



Identification and robust control methods using ellipsoidal parametric uncertainty descriptions
by Steven Michael Hietpas

A thesis submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in
Electrical Engineering

Montana State University

© Copyright by Steven Michael Hietpas (1994)

Abstract:

This dissertation addresses the areas of identification and robust control of uncertain systems. The first portion of this dissertation concerns system identification. The focus of system identification is grounded in the signal analysis technique developed by G. R. B. Prony in 1795. This approach uses two separate least-squares solutions, with the first least-squares solution resulting in the eigenvalues of the output signal and the second yielding the output residues. Previous system identification methods based on Prony signal analysis apply to systems utilizing a continuous-time input. This research develops a method that allows for a sampled-and-held version of the previously used continuous-time input. The new method results in a more simplified algorithm due to the nature of sampled-and-held input signals. Various examples are provided to show the characteristics of the method. The second portion of this dissertation concerns control of systems which are modeled through system identification techniques that provide models with a parametric uncertainty set. The robust control strategy is viewed from dynamic game theory such that a saddle point solution is desired. The game theory method employed has two players: the first player, the feedback control signal, minimizes an energy cost function; and the second player: the set of uncertain parameters, maximizes an energy cost function. It is shown that the optimal control signal necessarily is derived from Linear Quadratic Regulation (LQR) theory. The maximization is performed on the final cost function from the LQR solution and is constrained by an ellipsoidal parameter bound and an appropriate Lyapunov equation. Iterative algorithms for both continuous-time and discrete-time systems are developed. A sufficient condition for the existence of the maximization portion of the discrete-time algorithm is given. Examples show the fixed-point solution of the algorithm is a saddle point. A closed-loop discrete controlled servo-mechanism problem uses the Optimal Volume Ellipsoid (OVE) algorithm of Man-Fung Cheung (1990) for identification of a model and an ellipsoid uncertainty region. The discrete minimax algorithm is applied to determine the optimal state-feedback gain. The servo example is stable and shows the worst-case tracking for the noisy, uncertain system and is compared to the known system.

IDENTIFICATION AND ROBUST CONTROL METHODS USING
ELLIPSOIDAL PARAMETRIC UNCERTAINTY DESCRIPTIONS

by

Steven Michael Hietpas

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Doctor of Philosophy

in

Electrical Engineering

MONTANA STATE UNIVERSITY
Bozeman, Montana

July 1994

APPROVAL

of a thesis submitted by

Steven Michael Hietpas

This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the College of Graduate Studies.

7/14/94
Date

Donald A. Pierre
Donald A. Pierre
Chairperson, Graduate Committee

Approved for the Major Department

7/14/94
Date

Victor Gerez
Victor Gerez
Head, Electrical Engineering

Approved for the College of Graduate Studies

7/23/94
Date

Robert Brown
Robert Brown
Graduate Dean

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment for a doctoral degree at Montana State University, I agree that the Library shall make it available to borrowers under rules of the Library. I further agree that copying of this thesis is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U. S. Copyright Law. Requests for extensive copying or reproduction of this thesis should be referred to University Microfilms International, 300 North Zeeb Road, Ann Arbor, Michigan 48106, to whom I have granted "the exclusive right to reproduce and distribute my dissertation for sale in and from microform or electronic format, along with the right to reproduce and distribute my abstract in any format in whole or in part."

Signature

Steve Pappas

Date

7-14-94

ACKNOWLEDGEMENTS

This research was made possible through financial support from the National Science Foundation, the Montana Electric Power Research Association and the Engineering Experiment Station of Montana State University. I express my gratitude to Professor Donald Pierre, my principal advisor, for his continuous support, and certainly for his enduring patience. Dr. Pierre has given me a great example of a professional educator and researcher. Through the efforts of Professor Roy Johnson in his continual quest for upgrading existing computer equipment to the best possible – I had it very nice. Dr. Ming Lau of Sandia Laboratories, New Mexico, played a strong part in the robust control portion of this research, and offered much of his time to the review of earlier work and carefully steered me in the correct direction at crucial times. To the rest of my committee, Dr. James Smith, Dr. Hashem Nehrir, Dr. Curtis Vogel and Dr. Richard Swanson, thank you for the time you spent listening to my concerns and offering advice where needed. I extend my appreciation to all of my fellow graduate students, Cesar Amicarella, Fereshteh Fatehi, Don Hammerstrom, Ed Ray, Tia Sharpe and Stott Woods, who have always shown confidence in me when things were not going as well as I desired. A special thanks to Don Hammerstrom, a great engineer and a good friend, for all the helpful discussions, assistance in mastering L^AT_EX, and for all the exciting racquet ball games. My pastors, Blake Updike and Dave Green, were always available to listen and offer their wisdom and advice when I needed it most. Of course, without a doubt, I thank my parents, Jerry and JoAnne Hietpas, and Jack and Mary Gay Wells for their encouragement. I am fortunate to have wonderful in-laws who also encouraged me throughout this dissertation; thanks

Lance and Marllys Johnson and Collin. I also have the best brother, Tom and sisters, Elise and Tara, who have always been there for me, thanks. My special buddies, Taylor and Aaron, you have always been and always will be an inspiration and joy to me. In addition to these wonderful boys, on June 16, 1994, I was blessed with a beautiful girl, Megan Lynn. Lastly, and most importantly, to my wife Tara, your unfailing love, commitment and confidence in me has kept me going. I owe much of this work to you; thank you so much for being at my side through everything while taking care of the home front. At the beginning of this dissertation, Tara gave me the following quote, Philippians 4:13, New International Version of the Holy Bible; and I close, acknowledging that Christ Jesus is truly the sustainer of all things on this earth.

