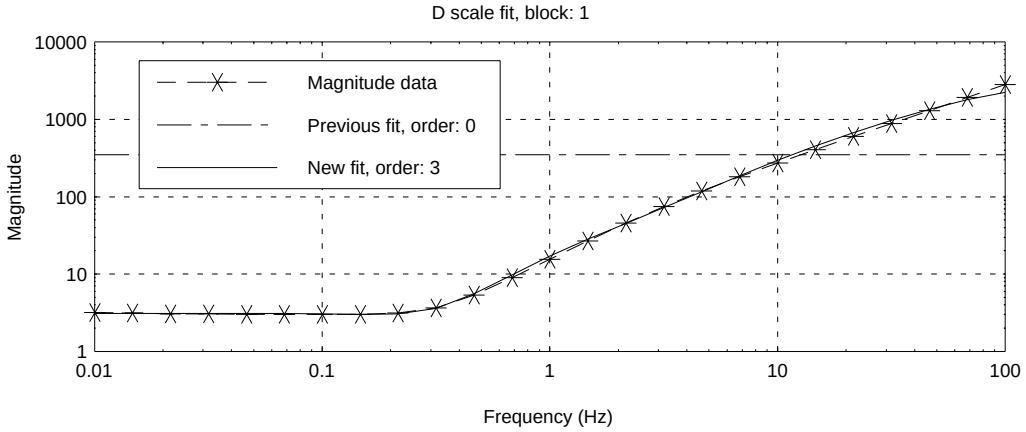
```
P = 1/makepoly([1,0,-0.01],"s")
W = makepoly([1,20],"s")/makepoly([1,200],"s")

# Set up an unweighted interconnection structure
# for unity gain negative feedback.  We include
# the perturbation too.

nms = ["P";"W"]
inp = ["delt";"ref";"noise";"control"]
outp = ["W" ;"ref- delt - P";"control";"ref-delt-P-noise"]
cnx = ["control";"P"]
ic = sysic(nms,inp,outp,cnx,P,W)

# Now include some weights for performance:

Wperf = makepoly([0.01,1],"s")/makepoly([1,0.01],"s")
Wact = 0.1* makepoly([1,1],"s")/makepoly([0.05,1],"s")
Wnoise = 0.01
Wref = makepoly([0.005,1],"s")/makepoly([0.05,1],"s")

# Apply weights to error signals and unknown
# inputs.

wghtic = daug(1,Wperf,Wact,1)*ic*daug(1,Wref,Wnoise,1)

# Perform an H_infinity design.

nmeas = 1            # number of measurements
ncntrls = 1          # number of controls
gmin = 0
gmax = 10
Kinf = hinfsyn(wghtic,nmeas,ncntrls,[gmin;gmax])
Test bounds:      0.0000  <  gamma  <=     10.0000
  gamma      Hx_eig     X_eig    Hy_eig      Y_eig   nrho_xy   p/f
   10.000     7.1e-01   7.7e-04   9.9e-03  -3.8e-19   0.0049    p
    5.000     7.1e-01   7.7e-04   9.8e-03  -1.5e-19   0.0207    p
    2.500     7.0e-01   7.8e-04   9.2e-03   0.0e+00   0.1061    p
    1.250     6.5e-01  -5.6e+02   6.0e-03   0.0e+00   2.3572    f
    1.875     6.9e-01   7.9e-04   8.5e-03  -1.1e-18   0.2681    p
```

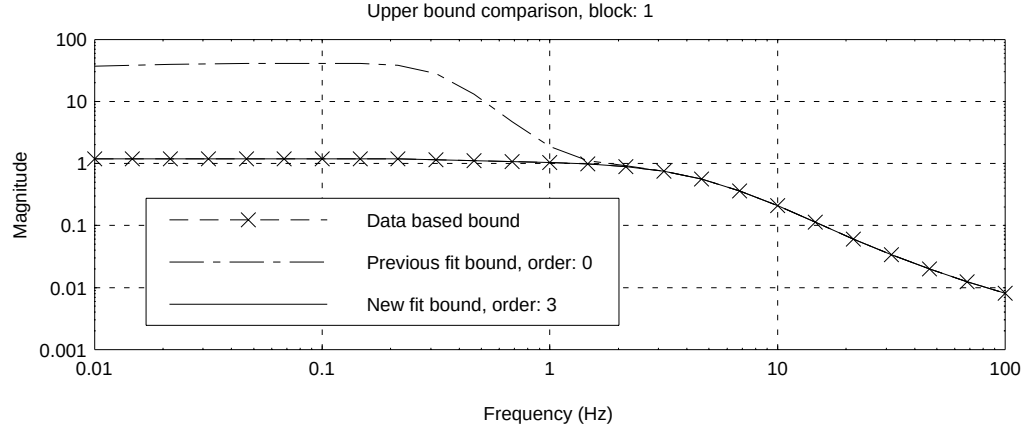
```
    1.562     6.8e-01   8.0e-04   7.7e-03   0.0e+00   0.6811    p
    1.406     6.7e-01   8.1e-04   7.0e-03  -1.1e-18   2.0900    f
    1.484     6.7e-01   8.0e-04   7.4e-03   0.0e+00   1.0394    f
    1.523     6.7e-01   8.0e-04   7.5e-03  -5.0e-19   0.8249    p

 Gamma value achieved:     1.5234
```

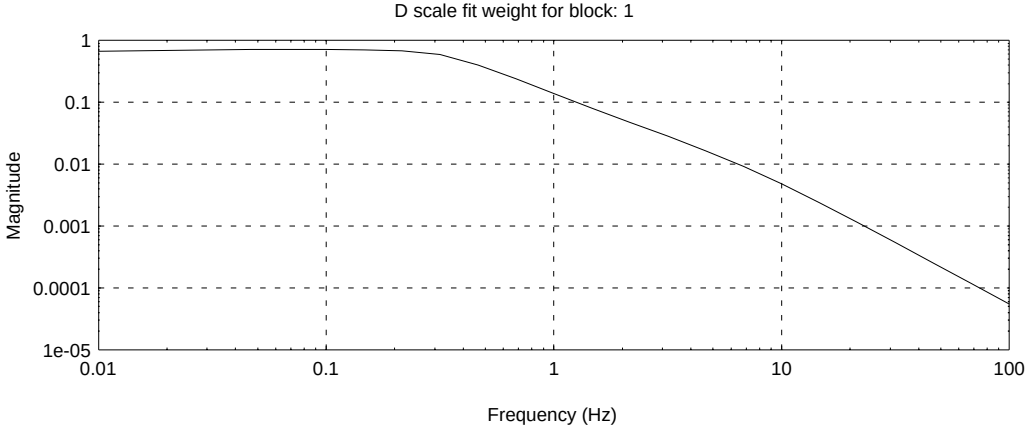
```
G = starp(wghtic,Kinf)
omega = logspace(0.01,100,25)
Gg = freq(G,omega)
G11g = Gg(1,1)
G22g = Gg(2:3,2:3)
rs = max(svd(G11g))
np = max(svd(G22g))

blk = [1,1; 2,2]
[rpbnds1,D1,Dinv1,Delta1,sens1] = mu(Gg,blk)

gph1 = ctrlplot([np;rs;rpbnds1],{log});
gph1 = plot(gph1,{!grid,title="mu analysis",legend=["nominal
perf.";...
       "robust stab.";"robust perf.(upper)";"robust perf.
(lower)"]})?
```


mu analysis
1.2
1
0.8
0.6
0.4
0.2
0
0.01
0.1
1
10
100
nominal perf.
robust stab.
robust perf.(upper)
robust perf. (lower)

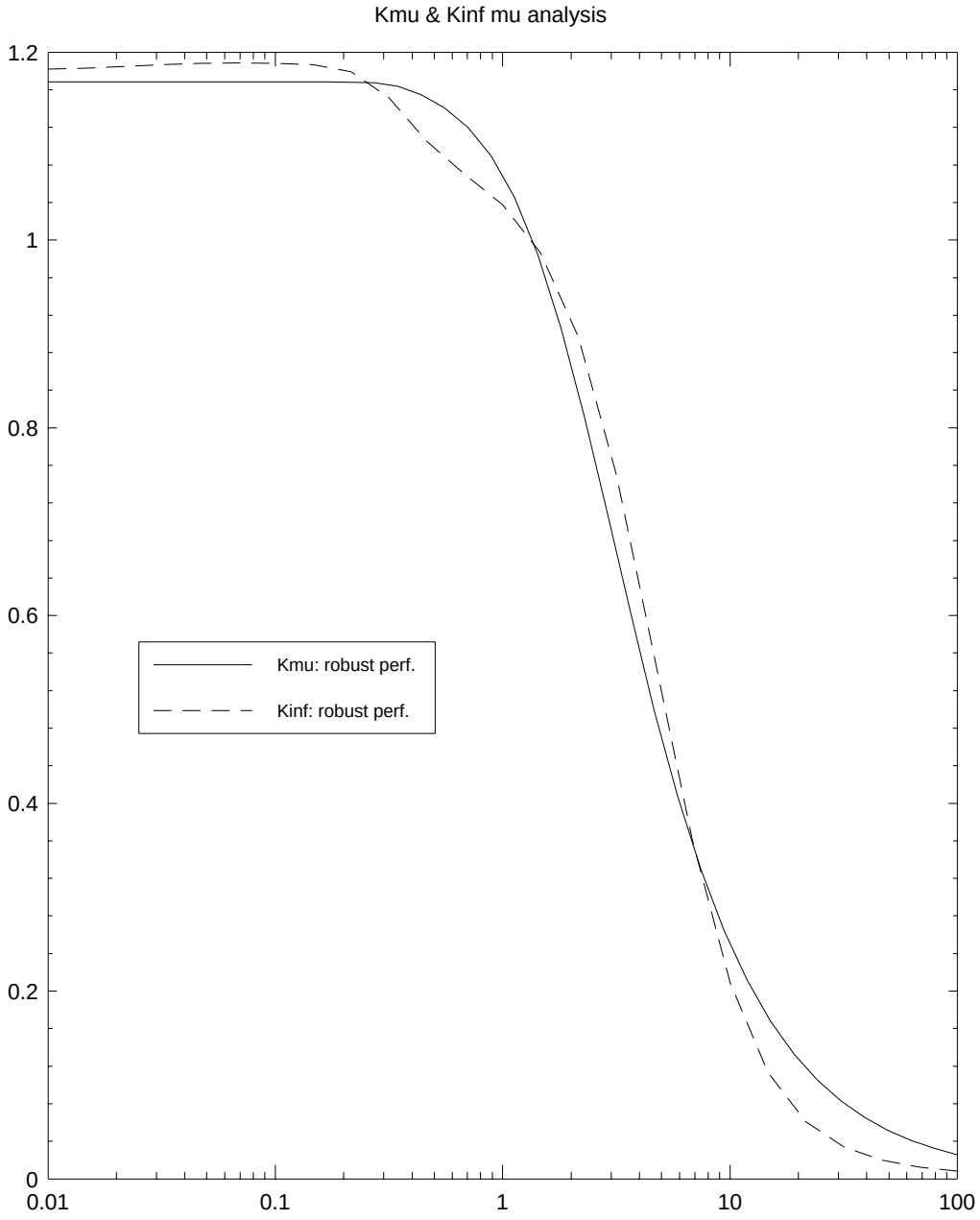
```
# Fit transfer functions to D1 & Dinv1 for a mu
# synthesis iteration

[Ds,Dinvs] = musynfit(D1,blk,nmeas,ncntrls,sens1,Gg,{Hertz})
```


D scale fit, block: 1
Magnitude data
Previous fit, order: 0
New fit, order: 3
Magnitude
Frequency (Hz)
10000
1000
100
10
1
0.01
0.1
1
10
100


Upper bound comparison, block: 1
Data based bound
Previous fit bound, order: 0
New fit bound, order: 3
Magnitude
Frequency (Hz)
100
10
1
0.1
0.01
0.001
0.01
0.1
1
10
100