I can do everything through Him who gives me strength.

TABLE OF CONTENTS

	Page
LIST OF TABLES	ix
LIST OF FIGURES	x
ABSTRACT	xi
1. INTRODUCTION	1
2. DISCRETE PRONY IDENTIFICATION	11
Review of System Identification Methods	11
Autoregressive Models (ARX)	13
State-Space Models	15
Prony Signal Identification for Discrete Control Systems	24
Introduction: Discrete Prony	24
System Characterization	25
Prony Based Analysis	29
Examples	36
Discussion of Results	50
3. SET-MEMBERSHIP APPROACH TO CONTROL DESIGN	54
Ellipsoid Set Estimation	56
Disturbance/Noise Induced Ellipsoid	56
Non-parametric Uncertainty Induced Ellipsoid	58
Linear Quadratic Regulator	62
Continuous-Time LQR	62
Discrete-Time LQR	64
Minimaximization Problems	65
Example: Lau	66
Example: Mills	67
Proposed Minimaximization Problems	68
Discussion of Results	71

TABLE OF CONTENTS — Continued

	Page
4. MINIMAXIMIZATION FOR SINGLE-INPUT SINGLE-OUTPUT SYSTEMS	73
Continuous-Time System Problem Statement	73
Some Useful Results from LQR Control Theory	75
Iterative Solution Approaches	76
Examples	81
Discrete-Time System Problem Statement	87
Iterative Solution Approach	89
On The Nature Of The Solution Of Algorithm 3	90
Set-membership Control Problem	94
Discussion of Results	100
5. CONCLUSION	102
6. FUTURE RESEARCH	106
REFERENCES CITED	109
APPENDICES	117
APPENDIX A — Identification Code	118
runcomp.m	119
id_plt.m	130
diff_model.m	134
diff_modes.m	136
diff_nn.m	140
diff_noise.m	141
pr1.m	145
pr1b.m	153
pr2.m	156
pr2b.m	163
pr3.m	171
prony_in.m	179
u_prony.m	181
get_prony_in.m	183
in_exmpl.m	186
rm_mode.m	187

TABLE OF CONTENTS — Continued

	Page
xdisp_mode.m	189
make_obsvble.m	190
obsv.m	194
sys2obsv_can.m	195
moon1.m	199
moon2.m	203
moon_in.m	206
u_moonen.m	208
get_moon_in.m	209
u_modes.m	211
sysm6.m	212
discr.m	215
fftst.m	217
dir_stamp.m	219
file_stamp.m	220
time.m	222
Chapter 2, Example 4: Equations and Code	223
sys_infty.m	227
rk.m	237
APPENDIX B – Robust Control Code	239
mmJ.m	240
minJ1.m	245
maxJ1.m	246
gradJ1.m	247
maxJ2.m	249
gradJ2.m	250
ricc.m	252
servo.m	254
mmJd.m	263
maxJd.m	271
ove.m	274

LIST OF TABLES

Table	Page
1 Actual system model data for Examples 1-3.	38
2 Input drive data for Examples 1-2.	38
3 Data from Prony algorithm for Example 1.	39
4 Input drive data for <i>discrete Prony</i> of Example 3.	44
5 Example 3 error data for noise applied at $d(t)$	47
6 Final model for Example 4.	49
7 Interval bounds on parameters of Example 1 using OVE algorithm. .	98
8 Machine data for Example 4 of Chapter 2.	224

LIST OF FIGURES

Figure,	Page
1 System and actual model in parallel form.	26
2 The linear mass/spring diagram for Examples 1-3.	37
3 Probing input $u(t)$ and its Fourier transform equivalent for Examples 1-2.	38
4 Error in dB for each additional mode set for Example 1.	40
5 The output for both actual, $y(t)$ and model, $\hat{y}(t)$ for Example 1.	40
6 Error in dB for each additional mode set for Example 2.	41
7 The output for both actual, $y(t)$ and model, $\hat{y}(t)$ for Example 2 with additive output noise.	42
8 The output for both actual, $y(t)$ and model, $\hat{y}(t)$ for Example 2 with additive output noise removed.	42
9 The poles of the s -domain for the actual system and the model for Example 2.	43
10 Probing input $u(t)$ and its Fourier transform equivalent for Example 3 using the <i>discrete Prony</i> algorithm.	44
11 Probing input $u(t)$ and its Fourier transform equivalent for Example 3 using the <i>Moonen</i> algorithm.	45
12 The singular values of the Hankel matrix for Example 3 using the <i>Moonen</i> algorithm, no noise case.	46
13 The singular values of the Hankel matrix for Example 3 using the <i>Moonen</i> algorithm with 50dB output SNR.	47
14 One-machine/infinite-bus power system diagram.	48
15 Residue ordering for example 4.	49
16 Detailed diagram of control scheme for Example 4.	50
17 Transient simulation of example 4: Open loop vs. LQG.	51
18 Set-membership adaptive control scheme diagram.	55
19 Illustration of saddle point on the boundary of Ω for Example 1.	84
20 Illustration in two dimension of saddle point condition for Example 1.	85
21 Illustration of saddle point on the boundary of Ω for Example 2.	87
22 LQR costs for plants within interior of uncertainty region using $K^* = 1.4692$; Example 2.	88
23 Set-membership adaptive control detailed block diagram.	97
24 Servomechanism problem for standard LQG and set-membership control.	99
25 Steady-state vector diagram of bus voltages.	224

ABSTRACT

This dissertation addresses the areas of identification and robust control of uncertain systems. The first portion of this dissertation concerns system identification. The focus of system identification is grounded in the signal analysis technique developed by G. R. B. Prony in 1795. This approach uses two separate least-squares solutions, with the first least-squares solution resulting in the eigenvalues of the output signal and the second yielding the output residues. Previous system identification methods based on Prony signal analysis apply to systems utilizing a continuous-time input. This research develops a method that allows for a sampled-and-held version of the previously used continuous-time input. The new method results in a more simplified algorithm due to the nature of sampled-and-held input signals. Various examples are provided to show the characteristics of the method. The second portion of this dissertation concerns control of systems which are modeled through system identification techniques that provide models with a parametric uncertainty set. The robust control strategy is viewed from dynamic game theory such that a saddle point solution is desired. The game theory method employed has two players: the first player, the feedback control signal, minimizes an energy cost function; and the second player: the set of uncertain parameters, maximizes an energy cost function. It is shown that the optimal control signal necessarily is derived from Linear Quadratic Regulation (LQR) theory. The maximization is performed on the *final cost function* from the LQR solution and is constrained by an ellipsoidal parameter bound and an appropriate Lyapunov equation. Iterative algorithms for both continuous-time and discrete-time systems are developed. A sufficient condition for the existence of the maximization portion of the discrete-time algorithm is given. Examples show the fixed-point solution of the algorithm is a saddle point. A closed-loop discrete controlled servo-mechanism problem uses the Optimal Volume Ellipsoid (OVE) algorithm of Man-Fung Cheung (1990) for identification of a model and an ellipsoid uncertainty region. The discrete minimax algorithm is applied to determine the optimal state-feedback gain. The servo example is stable and shows the worst-case tracking for the noisy, uncertain system and is compared to the known system.

CHAPTER 1

INTRODUCTION

Very often, a dynamical system is sufficiently complex such that linear or nonlinear sets of equations that adequately describe the system are not available. This is true for many systems, including sophisticated aircraft, chemical processes, large power systems, etc. Under these circumstances it may be necessary to assume a particular parameterized model and apply system identification techniques to arrive at a useful mathematical model of the system to be controlled. The subject of this dissertation addresses system identification and robust control of uncertain systems.

System identification deals with the problem of building mathematical models of dynamical systems based on observed data from the system. A system is an object in which variables of different kinds interact and produce observable signals. Observable signals that emanate from the system are often referred to as outputs. External signals that can be manipulated by the observer and applied to the system are called controlled inputs. Other system input signals are called disturbances, some of which can be directly measured, while others can only be observed through their influence on the output. Systems such as aircraft, space structures, bridges, ships and power grids, to name a few, are considered dynamical, which means that the current outputs depend not only on the current inputs but also on their earlier values. Outputs of dynamical systems whose external stimuli are not observed, e.g., the sound of a human voice generated by the vibration of the vocal chords, are often called *time series*. This term is common in economic applications. Clearly, the list of examples of dynamical systems can be very long and it stretches over many fields of human endeavor.

A major part of the engineering field deals with how to make good designs based on mathematical models. A model describes the relationship among system variables in terms of mathematical expressions like difference or differential equations. Such models are called *mathematical* (or *analytical*) *models*. Mathematical models may be further characterized by a number of adjectives: time continuous or time discrete, lumped or distributed, deterministic or stochastic, linear or nonlinear, single or multi-input/output, etc. When input and output signals from a system that are subjected to data analysis in order to infer a model, the activity is known as *system identification*.

The construction of a model from data involves three basic entities [1]:

- I. The data.
- II. A set of candidate models.
- III. A rule by which candidate models can be assessed using the data.

The data record should be selected to be maximally informative, subject to constraints that may be at hand. Obtaining maximally informative input-output data is affected by which output signals are measured, when and how the signals are measured, and the choice of input signals for external stimuli. The set of candidate models to choose from is certainly the most important step of the identification procedure. It is here where prior knowledge, engineering intuition and insight have to be combined with the formal properties of models. After selection of an appropriate model one is faced with various methods of fitting the model to the data. One approach to the assessment of the model quality is based on how the model performs when it attempts to reproduce the measured data.