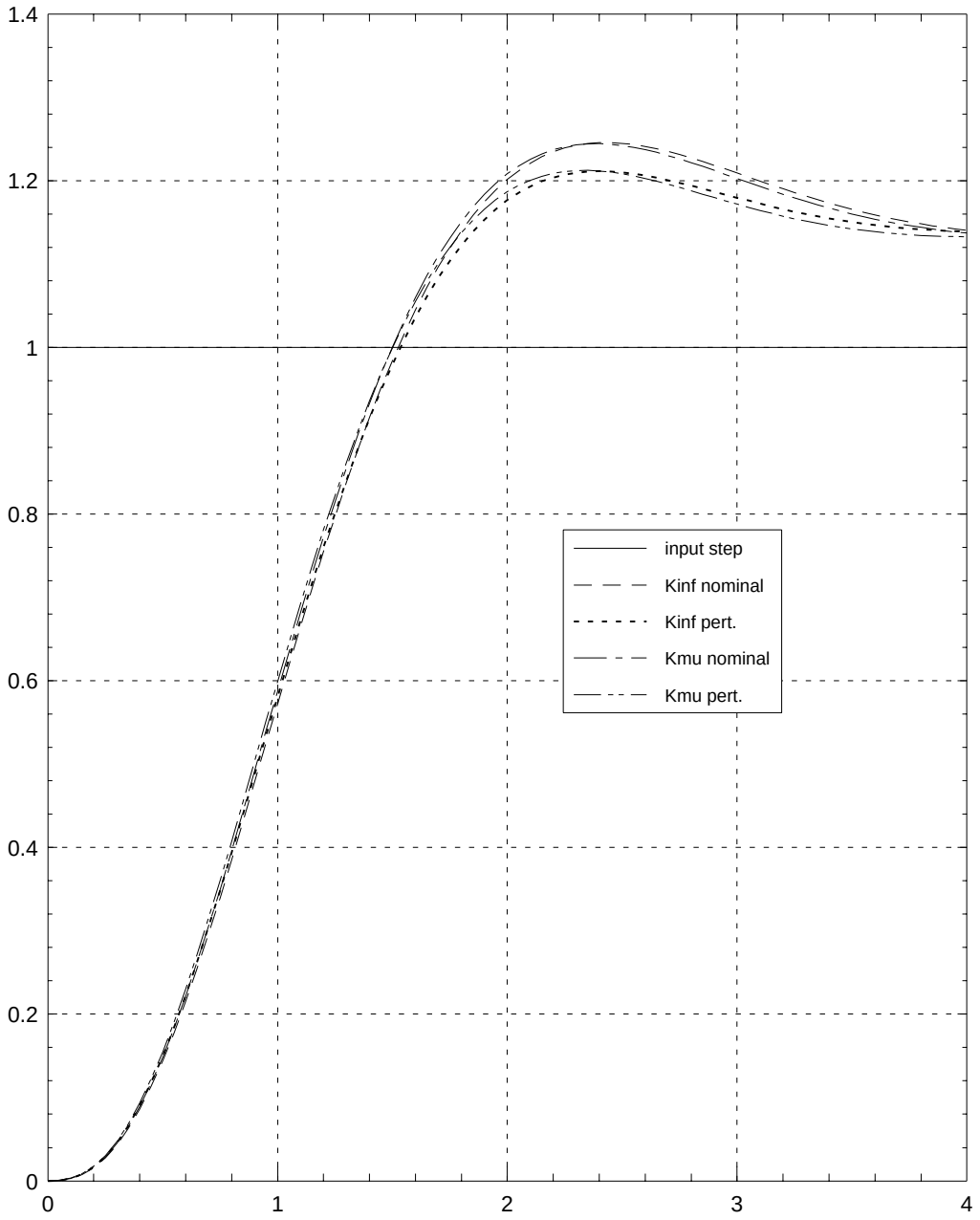

D scale fit weight for block: 1
Magnitude
Frequency (Hz)
1
0.1
0.01
0.001
0.0001
1e-05
0.01
0.1
1
10
100

```
# Apply the D scales to another H_infinity design

Kmu = hinfsyn(Ds*wghtic*Dinvs,nmeas,ncntrls,[gmin;gmax])
Test bounds:      0.0000  <  gamma  <=     10.0000
  gamma      Hx_eig     X_eig    Hy_eig      Y_eig   nrho_xy   p/f
   10.000     6.5e-01   5.6e-07   9.9e-03  -1.0e-15   0.0027    p
    5.000     6.4e-01   5.6e-07   9.8e-03  -6.3e-16   0.0113    p
    2.500     6.4e-01   5.6e-07   9.2e-03   0.0e+00   0.0517    p
    1.250     6.3e-01   5.6e-07   6.0e-03  -7.6e-20   0.4688    p
    0.625     5.6e-01  -1.1e+00   7.1e-14  *******   *******  f
    0.938     6.1e-01  -2.8e+01   7.1e-14  *******   *******  f
    1.094     6.2e-01   5.6e-07   4.1e-03  -3.5e-16   1.2451    f
    1.172     6.2e-01   5.6e-07   5.2e-03  -6.3e-18   0.6922    p

 Gamma value achieved:     1.1719

# Close the loop around the weighted interconnection
# structure.

Gmu = starp(wghtic,Kmu)
omega = logspace(0.01,100,40)
Gmug = freq(Gmu,omega)

blk = [1,1; 2,2]
[rpbnds2,D2,Dinv2,Delta2,sens2] = mu(Gmug,blk)
gph3 = ctrlplot(rpbnds2(1,1),{log});
gph3 = ctrlplot(rpbnds1(1,1),gph3,{log});
gph3 = plot(gph3,{!grid,title="Kmu & Kinf mu analysis",...
        legend=["Kmu:  robust perf.";"Kinf:  robust perf."]})?
```

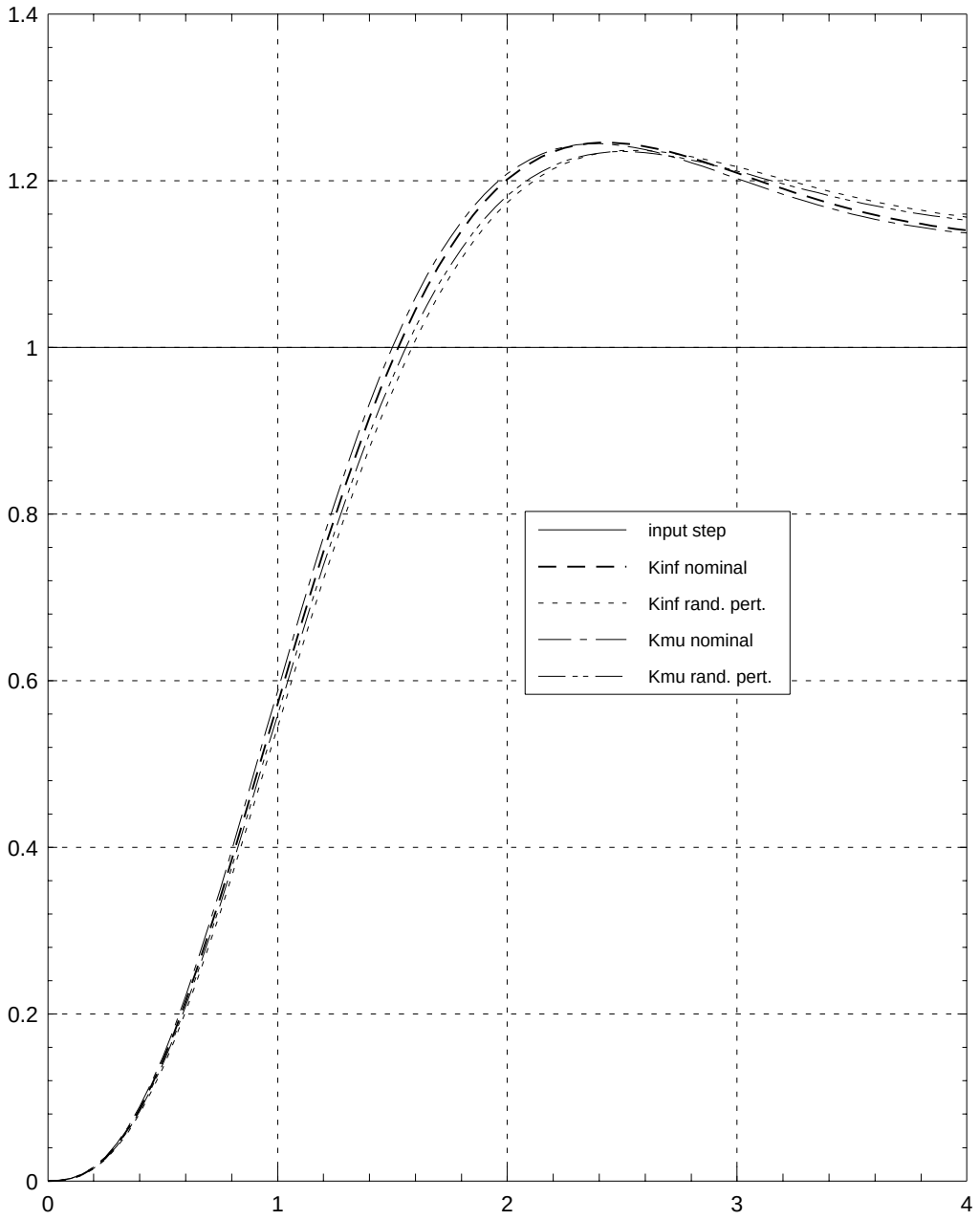

Kmu & Kinf mu analysis
1.2
1
0.8
0.6
0.4
0.2
0
0.01
0.1
1
10
100
Kmu: robust perf.
Kinf: robust perf.

```
# Look at the worst case perturbations for each of
# the Kinf and Kmu controllers.  Compare also a
# random perturbation for each controller.  In all cases the
# perturbation is of size 0.5 and we choose a perturbation
# which is bad at 1Hz.

mupert = mkpert(Delta2,blk,rpbnds2,{fselect=1,pnorm=0.5,Hertz})
mupert = mupert(1,1)     # select part to close around top
infpert = mkpert(Delta1,blk,rpbnds1,{fselect=1,pnorm=0.5,Hertz})
infpert = infpert(1,1)   # select part to close around top

# set up an LFT for the reference tracking problem

M = daug(W,1,1)*consys([0,0,1;1,0,1;-1,1,-1])*daug(1,1,P)

mupertic = starp(mupert,M)
mupertclp = starp(mupertic,Kmu)
munomclp = starp(starp(0,M),Kmu)

infpertic = starp(infpert,M)
infpertclp = starp(infpertic,Kinf)
infnomclp = starp(starp(0,M),Kinf)

rpert = randpert(blk,{pnorm=0.5})
rpert = rpert(1,1)
rpertic = starp(rpert,M)
murpert = starp(rpertic,Kmu)
infrpert = starp(rpertic,Kinf)

# Look at step responses

time = [0:0.05:4]
step = gstep(time,0,1)

yinfnom = infnomclp*step
yinfpert = infpertclp*step
ymunom = munomclp*step
ymupert = mupertclp*step

gph4 = ctrlplot([step,yinfnom(1,1),yinfpert(1,1),...
```

```
                          ymunom(1,1),ymupert(1,1)]);
gph4 = plot(gph4,{legend=["input step";"Kinf nominal";...
           "Kinf pert.";"Kmu nominal";"Kmu pert."]})?
```


1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
1
2
3
4
input step
Kinf nominal
Kinf pert.
Kmu nominal
Kmu pert.

```
# Compare with a random perturbation

yinfrandp = infrpert*step
ymurandp = murpert*step

gph5 = ctrlplot([step,yinfnom(1,1),yinfrandp(1,1),...
                         ymunom(1,1),ymurandp(1,1)]);
gph5 = plot(gph5,{legend=["input step";"Kinf nominal";...
       "Kinf rand.  pert.";"Kmu nominal";"Kmu rand.  pert."]})?
```


1.4
1.2
1
0.8
0.6
0.4
0.2
0
0
1
2
3
4
input step
Kinf nominal
Kinf rand. pert.
Kmu nominal
Kmu rand. pert.

See Also:

`mu`, `hinfsyn`, `mkpert`, `hinfnorm`.

ophank

Syntax

```
[SysR,SysU,HSV] = ophank(Sys,{nsr,onepass})
```

Parameter List

Inputs:	Sys	Linear, stable, state-space system (continuous)
	nsr	(optional) Order of the reduced system. If not specified, the user will be prompted for its value after the Hankel singular values are displayed.
Keywords:	onepass	(Boolean) If equal to 1 (true), reduction is calculated in one pass. If false (!onepass or onepass=0), reduction is calculated in (number of states of Sys - nsr) passes. Defaults to 1.
Outputs:	SysR	Reduced order system; dynamic system object.
	SysU	Anti-causal optimal system, (only with `one` keyword); dynamic system object.
	HSV	A column vector containing the Hankel singular values of the system, Sys.

Description

Calculates an optimal hankel norm reduction of Sys for the additive case.

Any initial state values or state names associated with Sys are not assigned to SysR or SysU. Input and Output names are maintained.

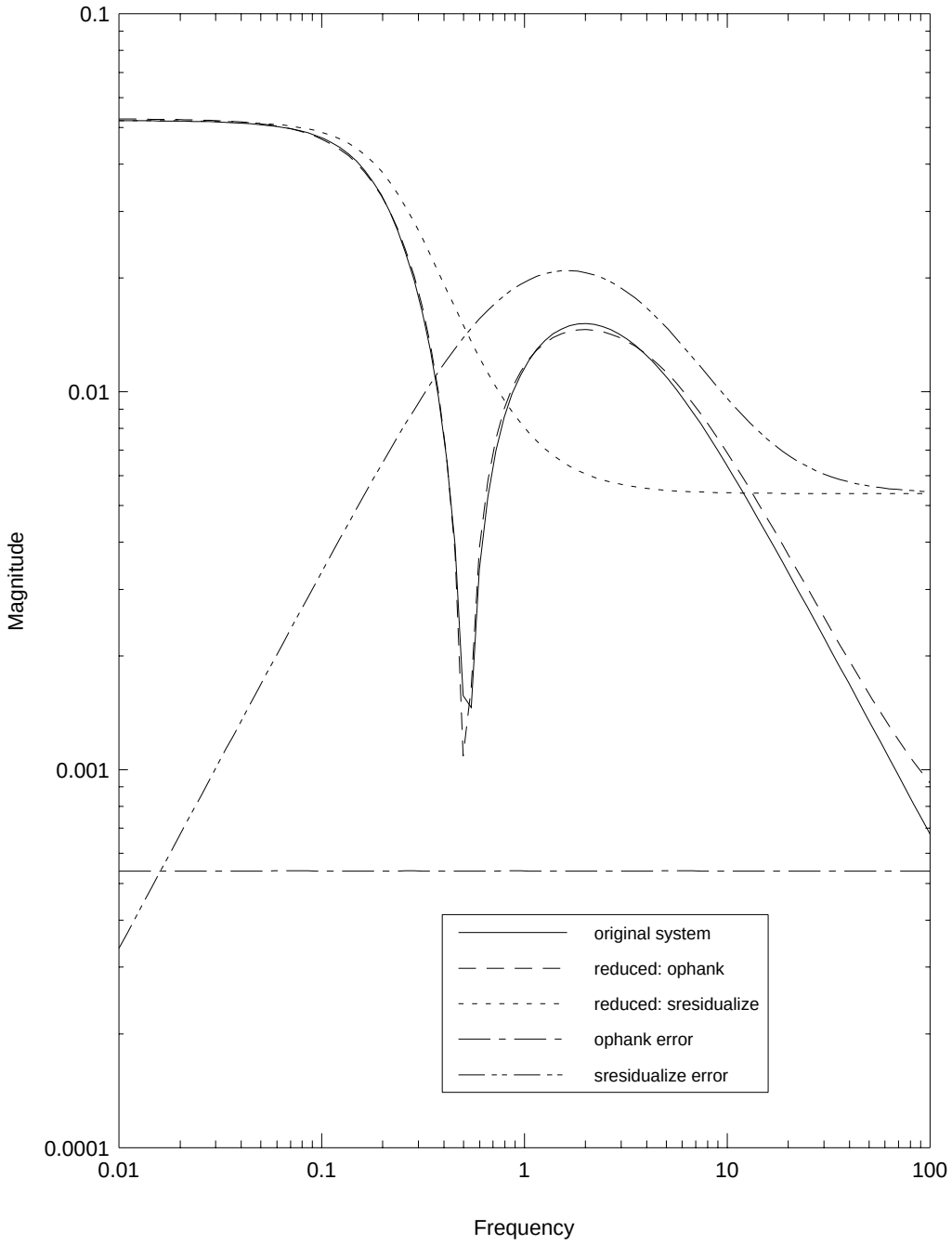
The current version of ophank is unable to deal with discrete time systems. Users are advised to call `makecontinuous` on discrete systems before calling `ophank`, and then re-discretize.

Uses additional subroutines `ophiter`, `ophred`, `ophmult` and `stable`.

This function is cross-licensed from the Model Reduction Module.