The first part of this dissertation is concerned with system identification. The focus of system identification in this research is grounded in the signal analysis tech-

nique developed by G. R. B. Prony [2] in 1795. In this work, Prony proposed a basic signal analysis method to approximate a signal $y(t)$ by a summation of n weighted exponentials, i.e.,

$$y(t) \approx \hat{y}(t) = \sum_{i=1}^n B_i e^{\lambda_i t}$$

for continuous time $t \geq 0$, where $B_i \in \mathbb{C}$. The approach uses two separate least-squares solutions, each of order n , with the first least-squares solution resulting in the eigenvalues of the signal $\{\lambda_i\}$, and the second yielding the weighting terms (or output residues) $\{B_i\}$. The basic Prony method is a signal analysis method rather than a system identification method. Extending this signal analysis method of Prony to system identification was possibly first presented by Poggio, et al., [3] in 1978. The system input in [3] is a summation of two real exponential inputs.

Typically the number of eigenvalues (or poles) n is chosen sufficiently large (greater than the ‘actual’ number of eigenvalues) prior to the formulation of the first least-square problem. The problem of solving the first set of equations is equivalent to a linear prediction (LP) problem. Kumaresan and Tufts [4]-[7] address the effect of noise on the solution to the LP problem in the first least-squares formulation. Their work shows, that from the over-determinization (i.e., large n), the effect of using a truncated singular value decomposition (SVD) essentially increases the signal-to-noise-ratio (SNR) in the data prior to obtaining the solution. The sample variance of the parameter estimates are compared to the Cramér-Rao (CR) bound for the pole damping factors and pole frequencies. A description of the Cramér-Rao bound can be found in [1, pages 183-187] and is briefly described here. Suppose we are interested in estimating a set of parameters θ from a set of data denoted by y^N . Let the estimated set be defined as $\hat{\theta}(y^N)$ and the “true value” of θ be denoted by θ_0 . The quality of the estimator can be assessed by its mean-square error matrix:

$$P = \mathbb{E}[\hat{\theta}(y^N) - \theta_0][\hat{\theta}(y^N) - \theta_0]^T$$

where $\mathbb{E}$ evaluates the mean. We may be interested in selecting estimators that make P small. It is then interesting to note that there is a lower limit to the values of P that can be obtained with the various unbiased estimators. Let the probability density function of y^N be given by $f_y(\theta_0; y^N)$ and assumed to be known. The definition of the *Cramér-Rao inequality* is stated as follows:

Definition 1.1 (Cramér-Rao inequality) *Let $\theta(y^N)$ be an estimator of θ such that $\mathbb{E}\theta(y^N) = \theta_0$ and suppose that y^N may take values in a subset of $\mathbb{R}^N$, whose boundary does not depend on θ_0 . Then*

$$\mathbb{E}[\hat{\theta}(y^N) - \theta_0][\hat{\theta}(y^N) - \theta_0]^T \geq M^{-1}$$

where

$$M = \mathbb{E} \left[\left. \frac{d}{d\theta} \log f_y(\theta; y^N) \right] \left[\frac{d}{d\theta} \log f_y(\theta; y^N) \right]^T \right|_{\theta=\theta_0} = -\mathbb{E} \left. \frac{d^2}{d\theta^2} \log f_y(\theta; y^N) \right|_{\theta=\theta_0}$$

The matrix M is known as the *Fisher information matrix* and evaluation requires the knowledge of θ_0 , so the exact value of M may not be available to the user. More information on the Cramér-Rao bound when dealing with dynamical systems can be found in [1, pages 186-187].

Further analysis of the effect of the truncated SVD and the variance of the parameter estimates is given in [8] and references therein. Although much of the work in [4]-[7] deals with signal analysis aspects, a transfer function identification example is given in [7]. In this example, however, the input is an impulse signal and after the poles of the system are determined using Prony's method, the zeros of the transfer function (roots of the numerator polynomial) are found using Shank's method [9]. Further investigation of Prony's method for signal modeling of noisy data and model order selection can be found in [10]-[12]. Parameter estimation of exponentially damped sinusoids using higher order statistics (HOS) utilizing Prony's method is explored in [13].

The work in [14]- [20] progressively extends the flexibility of using Prony analysis to perform system identification. In [14] and [15] the input is restricted to be a single square-wave pulse. This approach was improved in [16] where the input is allowed to exhibit a series of step changes, with the input characterized by a single real eigenvalue between step changes. In [17], using an input similar to [16], it is shown that nonzero initial conditions can be identified in addition to the system transfer function. For the results in [14] through [17], the assumed system is not allowed to have a feedthrough term, and the system input is not allowed to have an eigenvalue that is identical to an eigenvalue of the system. Feedthrough terms are included in [18] and [19], however, and in [19] the assumed conditions are relaxed to allow the system to have an eigenvalue that is identical to the one associated with the input. In [20] the class of allowed input signals is expanded: the signals can exhibit jump discontinuities and can be characterized by a finite number of eigenvalues between discontinuities. The advantage of this more general input form is that it can be tailored to excite system modes in frequency ranges of interest. A point in common with several of the later methods mentioned above is that all input signals are required to be time-continuous during each interval of discontinuity, which does not lend well to systems that are microprocessor controlled. This research addresses this point and results in a method that allows for a sampled-and-held version of the general input of [20]. The method developed here provides a more simplified algorithm than that of [20] as a result of the sampled-and-held input.