Example

```
# Create a five state system for reduction.

a = daug(-0.891334,[-1.20857,0.799042;-0.799042,-1.20857],...
         -4.74685,-21.3013)
b = [0.0262569;-0.189601;-0.113729;0.211465;-0.538239]
c = [0.120725,-0.336942,0.397198,-0.700524,-1.02235]
d = 0
sys1 = system(a,b,c,d)
fHz = logspace(0.01,100,100)
sys1g = freq(sys1,fHz)

# Reduce to 3 states by Hankel norm approximation

[sysout1,hsv] = ophank(sys1,{nsr=3})
sysout1g = freq(sysout1,fHz)
balerr = sys1g - sysout1g

# Reduce to a 3 state system by residualization
# for comparison purposes.

sysout2 = sresidualize(sys1,3)
sysout2g = freq(sysout2,fHz)
residerror = sys1g - sysout2g

gph1 = ctrlplot([sys1g,sysout1g,sysout2g,...
                 balerr,residerror],{logmagplot});
gph1 = plot(gph1,{!grid,legend=["original system";...
        "reduced:  ophank";"reduced:  sresidualize";...
        "ophank error";"sresidualize error"]})?
```


0.1
0.01
0.001
0.0001
Magnitude
0.01
0.1
1
10
100
Frequency
original system
reduced: ophank
reduced: sresidualize
ophank error
sresidualize error

See Also

`minimal`, `balmoore`.

orderstate

Syntax

```
outsys = orderstate(sys,indx)
```

Parameter List

Inputs:	sys	Input Dynamic System
	indx	Lists the desired order of the states in outsys.
Outputs:	outsys	output dynamic system.

Description

Reorder the states according to the index argument. The state names and initial condition are also ordered. The input and output names, and period (if discrete) are preserved.

Example

```
sys1 = randsys(3,1,1,{stable})
sys1 = system(sys1,{statenames=["s1";"s2";"s3"],...
                    x0=[0.1;0.2;0.3]})?

sys1 (a state space system) =

  A
  -2.27378     3.32295     -8.82005
   0.65518    -5.41526      9.0643
  -1.2488      4.7424     -13.2781

  B
  0.0879738
```

```
  0.710595
  0.688873

  C
  0.659532    0.181512    0.390497

  D
  0.15869

  X0
  0.1
  0.2
  0.3

  State Names
  -----------
  s1    s2    s3

  System is continuous

sys2 = orderstate(sys1,[2,1,3])?
sys2 (a state space system) =

  A
  -5.41526     0.65518      9.0643
   3.32295    -2.27378     -8.82005
   4.7424     -1.2488     -13.2781

  B
  0.710595
  0.0879738
  0.688873

  C
  0.181512    0.659532    0.390497

  D
  0.15869

  X0
```

```
0.2
0.1
0.3

State Names
-----------
s2    s1    s3

Input Names
-----------
Input 1

Output Names
------------
Output 1

System is continuous
```

randpdm

Syntax

```
pdmout = randpdm (ndomain,nrows,ncolumns,{keywords})
```

Parameter List

Inputs:	ndomain	length of the domain
	nrows	number of rows in pdmout
	ncolumns	number of columns in pdmout
Keywords:	complex	Boolean. A complex valued PDM is generated. Default = 0.
	zeromean	Boolean. The values are shifted so that zero is the mean. Default = 0.
	Dfirst	First value in the domain. This only a bound if !regular. Default = 0.
	Dlast	Last value in the domain. Again this is only a bound if !regular. Default = ndomain-1.
	regular	Boolean. Domain is regular. !regular generates a random domain between Dfirst and Dlast. Default = 1.
Outputs:	pdmout	Random PDM

Description

A random PDM, with user specified row, column and domain dimension, is generated. Several additional features can be specified by keywords: real or complex values, minimum and maximum values of the domain, and whether or not the domain is regular.

Note that the domain related defaults give a domain of [0:ndomain:1]. ndomain can be specified as zero in which case a random matrix is returned. This is a reasonable way of generating a random complex matrix.

Examples

```
pdm0 = randpdm(3,1,2,{zeromean})?

pdm0 (a pdm) =

domain |   Col 1       Col 2
-------+-------------------------
     0 |   -0.043917   -0.0789571
-------+-------------------------
     1 |    0.37878    -0.457265
-------+-------------------------
     2 |    0.0466939  -0.34413
-------+-------------------------

pdm1 = randpdm(4,1,1,{!regular,Dfirst=2,Dlast=19})?

pdm1 (a pdm) =

  domain |
---------+------------
 3.1543  |   0.0605523
---------+------------
 4.87235 |   0.900169
---------+------------
10.0323  |   0.932966
---------+------------
18.3191  |   0.645692
---------+------------

pdm2 = randpdm(pdm1)?

pdm2 (a pdm) =

  domain |
---------+------------
 3.1543  |   0.746793
---------+------------
 4.87235 |   0.174411
---------+------------
```

```
10.0323  |   0.922528
---------+------------
18.3191  |   0.81113
---------+------------

pdm3 = randpdm(0,3,2,{complex,zeromean})?

pdm3 (a rectangular matrix) =

  -0.237052  - 0.10845   j    -0.250958  - 0.392096 j
   0.097563  + 0.0140395 j    -0.255142  - 0.419362 j
   0.0178566 - 0.153032  j     0.0986898 + 0.375528 j
```

randpert

Syntax

```
[pert] = randpert(blk, {sys,sfreq,complex,pnorm})
```

Parameter List

Inputs:	blk	block structure (refer to mu function documentation). (matrix)
Keywords:	sys	Boolean. Specifies that pert is a dynamic system. Default = !sys, i.e. pert is a matrix.
	sfreq	Scalar valued. If a dynamic system is specified, sfreq is the frequency with significant phase. Units are Hertz. Default = 1.0.
	complex	Boolean. Specifies that pert is complex valued. Default = 1.
	pnorm	Scalar valued. Specifies the norm of the resulting system. Default = 1.
Outputs:	pert	Constructed perturbation

Description

Creates a perturbation of the structure specified by `blk`. By default this is a complex valued matrix. The `sys` keyword will create an allpass DYNAMIC SYSTEM. `sfreq` is used to specify a frequency (in Hertz) where there is significant phase in the system.

This function is used to construct random perturbations, the effects of which can be studied by simulation. Dynamic perturbations are appropriate for simulating the effects of unmodeled dynamics. In such cases, the crossover frequency, or a frequency where mu is large, are good choices.

Example

The use of `randpert` is studied in context in the on-line help for `musynfit` and the manual documentation for `musynfit` (page 299).

randsys

Syntax

```
sys = randsys (nstates,noutputs,ninputs,{keywords})
```

Parameter List

Inputs:	nstates	number of states in sys
	noutputs	number of outputs in sys
	ninputs	number of inputs in sys
Keywords:	stable	Boolean. sys is forced to be stable (default = 1)
	oscillatory	Boolean. Oscillatory poles are allowed. (default = 1). !oscillatory gives only real eigenvalues (continuous) or positive real eigenvalues (discrete).
	discrete	Boolean. Generate a discrete time system. (default = 0).
	dt	Sample period for discrete time system (default = 1/(2*Fmax)).
	Fmax	upper bound of sys pole frequencies (Hz) (default = 10 Hz or = 1/(2*dt) if dt specified)
	Fmin	lower bound of sys poles frequencies (Hz) (default = Fmax/100 Hz)
	Dterm	Select a random D term (default = 1: a D term is choosen)
	linear_dist	linear distribution of poles. The default is uniform (or normal) distribution on a log frequency axis. A linear frequency axis distribution may be more representative of some systems.
Outputs:	sys	Random Dynamic System

Description

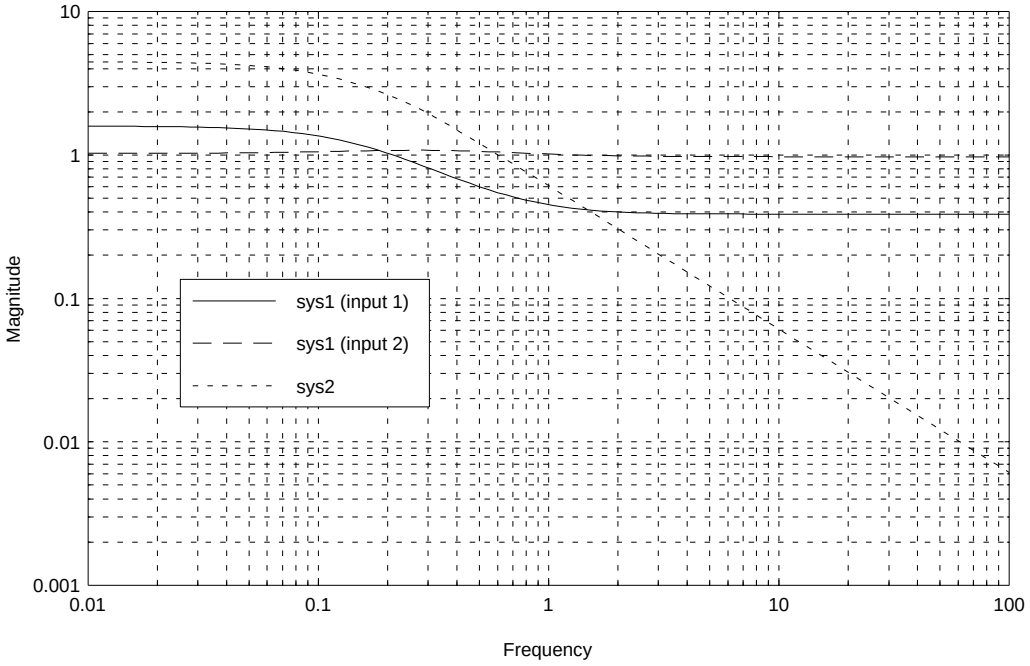
A random system, with user specified state, input and output dimension, is generated. Several additional features can be specified by keywords: whether or not the system is stable, where or not it can contain oscillatory modes, whether or not it has a D term, and bounds on the minimum and maximum pole frequencies.

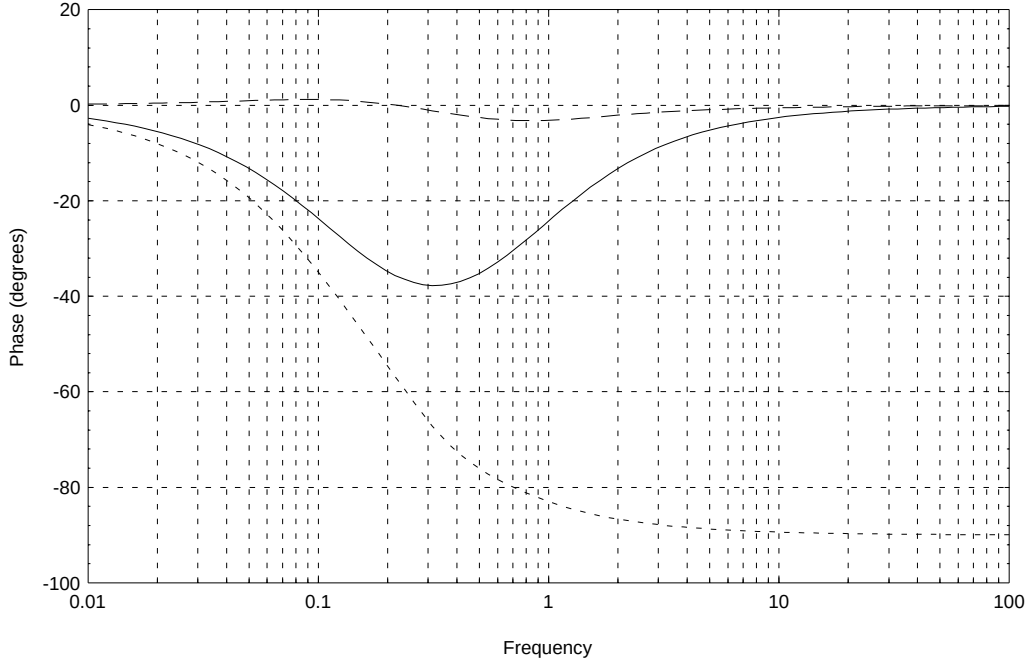
If the discrete keyword is specified, a bilinear transformation is performed to get the random discrete system. A sample time can be specified with dt.

Examples

```
sys1 = randsys(4,1,2,{stable,!oscillatory,...
                Fmin=0.1,Fmax=10})
size(sys1)?

ans (a row vector) =   1    2    4

rifd(sys1)?

Poles:

      real        imaginary      frequency      damping
                                 (rad/sec)       ratio

  -2.7920e+00     0.0000e+00     2.7920e+00     1.0000
  -5.2688e+00     0.0000e+00     5.2688e+00     1.0000
  -5.6953e+00     0.0000e+00     5.6953e+00     1.0000
  -5.3738e+01     0.0000e+00     5.3738e+01     1.0000


Zeros:

sys2= randsys(10,1,1,{stable,oscillatory,Fmin=0.1,...
        Fmax=1,!Dterm})
rifd(sys2)?

Poles:
```

```
      real        imaginary      frequency      damping
                                 (rad/sec)       ratio

  -1.0408e+00     0.0000e+00     1.0408e+00     1.0000
  -1.1390e+00     0.0000e+00     1.1390e+00     1.0000
  -2.8050e-01    -1.4858e+00     1.5120e+00     0.1855
  -2.8050e-01     1.4858e+00     1.5120e+00     0.1855
  -1.5337e+00     0.0000e+00     1.5337e+00     1.0000
  -5.5566e-01    -1.6958e+00     1.7845e+00     0.3114
  -5.5566e-01     1.6958e+00     1.7845e+00     0.3114
  -3.1454e+00     0.0000e+00     3.1454e+00     1.0000
  -5.6798e+00     0.0000e+00     5.6798e+00     1.0000
  -6.1039e+00     0.0000e+00     6.1039e+00     1.0000


Zeros:

      real        imaginary      frequency      damping
                                 (rad/sec)       ratio

  -8.6788e-02    -1.0161e+00     1.0198e+00     0.0851
  -8.6788e-02     1.0161e+00     1.0198e+00     0.0851
  -1.0395e+00     0.0000e+00     1.0395e+00     1.0000
  -1.5152e+00     0.0000e+00     1.5152e+00     1.0000
  -6.3796e-01    -1.9633e+00     2.0643e+00     0.3090
  -6.3796e-01     1.9633e+00     2.0643e+00     0.3090
  -3.1425e+00     0.0000e+00     3.1425e+00     1.0000
  -5.8916e+00     1.5472e-01     5.8937e+00     0.9997
  -5.8916e+00    -1.5472e-01     5.8937e+00     0.9997


fHz = logspace(0.01,100,100)
sys1g = freq(sys1,fHz)
sys2g = freq(sys2,fHz)

gph1 = ctrlplot([sys1g,sys2g],{bode});
gph1 = plot(gph1,{legend=["sys1 (input 1)";...
                "sys1 (input 2)";"sys2"]})?
```


10
1
0.1
0.01
0.001
Magnitude
sys1 (input 1)
sys1 (input 2)
sys2
0.01
0.1
1
10
100
Frequency


20
0
-20
-40
-60
-80
-100
Phase (degrees)
0.01
0.1
1
10
100
Frequency

```
sys3 = randsys(6,1,1,{discrete,dt=5})
rifd(sys3)?


Poles:

      radius       angle
                  (radians)

    0.1652      0.0000
    0.4580      0.0000
    0.7872     -0.0394
    0.7872      0.0394
    0.9307     -0.2163
    0.9307      0.2163


Zeros:

      radius       angle
                  (radians)

    0.1623      0.0000
    0.7981      0.0000
    0.6658      0.0000
    0.8921     -0.1981
    0.8921      0.1981
   15.4278      3.1416
```

rifd

Syntax

```
[stat] = rifd(vec,{discrete,Hertz,degrees})
```

Parameter List

Inputs:	vec	complex valued vector (or DYNAMIC SYSTEM - see below).
Keywords:	discrete	Boolean. Vector is to be interpreted in the z domain, rather than the s domain. Default = 0.
	Hertz	Boolean. Display frequency units in Hertz on the s plane. !Hertz gives a display in radians/sec. Default = 0.
	degrees	Boolean. Display angle in degrees for z plane. !degrees gives a display in radians. Default = 0.
Outputs:	stat	Status. stat = 1 if an error occurs.