While there are a large number of identification methods available in the literature; only a few methods of system identification techniques are reviewed in Chapter 2. Following this review of system identification techniques the method based on Prony signal analysis is developed for a discrete controlled system. Once the method is developed, several examples are provided to show various attributes

of the method. Further analysis is provided by comparing the Prony based method to another technique based on a signal energy analysis method that utilizes singular value decomposition. While the method that utilizes SVD for signal decomposition is easier to implement than Prony, there are fewer 'knobs' available for the user to work with in arriving at an optimal model. From an engineering perspective, it is often possible, that the more 'knobs' there are available to use, the better the chance to obtain an acceptable solution. Under the noisiest conditions the Prony method performs slightly better than the SVD method. As a last example, *discrete Prony* identification is applied to a one-machine infinite-bus power system to obtain a low-order model. Linear quadratic control synthesis based on this model is used for transient damping while the system experiences a transmission line fault. The control based on the *discrete Prony* identified model provides significant damping.

The second portion of this dissertation is concerned with control of systems which are modeled through system identification techniques that result in a model with an associated uncertainty set. Control systems are designed so that certain designated signals, such as tracking errors and actuator inputs, do not exceed prespecified levels. Hindering the achievement of this goal are uncertainty about the plant to be controlled and errors in measuring output signals. Uncertainty about the plant is a direct result of the fact that we use mathematical models in representing real physical systems. Furthermore, because sensors can measure signals only to a certain accuracy and because of the natural presence of noise in a system, additional uncertainty through identification occurs in the mathematical modeling process. The issue of uncertainty in the mathematical model leads to the *robust performance problem*, with the goal of achieving specified signal levels in the face of plant uncertainty.

Prior to the 1950s, classical control theory was well established based, in part, on the works of Nyquist [21] and Bode [22]. Following classical methods, control

theory entered the modern control era with emphasis placed on “state-space methods” (a name for all kinds of methodical procedures that has already been used for a long time, e.g., in analytical dynamics, quantum mechanics, theory of stability, in the solution of ordinary differential equations and in other fields). The application of these methods to automatic control was stimulated in the second half of the 1950s mainly by the work of L. S. Pontryagin [23, 24], by the method of dynamic programming as suggested by R. E. Bellman [25] and by the general theory of filtering and control theory elaborated by R. E. Kalman [26]. The recent literature on control systems that is based on state-space methods is very extensive.

Despite the seemingly obvious requirement of bringing plant uncertainty explicitly into control problems, it was only in the early 1980s that ‘modern’ control researchers re-established the link to the classical work of Nyquist, Bode and others by formulating a tractable mathematical notion of uncertainty in an input-output framework and developing rigorous mathematical techniques to cope with it. The basic technique is to model the plant as belonging to a set $\mathcal{P}$, where such a set can be either *structured* or *unstructured*. The important work of Doyle and Stein [27] established the basic framework for robust control synthesis for systems with unstructured uncertainty.

For an example of a structured set consider the plant model [28]

$$\frac{1}{s^2 + as + 1}.$$

This is a standard second-order transfer function with natural frequency 1 rad/sec. and damping ratio $a/2$ – it could represent, for example, a mass-spring damper or an R-L-C circuit. Suppose that the constant a is known only to the extent that it lies in some interval $[a_{min}, a_{max}]$. Then the plant belongs to the structured set

$$\mathcal{P} = \left\{ \frac{1}{s^2 + as + 1} : a_{min} \leq a \leq a_{max} \right\}.$$

For an example of unstructured uncertainty consider the nominal transfer function plant P with the perturbed plant transfer function of the form $\mathcal{P} = \{(1 + \Delta W)P\}$. Here W is a fixed stable transfer function, the weight, and Δ is a variable stable transfer function satisfying $\|\Delta\|_\infty \leq 1$. The uncertainty may be the result of unmodeled dynamics, particularly at high frequency. The infinity-norm on a transfer function is defined as

$$\|G\|_\infty \triangleq \sup_{\omega} |G(j\omega)|$$

and in simple terms says compute the maximum magnitude of the transfer function G over the entire frequency domain (i.e., the maximum gain of the transfer function).

As elaborated in [28], $|W(j\omega)|$ provides the uncertainty profile and at each frequency point the evaluation of $\mathcal{P}/P$ lies in the disk with center 1, radius $|W|$. Typically, $|W(j\omega)|$ is an increasing function of ω such that uncertainty increases with increasing frequency. The main purpose of Δ is to account for phase uncertainty and to act as a scaling factor on the magnitude of the perturbation (i.e., $|\Delta|$ varies between 0 and 1). This type of uncertainty is referred to as multiplicative uncertainty. More detail on other types of frequency domain (*unstructured*) uncertainty can be found in [29, 30] and references therein.

The focus of the robust control research in this dissertation is directed towards those dynamical systems that involve *structured* uncertainty in the form of a bounded ellipsoidal domain on the parameters of the model. In particular, the uncertainty is a direct result of system identification in the presence of noise.

Most identification methods when applied to a set of input/output data provide a model that is usually assumed to be exact for the particular operating point of interest. Quite often, the model is adequate for control design and implementation. In some instances, however, this approach may not be adequate. Thus, we consider the problem where the identified model is given in the form of a nominal set of

parameters including a range of uncertainty associated with the parameters. This type of uncertainty is sometimes referred to as "parametric uncertainty" and indeed falls under the class of *structured uncertainty*.

Recently, the works of [31]-[33] provide system identification methods where the uncertainty on the identified parameters is given in the form of a bounded ellipsoidal region around a nominal set of parameters. The ellipsoid uncertainty region resulting from the method given in [31] is strictly due to additive noise at the output of the system. The method given in [32] and [33] is due to a combination of including a model of unstructured uncertainty in the parameterized model and of additive noise at the output of the system. These methods of identification that result in a nominal set of parameters bounded by an ellipsoidal uncertainty domain are reviewed in Chapter 3.

A brief discussion on set-membership robust control is provided in Chapter 3. Set-membership robust control is simply a control strategy that assumes the system model is given in the form of a nominal set of parameters and an associated bounded uncertainty region. The idea of set-membership control may first have been introduced by Bertsekas and Rhodes in 1971 [34]. While set-membership is a general term, we consider a method of control where the parameters of uncertainty are known to lie within a bounded ellipsoidal domain. Recent results in set-membership control can be found in [35]-[37].

The robust control strategy is approached from a dynamic game theory point of view such that a saddle point solution is desired. For excellent discussion on differential and dynamic game theory see, for example, [38]-[40]. The saddle point is the result of a minimaximization of an energy (cost) function. The game theory method here has two players. Player one is the feedback control signal that attempts to minimize the cost. Player two is the set of uncertain parameters which attempts

to maximize the cost.

It is shown that the optimal control signal necessarily relates to Linear Quadratic Regulation (LQR). The theory of LQR formed a corner stone of the modern control era of the 1960s. The *final cost function* associated with a given plant and the LQR solution provides the means for an iterative approach to the minimaximization solution. The maximization is performed on the *final cost function* and is constrained by the ellipsoidal parameter bound and an appropriate Lyapunov equation from the LQR solution.

An iterative algorithm for both continuous-time and discrete-time systems is developed in Chapter 4. That the fixed-point solution of the algorithm is a saddle point is illustrated by several examples. A sufficient condition for the existence of the maximization portion of the discrete-time algorithm is given.

A closed-loop discrete controlled servo-mechanism example is also provided. The second order system is assumed to be unknown (i.e., a *black box*). Through an identification technique known as the *Optimal Volume Ellipsoid* (OVE) algorithm [31] (discussed in Chapter 3) a model and ellipsoid uncertainty region is found using noisy input/output data. A Kalman filter is used for state estimation. The discrete minimax algorithm is applied to determine the optimal state-feedback gain and the loop is closed. The servo example is stable and shows the worst-case tracking for the noisy, uncertain system. Results are compared to the case where the system is known (i.e., actual plant parameters are used).

A discussion of the results is provided in Chapter 5 and is followed by a section on possible future research in Chapter 6.

CHAPTER 2

DISCRETE PRONY IDENTIFICATION

In this chapter we extend previous results on transfer function identification using Prony signal analysis methods [20]. Recent work resulted in a generalized Prony identification method that incorporates piecewise continuous system input signals; between points of discontinuity, the input is characterized by input eigenvalues and input residues. We refer to the development of [20] as the *continuous Prony* identification method. Here we extend this work to account for sampled-and-held inputs that arise in digital filtering and control. This new method is referred to as *discrete Prony* identification. Using sampled input and output data, initial condition residues are estimated along with transfer function residues and eigenvalues. The two major phases of the *continuous Prony* method are modified to account for the sampled-and-held input; a major contribution is made to the way in which the residues are fit to corresponding eigenvalues in the second phase.

The organization of this chapter is as follows. First, a review of various *parametric* identification techniques is given as a means to establish the category in which Prony identification falls. Following this review of identification techniques the method of *discrete Prony* identification based on Prony signal analysis is presented. The *discrete Prony* algorithm was implemented using MATLAB, and examples are provided. The chapter ends with some concluding remarks.

Review of System Identification Methods

As a means of motivation for this chapter, a review of several methods of identification

is provided. This review is not meant to be exhaustive as this subject area is covered thoroughly in the literature, (e.g., see [1], [41] and [42]).

We can categorize identification methods in a number of ways. In this chapter we consider, among others, the attributes associated with model order selection and whether the method may be used on-line (real-time) or off-line (batch mode). Occasionally, exact knowledge of the system order is known, and identification methods that assume a priori system order should be used. For those situations where exact knowledge of the system order is not known, methods which make no a priori assumption on system order must be used. These methods start over-parameterized and must employ a stage within the identification process where model order selection (reduction) can be accomplished. Methods are available that optimally select the model order of the over-parameterized system. These order selection methods are well suited for on-line identification since they can often be coded. For off-line methods, optimal procedures for model order selection can be used but are not always necessary. Some applications, e.g., those that apply adaptive control require identification methods that can be implemented on-line. The nature of the system and the type of control strategy implemented often will dictate the type of identification to be used; thus, there are always trade-offs between identification schemes.