Description

Displays complex valued vectors in terms of s or z domain properties. The primary use is for interpreting the output of poles or zeros in engineering units.

For the s domain, the real and imaginary parts and frequency and damping are displayed.

For the z domain, the pole radius and angle are displayed. Note that the angle is actually the normalized frequency (radians/sample).

If `vec` is a DYNAMIC SYSTEM, the poles and zeros are calculated and displayed. This is a useful shorthand for the most common usage: `rifd(poles(system))`. In this case there is no need to specify the discrete keyword as this is be determined directly from the DYNAMIC SYSTEM.

Examples

```
sys1 = randsys(4,3,2,{stable})
rifd(sys1)

Poles:

      real        imaginary      frequency      damping
                                 (rad/sec)       ratio

  -1.6847e+00     0.0000e+00     1.6847e+00     1.0000
  -2.4383e+00     0.0000e+00     2.4383e+00     1.0000
  -8.7457e+00     0.0000e+00     8.7457e+00     1.0000
  -1.5041e+01     0.0000e+00     1.5041e+01     1.0000


Zeros:
ans (a scalar) =   0

# Compare rifd to A matrix eigenvalues

[a,b,c,d] = abcd(sys1)
Aeigs = eig(a)?

Aeigs (a column vector) =

   -1.68474
   -2.43829
   -8.7457
  -15.0408

rifd(Aeigs)

      real        imaginary      frequency      damping
                                 (rad/sec)       ratio

  -1.6847e+00     0.0000e+00     1.6847e+00     1.0000
  -2.4383e+00     0.0000e+00     2.4383e+00     1.0000
  -8.7457e+00     0.0000e+00     8.7457e+00     1.0000
  -1.5041e+01     0.0000e+00     1.5041e+01     1.0000
```

```
sys2 = randsys(3,3,2,{stable,discrete})
rifd(sys2)


Poles:

      radius       angle
                 (radians)

    0.9641      0.0000
    0.2985     -0.5348
    0.2985      0.5348


Zeros:
```

sdtrsp

Syntax

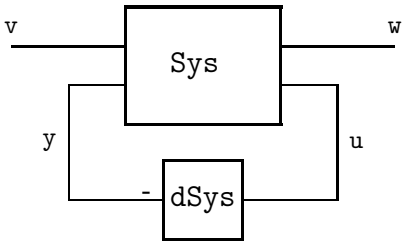
```
[v,y,u] = sdtrsp(Sys,dSys,w,tfinal,...
                              {ord,intstep,cdelay})
```

Parameter List

Inputs:	Sys	Continuous dynamic system. This is the upper system in the LFT. The initial states are used in the simulation.
	dSys	Digital dynamic system. Lower system in the LFT. The initial states are used in the simulation.
	w	PDM. Input signal.
	tfinal	Final time in the simulation (optional). Default = max time specified in w.
Keywords:	ord	Scalar valued. Specifies order of interpolation for continuous signals in the system. Options are 0 or 1. Default = 0.
	intstep	Scalar valued. Integration step size. If not supplied a default will be calculated based on the system eigenvalues. It will be rounded to make it divide into the discrete system sample time by an integer value of at least 2.
	cdelay	Scalar value. Delay implemented at the output to dSys. (in sec.) Optional, default = 0. It must be less than the digital system sample time.
Outputs:	v	Signal from upper system in the LFT.
	y	Signal into lower system in the LFT.
	u	Signal out of the lower system in the LFT.

Description

Time domain simulation of a sampled data interconnection. The applicable closed loop system is illustrated below.



This is conceptually the equivalent of:

$$\mathtt{v} = \mathtt{starp}(\mathtt{Sys}, \mathtt{dSys}) * \mathtt{w}.$$

This function will handle interconnections in which the continuous time signals are not necessarily synchronized with the digital system. Computation delays which are a fraction of the sample period may also be simulated. The following equations represent the discrete system,

$$\begin{aligned} z(kT+T) &= A_d z(kT) + B_d y(kT) \\ u(kT+cdelay) &= C_d z(kT) + D_d y(kT) \end{aligned}$$

For delays of greater than a sample period append additional states to dSys to model the integer sample period part of the delay. Use cdelay to model the fractional remainder.

The actual calculation is performed with a fast discrete time equivalent. Intstep is the fast integration step size. The input vector, w will be interpolated to the same step size. Either a zero or first order interpolation is used, depending on the keyword ord. If ord = 0 a zero-order hold equivalent is used for the continuous plant. In the ord = 1 case a

triangle hold equivalent is used. This is the same a linearly connecting the samples at the input to the hold.

Some care is needed in the choice of cdelay and intstep. The default for intstep is based on the continuous system eigenvalues and the minimum time spacing in the input vector, w. It will be forced to be an integer divisor of the digital sample time. Intstep must also be a divisor of cdelay. The user can inadvertently force a very small integration period by selecting cdelay without regard to the sample period. Warnings are printed if this is suspected to be the case.

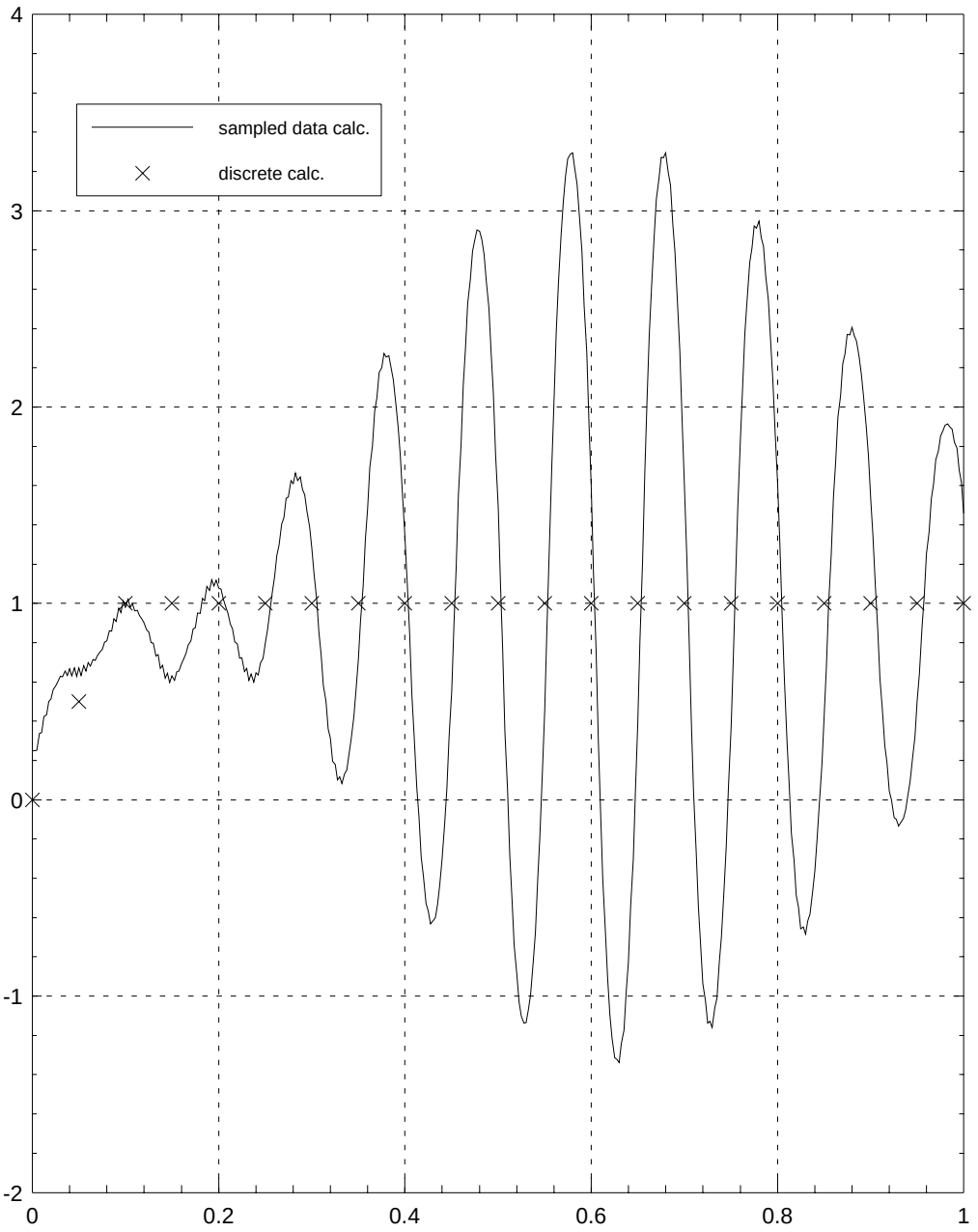
The upper LTF block (continuous sytem) can be a constant matrix. The lower block (digital system) must be a discrete dynamic system in order to specify the sampling time. If a constant gain digital system is required specify it with,

dSys = system([], [], [], Dd, T).

Example:

```
# Consider a continuous time plant,

P = 1/makepoly([1,0,-0.05],"s")
[a,b,c,d] = abcd(P)
P = system(a,b,c,d)
T = 1/20         # sample period

# The following digital controller is used.
# The first input is the reference and the # second is the
measurement.

digC = system([-1.5,T/4; -2/T,-.5],[.5,2;1/T,1/T],...
                [-1/T^2,-1.5/T], [1/T^2,0],T)

# Now consider a sample & hold version of P and
# check that the closed loop digital system is stable
# and has a reasonable step response.

digP = discretize(P,T,exponential)

snm = ["digP"; "digC"]
```

```
inps = "ref"
outs = "digP"
cnx = ["digC"; ...        # input to digP
"[ ref;digP]"]     # input to digC
digclp = sysic(snm,inps,outs,cnx,digP,digC)

# Calculate the digital system step response

time = [0:T:20*T]
step = gstep(time,0,1)
digy = digclp*step

# Do a complete sampled data simulation.  We
# will assume that the controller has an input to
# output calculation delay of 0.1*T. The
# plant will also have a non-zero initial
# condition and a sinusoidal disturbance will
# be imposed at the output.  The disturbance has
# deliberately been chosen at above the Nyquist
# frequency.

#P = system(P,x0=[0.25;0])            # initial condition
calcdelay = 0.1*T                      # calculation delay
finetime = [0:0.005:1]
dist=0.25*gsin(finetime,frequency=11) # disturbance
step = gstep(finetime,0,1)             # step

# calculate the continuous time part of the
# system as the upper LFT object.

ctssys = consys([0,1,1;1,0,0;0,1,1])*daug(1,1,P)
ctssys = system(ctssys,x0=[0.25;0])
ctsy = sdtrsp(ctssys,digC,[step;dist],...
                cdelay=calcdelay)

gph1 = ctrlplot(ctsy);
gph1 = plot(gph1,digy,{line=0,marker=1,...
legend= ["sampled data calc.";"discrete calc."]})?
```


sampled data calc.
discrete calc.
4
3
2
1
0
-1
-2
0
0.2
0.4
0.6
0.8
1

See Also

`trsp`

simtransform

Syntax

```
out = simtransform(sys,X)
```

Parameter List

Inputs:	sys	Input system. This may be a state-space system, PDM or constant matrix.
	X	Similarity transform. X must be invertible
Outputs:	out	output system - in the same class as the input.

Description

Apply a similarity transform to `sys`. If `sys` is a matrix or PDM this gives `out = inv(X)*sys*X`. If sys is a DYNAMIC SYSTEM this transform is applied to the state.

Applying this transform to a transfer function is meaningless as a state matrix has not been uniquely defined. This will return a warning and leave the transfer function unchanged.

This command will remove the state names as they no longer have any meaning.

Example

```
sys1 = randsys(3,1,1,{stable})
rifd(sys1)

Poles:

      real        imaginary      frequency      damping
                                 (rad/sec)       ratio
```

```
  -1.9737e+01     0.0000e+00     1.9737e+01     1.0000
  -9.7569e+00    -2.9129e+01     3.0720e+01     0.3176
  -9.7569e+00     2.9129e+01     3.0720e+01     0.3176


Zeros:

      real        imaginary      frequency      damping
                                 (rad/sec)       ratio

  -2.1990e+01     0.0000e+00     2.1990e+01     1.0000
  -9.7962e+00    -2.9186e+01     3.0786e+01     0.3182
  -9.7962e+00     2.9186e+01     3.0786e+01     0.3182

[a,b,c,d] = abcd(sys1)

# Do a schur transform on the A matrix
# and use for a similarity transform on the system.

[as,X] = schur(a,{real})
sysout = simtransform(sys1,X)?

sysout (a state space system) =

  A
  -19.737            3.66249     4.10895
    0               -9.75694    49.404
   -8.88178e-16    -17.1748     -9.75694

  B
   1.11535
   0.042032
  -0.151334

  C
  1.17657    -0.369351    -0.163514

  D
  0.566721
```

```
X0
0
0
0


Input Names
-----------
Input 1

Output Names
------------
Output 1

System is continuous
```

spectrad

Syntax

```
out = spectrad(mat)
```

Parameter List

Inputs:	mat	Square matrix or PDM.
Outputs:	out	spectral radius of the input.

Description

Calculates the spectral radius (magnitude of the maximum eigenvalue) of the input matrix.

Example

```
pdm1 = randpdm(3,2,2,{zeromean})?

pdm1 (a pdm) =

domain |        Col 1       Col 2
-------+-----------------------------
     0 | Row 1   0.0504605  -0.0914956
       | Row 2   0.221744   -0.0231464
-------+-----------------------------
     1 | Row 1   0.139306    0.496387
       | Row 2  -0.342521    0.0350694
-------+-----------------------------
     2 | Row 1  -0.287094    0.0591451
       | Row 2  -0.0695034  -0.477195
-------+-----------------------------
```

```
eig(pdm1)?

ans (a pdm) =

domain |
-------+-------------------------------
     0 | Row 1   0.013657  + 0.137601 j
       | Row 2   0.013657  - 0.137601 j
-------+-------------------------------
     1 | Row 1   0.0871876 + 0.409031 j
       | Row 2   0.0871876 - 0.409031 j
-------+-------------------------------
     2 | Row 1  -0.311974
       | Row 2  -0.452314
-------+-------------------------------

spectrad(pdm1)?

ans (a pdm) =

domain |
-------+-----------
     0 |   0.138277
-------+-----------
     1 |   0.41822
-------+-----------
     2 |   0.452314
-------+-----------
```

sresidualize

Syntax

```
sysout = sresidualize(sysin,ord)
```

Parameter List

Inputs:	sysin	Input Dynamic System
	ord	Order of sysout
Outputs:	sysout	output Dynamic System

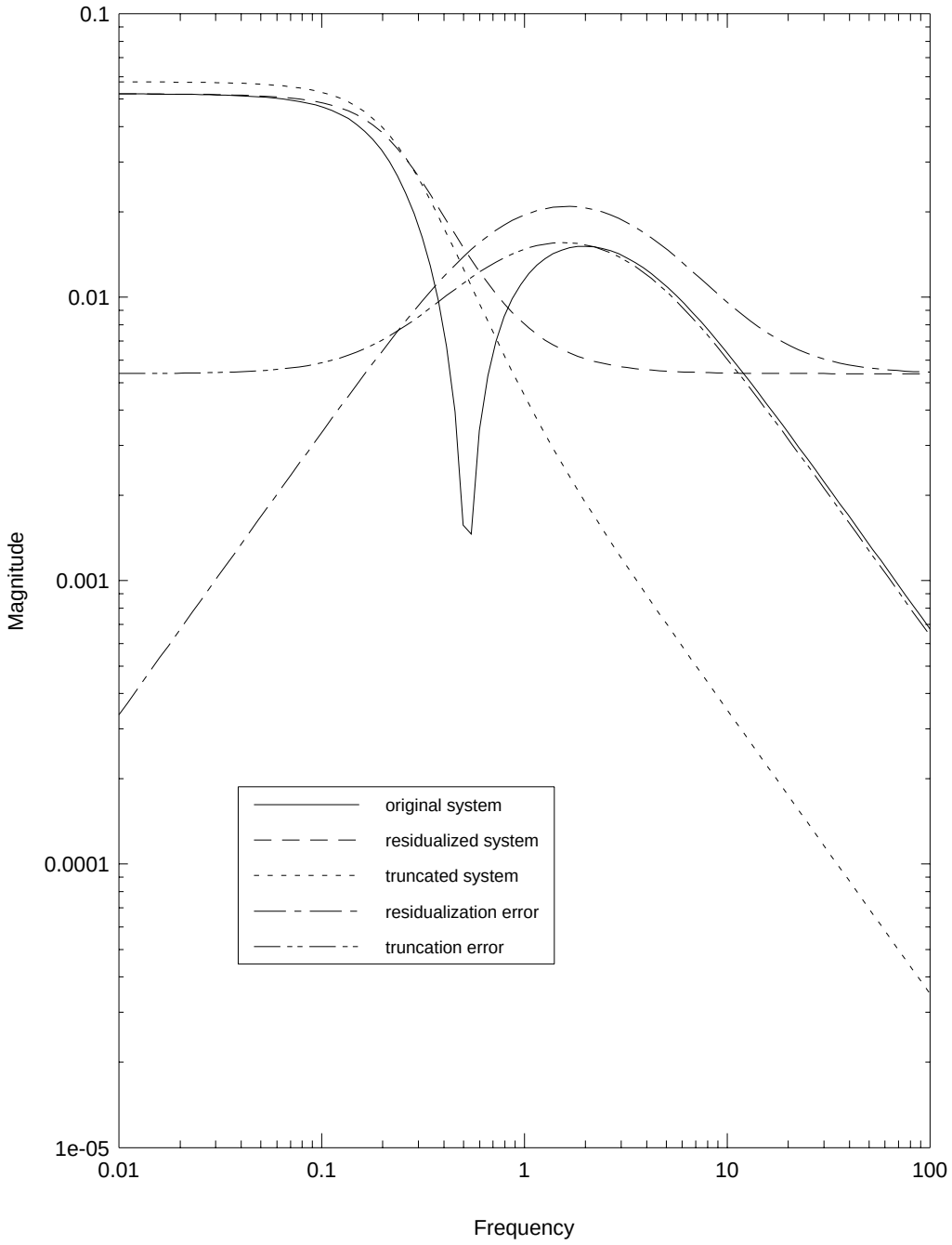
Description

Residualize the states of `sysin` to the number specified by `ord`. The last $n_x - $`ord` states ($n_x$ is the number of states in `sysin`) are residualized. If `ord` is greater than the number of states in `sysin` then `sysout` = `sysin` and a warning is displayed. `sysin` may be a constant matrix, in which case it is treated as a system with zero states.

Example

```
# Create a five state system for reduction.

a = daug(-0.891334,[-1.20857,0.799042;-0.799042,-1.20857],...
             -4.74685,-21.3013)
b = [0.0262569;-0.189601;-0.113729;0.211465;-0.538239]
c = [0.120725,-0.336942,0.397198,-0.700524,-1.02235]
d = 0
sys1 = system(a,b,c,d)

# Reduce to a 3 state system by residualization
# and truncation.
```

```
sysout1 = sresidualize(sys1,3)
sysout2 = truncate(sys1,3)

fHz = logspace(0.01,100,100)
sys1g = freq(sys1,fHz)
sysout1g = freq(sysout1,fHz)
sysout2g = freq(sysout2,fHz)
residerror = sys1g - sysout1g
truncerror = sys1g - sysout2g

gph1 = ctrlplot([sys1g,sysout1g,sysout2g,residerror,...
                          truncerror],{logmagplot});
gph1 = plot(gph1,{!grid,legend=["original system";...
         "residualized system";"truncated system";...
         "residualization error";"truncation error"]})?
```


0.1
0.01
0.001
0.0001
1e-05
Magnitude
0.01
0.1
1
10
100
Frequency
original system
residualized system
truncated system
residualization error
truncation error

See also

`rifd`, `simtransform`, `orderstate` `modalstate`, `truncate`.

starp

Syntax

```
out = starp (upper,lower,dim1,dim2,skipChks)
```

Parameter List

Inputs:	upper	Upper object (DYNAMIC SYSTEM, constant or PDM) in the interconnection
	lower	Lower object in the interconnection.
	dim1	The number of outputs of upper to be connected as inputs to lower. (Optional - see description for default)
	dim2	The number of outputs of lower to be connected as inputs to upper. (Optional - see description for default)
Keywords:	skipChks	(Boolean, default = 0). Don't check input arguments – this includes type (DYNAMIC SYSTEM, PDM) consistency and fractional well posedness. For PDMs, the well posedness test may involve significant computation.
Outputs:	out	Resulting interconnection.

Description

Connects two objects in a Redheffer "star product," as illustrated in the following diagram.

This results in the following system.

Constant matrices can be interconnected with either pdms or dynamical systems. As expected a DYNAMIC SYSTEM and a PDM cannot be interconnected in this way.

If `dim1` and `dim2` are omitted, it is assumed that a linear fractional transformation is

specified. This is equivalent to:

$$\texttt{dim1} = \min(\texttt{upper\_output\_dim}, \texttt{lower\_input\_dim})$$

and

$$\texttt{dim2} = \min(\texttt{upper\_input\_dim}, \texttt{lower\_output\_dim}).$$

Examples

```
# Look at a constant matrix,

M = random(4,4)
lower = [3,4]

# The interconnected LFT will have 2 rows and 3 columns

result = starp(M,lower)?

result (a rectangular matrix) =

   0.261939    -0.558202    0.484644
  -0.648616    -1.11273     0.107883

# This is equivalent to:

M11 = M(1:2,1:3)
M12 = M(1:2,4)
M21 = M(3:4,1:3)
M22 = M(3:4,4)
M11 + M12*lower*inv(eye(2,2)-M22*lower)*M21?

ans (a rectangular matrix) =

   0.261939    -0.558202    0.484644
  -0.648616    -1.11273     0.107883

# A much more common usage is to provide an interconnection
```

```
# structure for closing control loops.  The following is
# the structure for a simple unity gain negative feedback
# system with plant P.

P = 1/makepoly([1,1],"s")
M = consys([0,1;1,-1])*daug(1,P)

# Test this with a Proportional and PI controller

Kp = 10
KpInt = makepoly([10,100],"s")/makepoly([1,0],"s")

rifd(starp(M,Kp))          # look at poles and zeros

Poles:

      real        imaginary      frequency      damping
                                 (rad/sec)       ratio

  -1.1000e+01     0.0000e+00     1.1000e+01     1.0000


Zeros:


rifd(starp(M,KpInt))

Poles:

      real        imaginary      frequency      damping
                                 (rad/sec)       ratio

  -5.5000e+00    -8.3516e+00     1.0000e+01     0.5500
  -5.5000e+00     8.3516e+00     1.0000e+01     0.5500


Zeros:

      real        imaginary      frequency      damping
                                 (rad/sec)       ratio
```

```
  -1.0000e+01     0.0000e+00     1.0000e+01     1.0000


# Now consider the system with an output multiplicative
# perturbation (of 10%)

W = consys(0.1)
G = daug(W,1,1)*consys([0,0,1;1,0,1;-1,1,-1])*daug(1,1,P)

# The nominal system is constructed by closing the
# upper loop with 0.  Any thing else is a perturbed
# system.

Gnom = starp(0,G)
Gpert = starp(-1,G)

# Now examine the closed loop affects for the PI
# controller.

rifd(starp(Gnom,KpInt))         # same as before

Poles:

      real        imaginary      frequency      damping
                                 (rad/sec)       ratio

  -5.5000e+00    -8.3516e+00     1.0000e+01     0.5500
  -5.5000e+00     8.3516e+00     1.0000e+01     0.5500


Zeros:

      real        imaginary      frequency      damping
                                 (rad/sec)       ratio

  -1.0000e+01     0.0000e+00     1.0000e+01     1.0000


rifd(starp(Gpert,KpInt))        # perturbation moves the poles.
```

Poles:

real	imaginary	frequency (rad/sec)	damping ratio
-5.0000e+00	-8.0623e+00	9.4868e+00	0.5270
-5.0000e+00	8.0623e+00	9.4868e+00	0.5270

Zeros:

real	imaginary	frequency (rad/sec)	damping ratio
-1.0000e+01	0.0000e+00	1.0000e+01	1.0000

substr

Syntax

```
littlestring = substr(bigstring,charindex,{skipChks})
```

Parameter List

Inputs:	bigstring	Input string (a 1×1 string matrix)
	charindex	vector indexing the characters to be returned in the output.
Keywords:	skipChks	Boolean specifying that syntax checking is to be skipped.
Outputs:	littlestring	Output string

Description

`substr` takes characters from the string, `bigstring` and concatentates them to from the output, `littlestring`. The vector, `charindex`, determines the characters and their order.

This function allows the user to select substrings from an input string. However, as shown in the example, there is no requirement that the characters specified in charindex be contiguous or non-repeated.

To find a specified character in a string use the function: index.

Example

```
alphabet = " abcdefghijklmnopqrstuvwxyz"
mantra = substr(alphabet,[14,22,1,19,22,13,6,20])?
```

sysic

Syntax

```
[sys] = sysic(sysNames,sysInputs,sysOutputs,sysConnects,...
                              subsys1,subsys2,...)
```

Parameter List

Inputs:	sysNames	A vector of strings (of the same length as the number of subsystems) naming the subsystems.
	sysInputs	A vector of strings naming the exogenous inputs to the final system. Each named input must be a different row — however parenthesis can be used to denote vector valued input names.
	sysOutputs	A vector of strings defining the exogenous outputs of the final system. These are defined in terms of input names or subsystem outputs.
	sysConnects	A vector of strings (of the same length as the number of subsystems) defining the input to each subsystem.
	subsys1	Subsystems to be connected. The subsystems can be combinations of matrices and dynamic systems or combinations of matrices and pdms.
	⋮	"
Outputs:	sys	The interconnected system, which can be either a dynamic system, a pdm, or a matrix depending on the subsystems.

Description

`sysic` interconnects subsystems to form a single, larger system. The interconnection specification strings and the actual subsystems are passed as arguments to `sysic`. In this context, systems can be either DYNAMIC SYSTEMS or PDMS. An interconnection with both DYNAMIC SYSTEMS and PDMS is not well defined and will generate an error.

Matrices can also be included in the interconnection and are considered to be constant gains when connected with DYNAMIC SYSTEMs and considered to be constant for all domain values when connected with PDMs.

`sysic` is able to connect linear systems together. For more complete interconnection and simulation capabilities SystemBuild should be used.

Limitations

Only 20 subsystems can be interconnected with a single `sysic` call.