Identification methods can also be categorized further by considering the effects of noise in the system or the effects of modeled uncertainty. Those models that do not take into account noise are considered deterministic while those that do account for noise are considered stochastic models. Methods of identification that utilize models which include modeled uncertainty are referred to as robust methods of identification.

In the following subsections, we consider two popular models used in the control community and review some of the methods of analysis used to arrive at realizations of the model from the measured input and output data. Following this

review the *continuous Prony* method is introduced and the *discrete Prony* method is developed.

Autoregressive Models (ARX)

Consider the autoregressive model with exogenous input (ARX). The exogenous input is $u : \mathbb{N} \mapsto \mathbb{R}$. Assume that n_a parameters are associated with the AR part of the model, and n_b parameters are associated with the exogenous input. The ARX model is single-input/single-output (SISO) and can be given as

$$\begin{aligned} y(k) &= - \sum_{i=1}^{n_a} a_i y(k-i) + \sum_{j=1}^{n_b} b_j u(k-j) \\ &= \theta^T \phi(k) \end{aligned} \quad (2.1)$$

where the parameter vector to be estimated is

$$\theta^T = [a_1, \dots, a_{n_a}, b_1, \dots, b_{n_b}]$$

and the regression vector containing the past input and output values is given by

$$\phi(k) = [-y(k-1), \dots, -y(k-n_a), u(k-1), \dots, u(k-n_b)]^T.$$

The model order is assumed to be known a priori. Equation (2.1) is considered deterministic since there is no inclusion of noise in the model. The stochastic version can be obtained by introducing an additional noise term, say $\nu(k)$ to the right hand side of (2.1) as seen below.

$$\begin{aligned} y(k) &= - \sum_{i=1}^{n_a} a_i y(k-i) + \sum_{j=1}^{n_b} b_j u(k-j) + \nu(k) \\ &= \theta^T \phi(k) + \nu(k) \end{aligned} \quad (2.2)$$

where θ^T and $\phi(k)$ are defined above.

An expanded version of the ARX model is the autoregressive, moving average model with exogenous input (ARMAX). The moving average part comes from a third

set of parameters operating on the noise terms $\nu(k)$, and is given in (2.3).

$$\begin{aligned} y(k) &= -\sum_{i=1}^{n_a} a_i y(k-i) + \sum_{j=1}^{n_b} b_j u(k-j) + \sum_{l=1}^{n_c} c_l \nu(k-l) + \nu(k) \\ &= \theta^T \phi(k) + \nu(k) \end{aligned} \quad (2.3)$$

where the parameter vector to be estimated is

$$\theta^T = [a_1, \dots, a_{n_a}, b_1, \dots, b_{n_b}, c_1, \dots, c_{n_c}]$$

and the regression vector containing the past input, output and noise values is given by

$$\phi(k) = [-y(k-1), \dots, -y(k-n_a), u(k-1), \dots, u(k-n_b), \nu(k-1), \dots, \nu(k-n_c)]^T.$$

Here the size of the moving average vector, n_c , must be given a priori. Also, past noise terms are seldom known exactly, in which case estimates of them are used in place of actual noise values in the regression vector. It should be noted that the ARMAX model has become a standard tool for both system description and control design.

Least Squares Error Method After the model is established the method of determining the estimate of θ can be formulated. Consider the *least-squares* method. With (2.1) the prediction error becomes

$$\epsilon(k, \theta) = y(k) - \phi^T(k) \theta. \quad (2.4)$$

The *least-squares criterion* for (2.4) is

$$V_N(\theta) = \frac{1}{N} \sum_{k=1}^N \frac{1}{2} [y(k) - \phi^T(k) \theta]^2. \quad (2.5)$$

The *least-squares criterion* is a quadratic function in θ and can be minimized analytically, which gives, provided the indicated inverse exists,

$$\hat{\theta}_N^{LS} = \arg \min_{\theta \in \mathbb{R}} \left[\frac{1}{N} \sum_{k=1}^N \phi(k) \phi^T(k) \right]^{-1} \frac{1}{N} \sum_{k=1}^N \phi(k) y(k) \quad (2.6)$$

the *least-squares estimate* (LSE). From this development, one could introduce different weights on different measurements in the *least-squares criterion* resulting in the *weighted least-squares criterion*.

This method has not appealed to any statistical arguments for the estimation of θ . In fact, the framework of fitting these models to the input/output data makes sense regardless of the stochastic setting. Methods such as *maximum likelihood* are considered statistical parameter estimation techniques. Recursive forms of the ARX/LS and ARMAX/LS methods are available for on-line identification and adaptive control [41]. For a review of these and other methods see [1] and references therein.