Example

```
# A standard unity gain negative feedback system
# is set up.  To illustrate some of the capabilities
# an output noise and plant input disturbance are also
# added.  The output signals are the plant output,
# controller effort and 10 x reference error.

p = 1/makepoly([1,0.1,1],"s")
c = makepoly([1,1],"s")/makepoly([1,10],"s")

snames = ["p";               # name for subsystem 1
          "c"]
inputs = ["ref";             # the name is arbitrary
          "dist(2)"]         # this input is a dim 2 vector
outputs = ["p + dist(1)";    # note reference to first dist input
           "c";
           "10*ref - 10*p"]  # linear combinations can be used
conx = ["c + dist(2)";       # input to snames(1), i.e.  "p"
        "ref - p - dist(1)"] # input to "c"

clp = sysic(snames,inputs,outputs,conx,p,c)
```

trsp

Syntax

```
[y,uint] = trsp(Sys,u,tfinal, ord,intstep)
```

Parameter List

Inputs:	Sys	Continuous dynamic system. The initial states are used in the simulation.
	u	PDM. Input signal.
	tfinal	Final time in the simulation (optional). Default = max time specified in u.
Keywords:	ord	Scalar valued. Specifies order of interpolation for continuous signals in the system. Options are 0 or 1. Default = 0.
	intstep	Scalar valued. Integration step size. If not supplied a default will be calculated based on the system eigenvalues. It will be rounded to make it divide into the discrete system sample time by an integer value of at least 2.
Outputs:	y	PDM. Output signal.
	uint	Interpolated version of u.

Description

Time domain simulation of a continuous system. This is conceptually the equivalent of:

$$\texttt{y} = \texttt{Sys} * \texttt{u}.$$

This function allows the user more control over the parameters in the simulation.

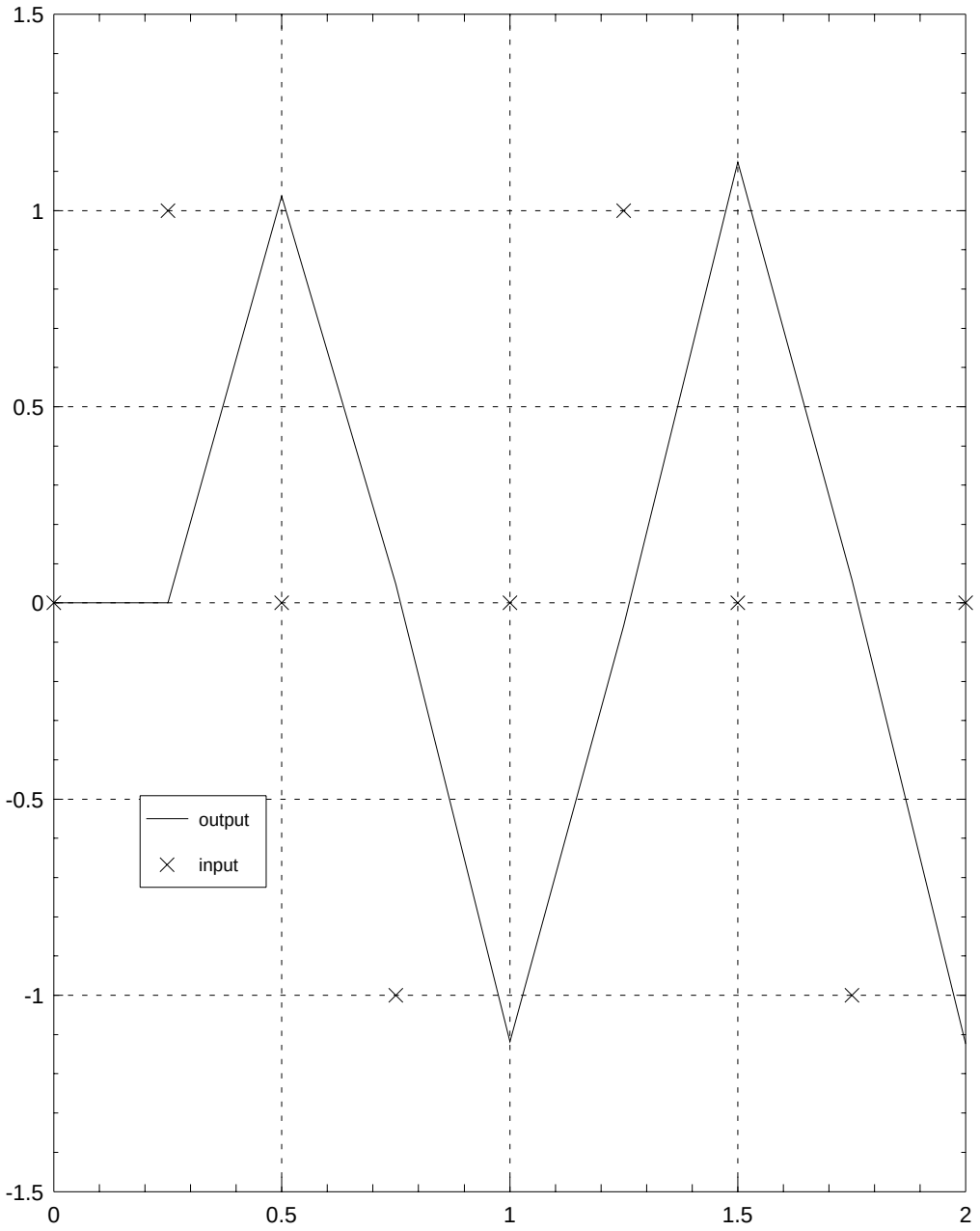
The user can select an integration step (`intstep`) independently of the data spacing in `u`. Furthermore, the spacing of `u` can be non-regular and `u` will be interpolated if

necessary. If intstep is not specified the integration stepsize is determined from the system eigenvalues and the minimum spacing in the input signal, `u`.

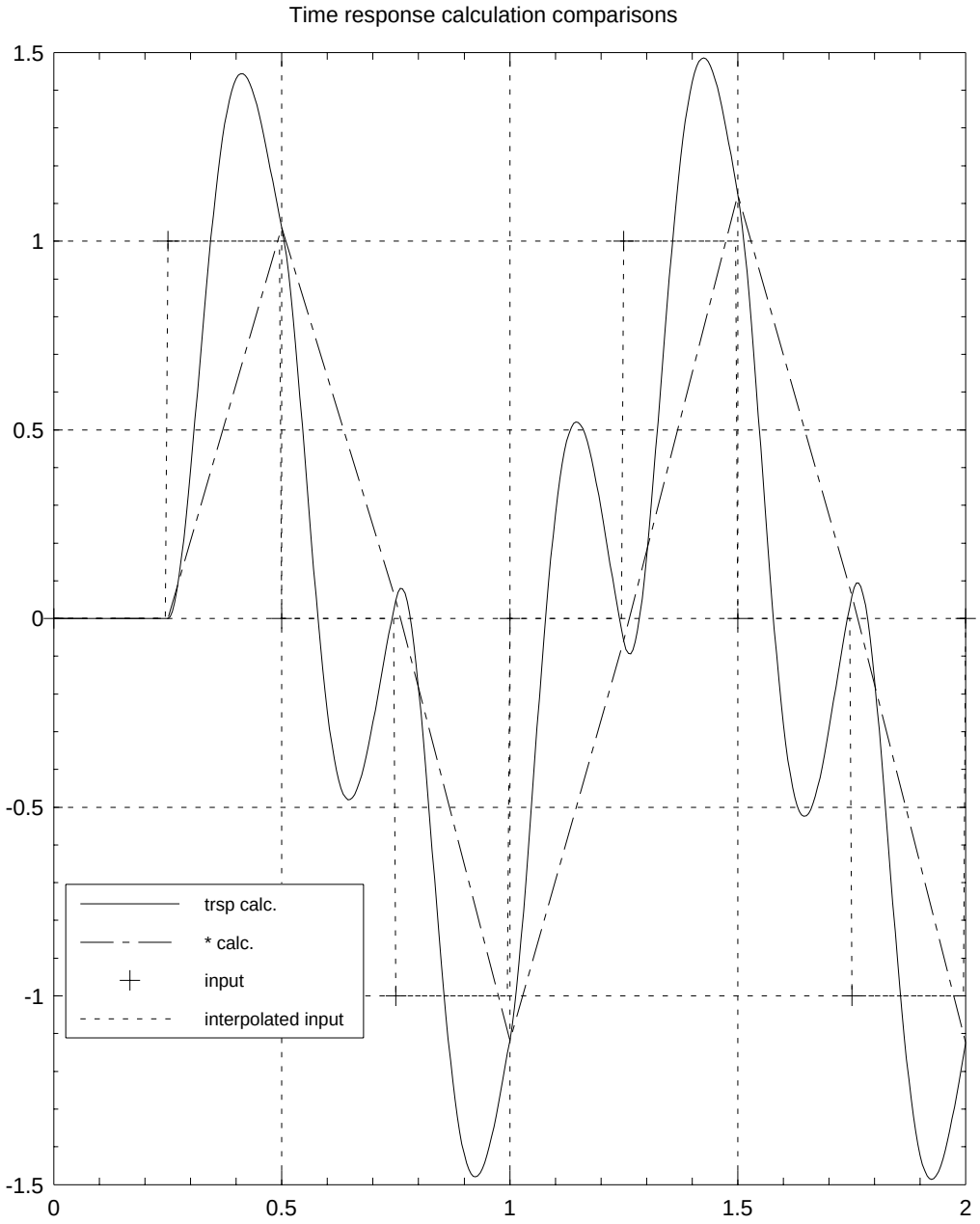
A zero or first order discrete equivalent of the systems can be specified. The standard Xmath `*` operator uses only a zero order discretization. The first order simulation actually uses a triangle hold equivalent. This is equivalent to connecting the samples going into the hold function. Although the continuous time hold is noncausal the discrete state-space system is causal. This is often more accurate at higher frequencies.

Examples

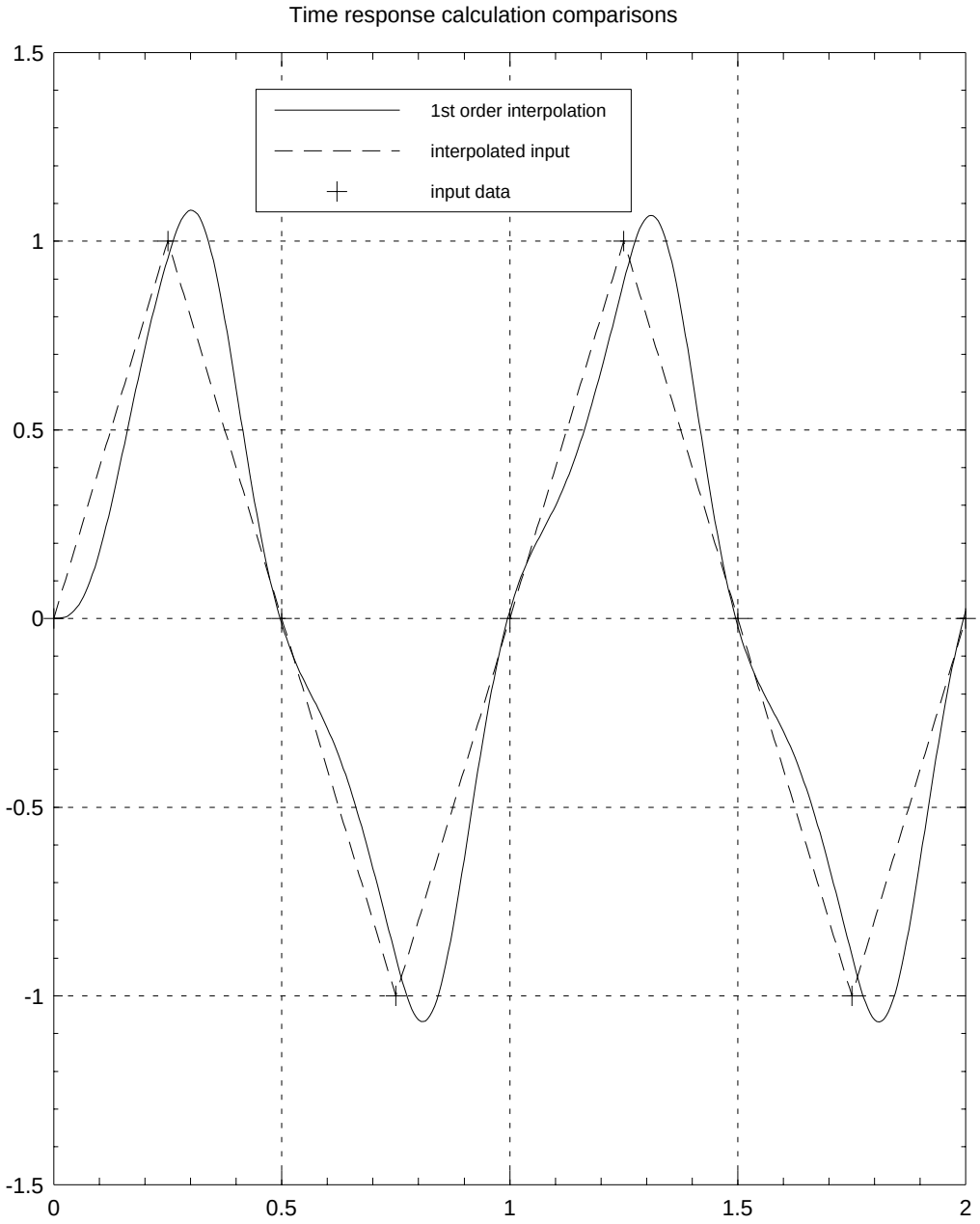
```
# We will use trsp to simulate the output of
# a second order system for various inputs

P = 400/makepoly([1,10,400],"s")

time = [0:0.25:2]
u = gstep(time,time,[0,1,0,-1,0,1,0,-1,0])

# Examine the standard Xmath simulation

y = P*u
gph1 = ctrlplot(y);
gph1 = plot(u,gph1,{marker=1,line=0,legend=["output";"input"]})?
```


1.5
1
0.5
0
-0.5
-1
-1.5
0
0.5
1
1.5
2
output
input

```
# Compare trsp calculation to standard
[ytrsp,uint] = trsp(P,u)
gph2 = ctrlplot(ytrsp);
gph2 = plot(y,gph2,{line_style=4});
gph2 = plot(u,gph2,{line=0,marker=1});
gph2 = plot(uint,gph2,{line_style=3,legend=["trsp calc.";...
             "* calc.";"input";"interpolated input"],title=...
             "Time response calculation comparisons"})?
```


Time response calculation comparisons
1.5
1
0.5
0
-0.5
-1
-1.5
0
0.5
1
1.5
2
trsp calc.
* calc.
input
interpolated input

```
# Now look at 1st order interpolation

[y1trsp,u1int] = trsp(P,u,{ord=1})
gph3 = ctrlplot([y1trsp,u1int]);
gph3 = plot(u,gph3,{line=0,marker=1,legend=...
         ["1st order interpolation";...
         "interpolated input";"input data"],title=...
         "Time response calculation comparisons"})?
```


Time response calculation comparisons
1st order interpolation
interpolated input
input data
1.5
1
0.5
0
-0.5
-1
-1.5
0
0.5
1
1.5
2

truncate

Syntax

```
sysout = truncate(sysin,ord)
```

Parameter List

Inputs:	sysin	Input Dynamic System
	ord	Order of the truncated system: sysout
Outputs:	sysout	Truncated output Dynamic System

Description

The function `truncate` is cross-licensed from the model reduction toolbox and has slightly more capabilities than those described here. See the online help for further details.

Consider a partitioning of the input Dynamic System, `sysin`, as follows.

$$\texttt{sysin} = \left[\begin{array}{cc|c} A_{11} & A_{12} & B_1 \\ A_{21} & A_{22} & B_2 \\ \hline C_1 & C_2 & D \end{array}\right],$$

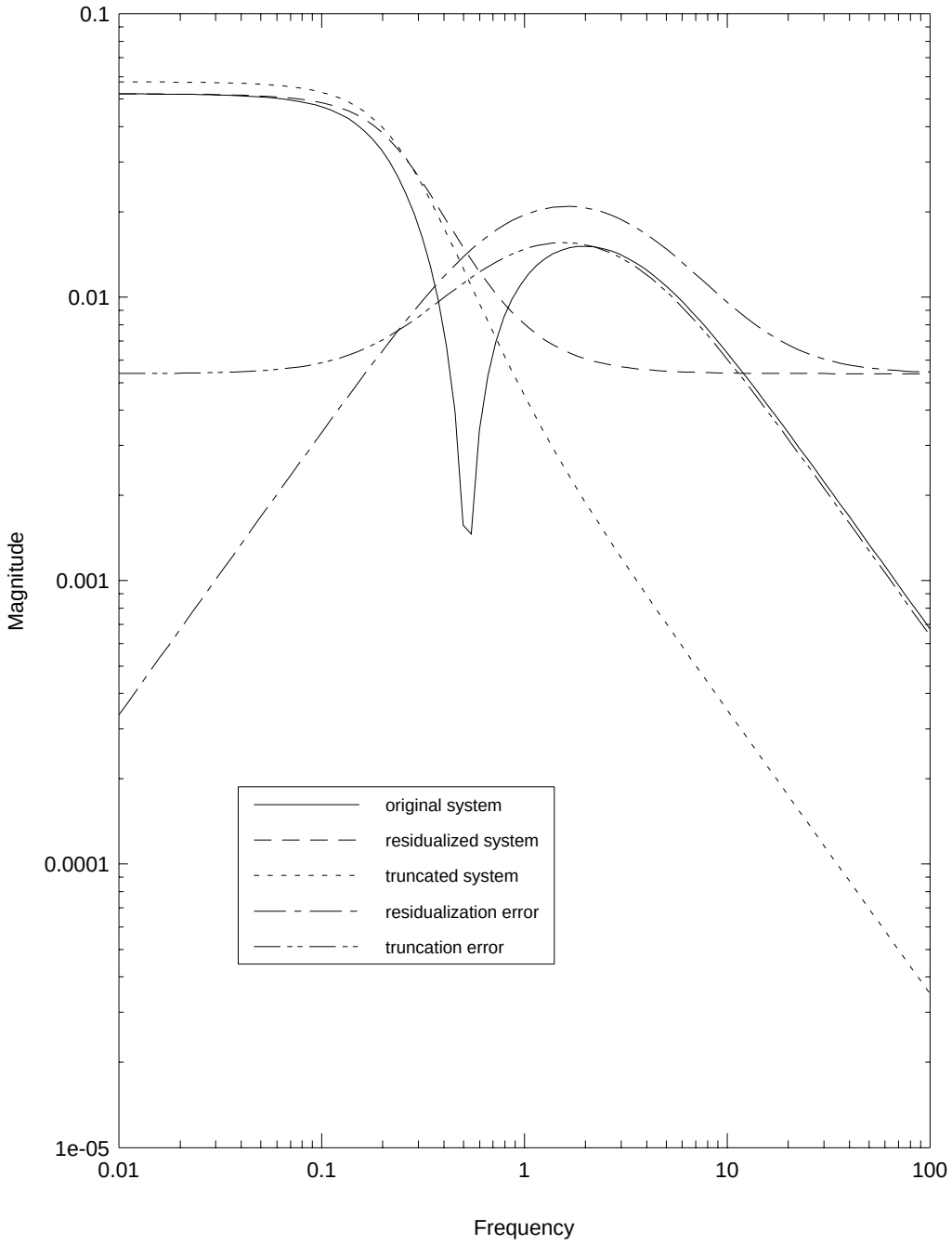
where A_{11} is of dimension `ord` $\times$ `ord`. The truncated output Dynamic System, `sysout` is simply,

$$\texttt{sysin} = \left[\begin{array}{c|c} A_{11} & B_1 \\ \hline C_1 & D \end{array}\right].$$

Example

This example is identical to that described for `sresidualize` and compares the two methods of model reduction.

```
# Create a five state system for reduction.

a = daug(-0.891334,[-1.20857,0.799042;-0.799042,-1.20857],...
             -4.74685,-21.3013)
b = [0.0262569;-0.189601;-0.113729;0.211465;-0.538239]
c = [0.120725,-0.336942,0.397198,-0.700524,-1.02235]
d = 0
sys1 = system(a,b,c,d)

# Reduce to a 3 state system by residualization
# and truncation.

sysout1 = sresidualize(sys1,3)
sysout2 = truncate(sys1,3)

fHz = logspace(0.01,100,100)
sys1g = freq(sys1,fHz)
sysout1g = freq(sysout1,fHz)
sysout2g = freq(sysout2,fHz)
residerror = sys1g - sysout1g
truncerror = sys1g - sysout2g

gph1 = ctrlplot([sys1g,sysout1g,sysout2g,residerror,...
                           truncerror],{logmagplot});
gph1 = plot(gph1,{!grid,legend=["original system";...
         "residualized system";"truncated system";...
         "residualization error";"truncation error"]})?
```


0.1
0.01
0.001
0.0001
1e-05
Magnitude
0.01
0.1
1
10
100
Frequency
original system
residualized system
truncated system
residualization error
truncation error

See also

`rifd`, `simtransform`, `sresidualize`, `orderstate` `modalstate`.

6.2 Xμ Subroutines and Utilities

Several subroutines may also be of interest to the user. These subroutines typically perform self contained parts of a calculation. They may be of interest to those developing new robust control algorithms or wishing to study the calculation details of the algorithms given here. Beware of the fact that these subroutines may not contain error checking.

These subroutines are included in alphabetical order and are cross-referenced by their calling functions in the following list.

`hinfsyn` *subroutines*

`mu` *subroutines*

blkbal

Syntax

```
d = blkbal(M)
```

Description

Balances a square matrix assuming only scalar blocks. The Osborne method (growth rate: n^2) is used for large systems and the Perron method (growth rate: n^3) for smaller systems. The Perron method will exactly calculate mu for positive matrices.

hinfcalc

Syntax

```
[X,Y,f,h,Ric_fail,HX,HY,HXmin,HYmin] = ...
                              hinfcalc(p,nmeas,ncon,g,epr,{keywords})
```

Parameter List

Inputs:	p	Generalized interconnection structure (DYNAMIC SYSTEM)
	nmeas	measurement vector dimension.
	ncon	control vector dimension.
	g	H_∞ norm of suboptimal controller to be calculated. Referred to in the literature as gamma.
	epr	Tolerance for determining when the Hamiltonian eigenvalues lie on the $j\omega$-axis.
Keywords:	schur_solution	real Schur decomposition for Riccati solution (default)
	eig_solution	eigendecomposition for Riccati solution.
Outputs:	X	Riccati solution
	Y	Riccati solution
	Ric_fail	status of solution: 0 Solution found. 1 $j\omega$ axis eigenvalues in Hamiltonian 2 Unequal number of +ve & -ve eigenvalues in Hamiltonian. This represents a numerical failure in the eigenvalue ordering. 3 Both of the above errors detected.
	f	Intermediate calculation for scaling and normalization
	h	Intermediate calculation for scaling and normalization
	HX	X Hamiltonian
	HY	Y Hamiltonian
	HXmin	Minimum absolute value of the real part of the X Hamiltonian eigenvalues. In other words, how close to the $j\omega$ axis the eigenvalues lie.
	HYmin	Minimum absolute value of the real part of the Y Hamiltonian eigenvalues.

Description

Form and solve the Riccati equations for the H_∞ control problem. X and Y are the resulting Riccati solutions.

THIS FUNCTION IS INTENDED ONLY AS A SUBROUTINE CALLED BY THE HINFSYN FUNCTION.

*** NO ERROR CHECKING ***

powermu

Syntax

```
[lbnd,delta,errstat] = powermu(M,blk,rp,cp)
```

Description

Lower bound power algorithm based on the work of Andy Packard. The vector naming roughly corresponds to that in his thesis.

*** NO ERROR CHECKING ***

riccati_eig

Syntax

```
[x1,x2,stat,Heig_min] = riccati_eig(H,epp)
```

Parameter List

Inputs:	H	Hamiltonian matrix.
	epp	Tolerance for detecting proximity of eigenvalues to the $j\omega$ axis.
Outputs:	x1,x2	Basis vectors for stable subspace. See description below.
	stat	Status flag. 0 Stable subspace calculated. 1 Failure to decompose into stable and unstable subspaces.
	Heig_min	Minimum absolute value of the real part of the eigenvalues of H.

Description

Solve the algebraic Riccati equation,

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

by an eigenvalue decomposition method. The Hamiltonian, H, contains the Riccati equation variables in the matrix,

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

If H has no $j\omega$ axis eigenvalues then there is an n dimensional ($n = \dim(A)$) stable subspace of H. The vector, $[x_1\ x_2]$ spans that stable subspace and, if x_1 is invertible, the

variable, $X = x_2 x_1^{-1}$, is the stabilizing solution to the Riccati equation.

If H has $j\omega$ axis eigenvalues then no stabilizing solution exists and the function returns a failure status. If any eigenvalue of H is within epp of the $j\omega$ axis it is considered to lie on the $j\omega$ axis and no solution is found. This may be due to numerical problems in finding the eigenvalues of poorly conditioned problems even when a stabilizing solution exists.

See Also

Riccati, riccati_schur

riccati_schur

Syntax

```
[x1,x2,stat,Heig_min] = riccati_schur(H,epp)
```

Parameter List

Inputs:	H	Hamiltonian matrix.
	epp	Tolerance for detecting proximity of eigenvalues to the $j\omega$ axis.
Outputs:	x1,x2	Basis vectors for stable subspace. See description below.
	stat	Status flag. 0 Stable subspace calculated. 1 Failure to decompose into stable and unstable subspaces.
	Heig_min	Minimum absolute value of the real part of the eigenvalues of H.

Description

Solve the algebraic Riccati equation,

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

by a real Schur decomposition method. The Hamiltonian, H, contains the Riccati equation variables in the matrix,

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

If H has no $j\omega$ axis eigenvalues then there is an n dimensional ($n = \dim(A)$) stable subspace of H. The vector, $[x_1\ x_2]$ spans that stable subspace and, if x_1 is invertible, the

variable, $X = x_2 x_1^{-1}$, is the stabilizing solution to the Riccati equation.

If H has $j\omega$ axis eigenvalues then no stabilizing solution exists and the function returns a failure status. If any eigenvalue of H is within epp of the $j\omega$ axis it is considered to lie on the $j\omega$ axis and no solution is found. This may be due to numerical problems in finding the eigenvalues of poorly conditioned problems even when a stabilizing solution exists.

See Also

Riccati, riccati_eig

Appendices

A Translation Between MATLAB μ-Tools and Xμ

This appendix outlines the functional equivalences between the MATLAB μ-Tools and Xmath Xμ. The objective is to provide a smooth transition for users moving from μ-Tools to Xμ. We will assume that the reader is familiar with MATLAB μ-Tools and the general operation of Xmath. The intent is that the overall functional capabilities are the same under either system and a prospective robust control designer chooses between MATLAB, Xmath, or future matrix languages, on other issues (cost, support, software compatibility, etc.).

There are enough similarities between μ-Tools and Xμ that one can move from one to the other without a great deal of additional learning. Most of the differences are syntactic. There is not always a direct function for function match between the two systems.

The major differences are:

- The built-in data structures available with Xmath.
- The discrete-time system is available as an Xmath data object.
- Slightly different function names.
- The `mu` and `musynfit` functions handle the scaling and perturbation matrices in matrix, rather than coded vector, form.
- The implementation of the *D-K* iteration is slightly different.

The functionally equivalent commands will be listed, in each sub-section, for convenience. A more detailed discussion is given to illustrate the more subtle differences in the `mu` and D-K iteration aspects. More importantly, the Himat demo is available in both μ-Tools and Xμ. For a fast start on moving between platforms, study these demos side by side.

A.1 Data Objects

The underlying data structure in MATLAB is the complex valued matrix, whereas in Xmath there various other objects available: polynomials, transfer functions, parameter dependent matrices, and dynamic systems.

SYSTEM/DYNAMIC SYSTEM Functions

The μ-Tools functions for creating and manipulating the SYSTEMmatrix have no Xμ equivalent as the underlying Xmath operators can directly handle the dynamic system and transfer function data objects.

Also residing within the Xmath state-space object is the initial state. This is used primarily for time response calculations. It is debatable whether or not the initial state is an intrinsic attribute of the system as one frequently changes it for simulation. It does have the advantage of reducing the time response calculation to a simple multiplication and it can easily be changed without accessing all of the other system variables.

The following table illustrates the equivalent functions, or indicates the data objects which provide the same functionality.

Description	μ-Tools Function	Xmath/Xμ equivalent
form system	`pck`	`system`
decompose system	`unpck`	`abcd`
form system	`nd2sys`	transfer function data objects
form system	`zp2sys`	transfer function data objects
decompose system	`sys2pss`	dynamic system data objects
form system	`pss2sys`	dynamic system data objects
random system	`sysrand`	`randsys`
generate filters	`mfilter`	`butterworth`, `chebyshev`
fit transfer functions	`drawmag`	`fitsys`, `tfid`

The μ-Tools `drawmag` function provides an interactive graphical interface. There is currently no equivalent in Xμ. The equivalent underlying data fitting algorithm can be found in the Xμ function `fitsys`.

VARYING/PDM **Functions**

The equivalent functions for construction and manipulation of PDM data objects are shown below.

Description	μ-Tools Function	Xmath/Xμ equivalent
form VARYING	`vpck`	`pdm`
break up VARYING	`vunpck`	`makematrix`
get ivs.	`getiv`	`domain`
join VARYING	`tackon`	`concatseg`, `insertseg`, `mergeseg`
sort by iv.	`sortiv`	`mergeseg`
select by iv. value	`xtract`	`extractseg`, `indexlist`, `find`
select by index	`xtracti`	indexing by `pdm(i)`
scale iv.	`scliv`	`domain`, `pdm`
compare ivs.	`indvcmp`	`check`
random VARYING	`varyrand`	`randpdm`

As an aside, note that the `sort` function in Xmath sorts each column of each matrix in a PDM, rather than sorting the domain.

Subblocks: selecting input & outputs

In μ-Tools the function `sel` selects rows and columns from a VARYING matrix or inputs and outputs from a SYSTEM matrix. In Xmath these can be obtained by specifying row and column indexes. More flexibility of selecting parts of a PDM can be obtained by using the `indexlist` function.

Augmentation

Augmentation is the building of matrices from component pieces. The μ-Tools commands which perform these functions are given in the table below. For PDMs or DYNAMIC SYSTEMs in Xmath these operations are generally performed identically to the equivalent matrix operation.

Description	μ-Tools Function	Xmath/Xμ equivalent
vertical augmentation	`abv`	`;`
horizontal augmentation	`sbs`	`,`
diagonal augmentation	`daug`	`daug`

Algebraic Operations

Similarly, algebraic operations on SYSTEM or VARYING matrices in μ-Tools require a dedicated function. In the Xmath case the usual matrix operation suffices.

Description	μ-Tools Function	Xmath/Xμ equivalent
addition	`madd`	`+`
subtraction	`msub`	`-`
multiplication	`mmult`	`*`
system scaling	`mscl`	`*`
system scaling	`sclin`	`*`
system scaling	`sclout`	`*`
system inverse	`minv`	`inv`
transpose	`transp`	`'`
conjugate transpose	`cjt`	`*'`

Note that the transpose and conjugate transpose operators are defined differently for MATLAB and Xmath.

SYSTEM/DYNAMIC SYSTEM Functions

The following functions perform useful manipulations to, or information about, the state of a SYSTEM or DYNAMIC SYSTEM.

Description	μ-Tools Function	Xmath/Xμ equivalent
calculate poles	`spoles`	`poles`
calculate zeros	`szeros`	`zeros`
display poles	`rifd`	`rifd`
state similarity transform	`statecc`	`simtransform`
reorder state	`reordsys`	`orderstate`
transform to modal format	`strans`	`modalstate`
zero order hold equivalent	`samhld`	`discretize`
Prewarped bilinear transform	`tustin`	`discretize`

VARYING/PDM Functions

A large number of VARYING matrix operations have been written in μ-Tools. These are not required in the Xmath version as most of the functions operate on PDMs as well as matrices. The following table lists these functions for μ-Tools and Xmath. These are only approximate functional equivalents — the more complicated functions differ in some important respects.

Description	μ-Tools Function	Xmath/Xμ equivalent
peak norm	`pkvnorm`	`norm`, `max`
absolute value	`vabs`	`abs`
diagonal matrix	`vdiag`	`diagonal`
round downwards	`vfloor`	`round`
round upwards	`vceil`	`round`
imaginary part	`vimag`	`imag`
real part	`vreal`	`real`
complex conjugate	`vconj`	`conj`
norm	`vnorm`	`norm`
determinant	`vdet`	`det`
eigenvalues	`veig`	`eig`
inverse	`vinv` or `minv`	`inv`
left division	`vldiv`	`\`
right division	`vrdiv`	`/`
pseudo-inverse	`vpinv`	`pinv`
spectral radius	`vrho`	`spectrad`
singular values	`vsvd`	`svd`
condition number	`vrcond`	`rcond` or `condition`
schur decomposition	`vschur`	`schur`
matrix exponential	`vexpm`	`expm`
interpolation	`vinterp`	`interp` or `interpolate`
decimation	`vdcmate`	`pdm(vector)`
FFT	`vfft`	`fft`
inverse FFT	`vifft`	`ifft`
spectral analysis	`vspectrum`	`spectrum`

Beware of syntactical differences here. One obvious example is the order in which the eigenvalues and eigenvectors are returned from `veig` and `eig`. Note also the `vrcond` returns the inverse of the condition number whereas `condition` returns the condition number.

The μ-Tools function `vebe` performs element-by-element function operations on a VARYING matrix. This has no counterpart in Xμ as the basic functions which operate on each element of a matrix (e.g. sin, cos, abs) are also defined on PDMs in Xmath. For the same reason, the μ-Tools function `veval` has no Xμ counterpart: the Xmath `execute` command can be used to the same effect.

Miscellaneous Utilities

Several utilities are provided in μ-Tools. These are subroutines used by other μ-Tools functions which may be of more general use.

Description	μ-Tools function	Xmath/Xμ equivalent
complex random number	`crand`	`randpdm`
fit system to data	`fitsys`	`fitsys`
Eigenvalue based Riccati solution	`ric_eig`	`riccati_eig`
Schur based Riccati solution	`ric_schr`	`riccati_schur`

A.2 Matrix Information, Display and Plotting

Xmath provides matrix/data object size information via a variable window. There is no MATLAB equivalent for this functionality. Command window information can be obtained via the following functions.

Description	μ-Tools function	Xmath/Xμ equivalent
matrix information	`minfo`	`check`, `size`, `is`
list workspace	`whos`	`who`

Plotting of VARYING matrices is provided by the MATLAB μ-Tools function `vplot`. As PDMs are a native data object in Xmath, the standard Xmath `plot` function will correctly plot a PDM. Multiple calls will overlay the data, even if the domains differ. The Xμ function `ctrlplot` is provided for more control specific plots: Bode, Nyquist, Nichols, log magnitude, etc..

Both MATLAB and Xmath allow interactive manipulation of the graphical data and storage and retrieval of plots from the workspace or the underlying file system.

A.3 System Response Functions

Creating Time Domain Signals

The Xmath PDM data object allows the creation of time domain signals via standard and operators.

Description	μ-Tools function	Xmath/Xμ equivalent
cosine waveform	`cos_tr`	`cos`
sine waveform	`sin_tr`	`sin`
stair-step waveform	`step_tr`	`gstep`
general waveform	`siggen`	standard functions & operators

Time Responses

Xmath calculates time responses with the `*` operation. However, the Xμ function `trsp` provides significantly more functionality in the continuous time case. The equivalences are summarized below.

Description	μ-Tools function	Xmath/Xμ equivalent
continuous time response	`trsp`	`trsp`, `*`, `deftimerange`
discrete time response	`dtrsp`	`*`
sampled data response	`sdtrsp`	`sdtrsp`

Frequency Responses

The relevant functions are shown below.

Description	μ-Tools function	Xmath/Xμ equivalent
frequency response	`frsp`	`freq`
logarithmic vector	`logspace`	`logspace`

The default frequency units in μ-Tools are radians/second whereas those in Xmath are Hertz. This applies generically to all commands where the user specifies frequency

information.

A.4 System Interconnection

Simple interconnection has already been outlined in the augmentation section above. The more complicated interconnection functions are almost identical in μ-Tools and Xμ. The only real difference is the calling syntax of `sysic`.

Description	μ-Tools function	Xmath/Xμ equivalent
Redheffer (star) product	`starp`	`starp`
system interconnection	`sysic`	`sysic`

A.5 Model Reduction

The model reduction functions in Xμ are:

Description	μ-Tools function	Xmath/Xμ equivalent
residualization	`sresid`	`sresidualize`
state truncation	`strunc`	`truncate`
balanced realization	`sysbal`	`balance`
Hankel norm reduction	`hankmr`	`ophank`

The following frequency weighted model reduction functions have no equivalent in the current version of Xμ. It is intended to introduce this functionality in the next version. The functions are: `sdecomp`, `sfrwtbal`, `sfrwtbld`, `sncfbal` and `srelbal`.

A.6 H_2 and H_∞ Analysis and Synthesis

The synthesis algorithms are identical and the syntax are similar.

Description	μ-Tools function	Xmath/Xμ equivalent
H_2 norm calculation	`h2norm`	`h2norm`
H_∞ norm calculation	`hinfnorm`	`hinfnorm`
H_2 controller synthesis	`h2syn`	`h2syn`
H_∞ controller synthesis	`hinfsyn`	`hinfsyn`

The major syntactical difference is that the Xμ functions do not return the closed loop system. This is easily calculated by a subsequent call to `starp`. The reason for this is that the *D-K* iteration changes typically involve a differently weighted closed loop system in subsequent operations. Having a separate calculation of the closed loop reduces the potential for confusion.

The μ-Tools function `hinffi` has no Xmath/Xμ equivalent. This function calculates the full information H_∞ controller. `hinfsyn` solves the more general problem and is of more practical use in controller applications.

A.7 Structured Singular Value (μ) Analysis and Synthesis

The issue of whether or not the frequency domain variable is radian/second or Hertz arises here. Although this makes no difference to the calculation of μ is affects how the D scales and Δ perturbations are interpolated. μ-Tools assumes that the frequency scale is radians/second. In Xμ the default assumption is Hertz. The reason for this is that Hertz is the default output of the frequency response calculation `freq`. In all cases where it makes a difference, the user can specify the keyword {`!Hertz`} to switch the meaning of the domain.

Calculation of μ

There is a difference in the returned variable format for the Xμ function `mu`. The MATLAB function returns the D-scale and perturbations in coded vector form. The Xμ `mu` function returns both the D scale and its inverse in matrix form. The relevant functions are summarized below.

Description	μ-Tools function	Xmath/Xμ equivalent
structured singular value	`mu`	`mu`
D scale decoding	`unwrapd`	not required
perturbation decoding	`unwrapp`	not required
block norm calculations	`blknorm`	`blknorm`
rational perturbation	`dypert`	`mkpert`
random perturbations	`randel`	`randpert`

In the Xμ `mu` function, only the default options of the μ-Tools `mu` calculations are available. In other words, a power iteration, with several random restarts, is used for the lower bound. The upper bound calculation uses an Osborne balancing method and enhances this with the Perron vector method for problems with less than 10 blocks. These methods have been found to be appropriate for the vast majority of practically motivated problems.

New algorithms for these calculations are currently under development. The most significant enhancement is the ability to calculate μ with respect to structures which include real valued blocks. Because of the development effort in this direction, a wide range of calculation options were not provided for the current Xμ `mu` function.

The scalar $\times$ identity block structure is not currently supported in the Xμ `mu` function. It will be included in the revised version discussed above.

The D-K Iteration

There is a significant difference in the way that Xμ handles the D-K iteration. The D scales are not incrementally factored into the previous iteration D-scales. Consider the initial design interconnection structure to be `ic`. An H_∞ design will produce the first controller: `k1`, using a function call like the following.

```
k1 = hinfsyn(ic,nmeas,ncon,gamma)
```

The closed loop system, obtained via

```
g1 = starp(ic,k1)
```

is then analyzed with `mu`. The typical function call is:

```
g1g = freq(g1,omega)
[mubnds,Dmagdata] = mu(g1g,blk)
```

From this, frequency domain D-scales are obtained, and a rational approximation is obtained via `musynfit`.

```
[Dsys,Dinvsys] = musynfit(Dmagdata,blk,nmeas,nctrls,weight,g1g)
```

A difference between the Xμ and μ-Tools implementations of `musynfit` is that, in the Xμ case, the inverse scaling system, `Dinvsys`, is generated by `musynfit`. Notice also that the previous D-scale fit is not passed as an argument.

The new, D-scale weighted interconnection structure, `ic2`, is formed by,

```
ic2 = Dsys*ic*Dinvsys
```

A second design, `k2`, is now obtained with `hinfsyn`. Up to this point, the μ-Tools and Xμ procedures are essentially the same.

The major difference is that the appropriate closed loop system is now formed with `ic`, not `ic2`. In other words,

```
g2 = starp(ic,k2)
```

Note that `ic` and `ic2` differ only by the D-scales and therefore μ(`g2`) would be the same whether `ic` or `ic2` were used for the closed loop system.

Now `mu` is used to analyze the frequency response of`g2`, giving rise to a new set of D-scales. Again, `musynfit` is used to fit state-space systems to these D-scales, giving rise to `D2sys` and `D2invsys`. Now the next interconnection structure, `ic3`, is formed by using these D-scales with the original interconnection. In command line form:

```
ic3 = D2sys*ic*D2invsys
```

Note that at each step, `ic`, is used to calculate the closed loop system, and also used to calculate the next design interconnection structure.

The advantage of this is that in order to restart, or reproduce, an iteration, one need only save the previous controller. The μ-Tools approach requires saving the rational approximation to the previous D-scales. The controller is a more applicable data object to save and the saving of the previous D-scales depends on the quality of the rational approximation. The disadvantage is that the upper bound in the next μ calculation takes slightly longer as the closed-loop system does not have the numerical benefit of the effects of the previous D-scales. The speed difference is likely to be insignificant in practical problems.

Fitting D Scales

Rational transfer function fitting of magnitude data is required for the D-K iteration and has been mentioned above. The relevant functions are summarized here.

Description	μ-Tools function	Xmath/Xμ equivalent
fit D scale data	`musynfit`	`musynfit`
phase calculation	`genphase`	`mkphase`
transfer function fitting	`fitsys`	`fitsys`, `tfid`

Note that the μ-Tools version of `musynfit` does not use the previous D scale magnitude data. This has implications in the D-K iteration and is discussed in more detail above.

The alternative μ-Tools linear programming approach for D scale fitting, `musynflp`, is not supported in Xμ. Similarly, the underlying linear programming approach to transfer function fitting (μ-Tools functions `fitmaglp`, `magdata`) is not available in Xμ. The batch functionality of the μ-Tools function, `muftbtch`, is available in the Xμ version of `musynfit`.

Technical Support and Professional Services

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

- **Support**—Online technical support resources at `ni.com/support` include the following:
  - **Self-Help Resources**—For immediate answers and solutions, visit the award-winning National Instruments Web site for software drivers and updates, a searchable KnowledgeBase, product manuals, step-by-step troubleshooting wizards, thousands of example programs, tutorials, application notes, instrument drivers, and so on.
  - **Free Technical Support**—All registered users receive free Basic Service, which includes access to hundreds of Application Engineers worldwide in the NI Developer Exchange at `ni.com/exchange`. National Instruments Application Engineers make sure every question receives an answer.
- **Training and Certification**—Visit `ni.com/training` for self-paced training, eLearning virtual classrooms, interactive CDs, and Certification program information. You also can register for instructor-led, hands-on courses at locations around the world.
- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, NI Alliance Program members can help. To learn more, call your local NI office or visit `ni.com/alliance`.

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