State-Space Models

In the state-space form the relationship between the input and output signals is written as a system of first-order differential or difference equations using an auxiliary state vector $\mathbf{x}(t)$. This description of linear dynamical systems became an increasingly dominating approach after Kalman's work in 1960 [43] on prediction and linear quadratic control. This model is especially useful in that insights into physical mechanisms of the system can usually more easily be incorporated into state-space models than into models of the ARX form.

Consider the multi-input multi-output (MIMO) state-space model given by

$$\dot{\mathbf{x}}(t) = A\mathbf{x}(t) + B\mathbf{u}(t), \quad (2.7)$$

where $A \in \mathbb{R}^{n_a \times n_a}$, $B \in \mathbb{R}^{n_a \times m_i}$, the number of inputs is given by m_i and control $\mathbf{u} : \mathbb{R}^+ \mapsto \mathbb{R}^{m_i \times 1}$, and state $\mathbf{x} : \mathbb{R}^+ \mapsto \mathbb{R}^{n_a \times 1}$. The output of the system is given by

$$\mathbf{y}(t) = C\mathbf{x}(t) + D\mathbf{u}(t), \quad (2.8)$$

where $C \in \mathbb{R}^{m_o \times n_a}$ and $D \in \mathbb{R}^{m_o \times m_i}$; the number of outputs is given by m_o .

Consider the single-input single-output (SISO) version of (2.7) and (2.8) with $m_i = m_o = 1$. Let the Laplace transform of $u(t)$ and $y(t)$ be given as $U(s)$ and $Y(s)$, where $s \in \mathbb{C}$. The transfer function of the input/output model from $U(s)$ to $Y(s)$ is given in the Laplace variable s as

$$G(s) = \frac{b_{n_b+1}s^{n_b} + b_{n_b}s^{n_b-1} + b_{n_b-1}s^{n_b-2} + \dots + b_2s + b_1}{s^{n_a} + a_{n_a}s^{n_a-1} + \dots + a_2s + a_1} \quad (2.9)$$

where $n_b \leq n_a$. The poles of (2.9) are defined as the roots of the denominator polynomial which are equivalent to the eigenvalues of A from (2.7). For multi-input multi-output (MIMO) transfer functions and their characteristics, see [44, 45] and references therein.

A discretized version of the continuous time system of equations of (2.7) and (2.8) is given by (2.10) and (2.11) [46]. This discretized version is obtained when the input $u(t)$ is sampled-and-held periodically for a time period T .

$$\mathbf{x}((k+1)T) = F(T)\mathbf{x}(kT) + H(T)\mathbf{u}(kT) \quad (2.10)$$

and

$$\mathbf{y}(kT) = C\mathbf{x}(kT) + D\mathbf{u}(kT). \quad (2.11)$$

Here we have

$$F(T) = e^{AT} \quad (2.12)$$

and

$$H(T) = \left(\int_0^T e^{A\lambda} d\lambda \right) B. \quad (2.13)$$

If matrix A is nonsingular, $H(T)$ given by (2.13) can be simplified to

$$H(T) = A^{-1}(e^{AT} - I)B.$$

Note that the values of C and D of the continuous time state-space model are not affected by the sampled-and-held input signal, $\mathbf{u}(kT)$.

There have been various approaches to system identification using the state-space model. Most of the methods involve extensive use of singular value decomposition (SVD). The derivations of these procedures are relatively lengthy and only a few are described below. It is noted that each of the identification approaches is based on some distinctive method of analysis, as can be seen in the following sections.

Principal Component Analysis Method Consider the approach based on the pulse response matrix of the strictly proper ($D = 0$) discrete system of (2.10) and (2.11). The response $y(k)$ of the system is given by

$$y(k) = CF^k x(0) + \sum_{i=1}^k CF^{i-1} H u(k-i), k \geq 1. \quad (2.14)$$

The pulse response matrix (also termed Markov parameters) is given by

$$y_M(k) = CF^{k-1}H, k = 1, 2, \dots \quad (2.15)$$

The system Hankel matrix can be constructed from these Markov parameters as follows

$$\Gamma_{rs}(k-1) = \begin{bmatrix} y_M(k) & y_M(k+t_1) & \cdots & y_M(k+t_{s-1}) \\ y_M(j_1+k) & y_M(j_1+k+t_1) & \cdots & y_M(j_1+k+t_{s-1}) \\ \vdots & \vdots & \ddots & \vdots \\ y_M(j_{r-1}+k) & y_M(j_{r-1}+k+t_1) & \cdots & y_M(j_{r-1}+k+t_{s-1}) \end{bmatrix}$$

where $j_i (i = 1, \dots, r-1)$ and $t_i (i = 1, \dots, s-1)$ are arbitrary integers which can be adjusted to include (or eliminate) specific observations in the Hankel matrix. In [47] a realization (identification) method is developed which provides a full signal model in a single stage, from a single SVD of the system Hankel matrix. Let us suppose for the sake of simplicity that $\Gamma_{rs}(0)$ is a square matrix of order $N = m_o r = m_i s$, where m_o and m_i are defined in (2.7) and (2.8). The SVD then consists of finding an $N \times N$ orthonormal matrix P_N and an $N \times N$ orthonormal matrix Q_N so that

$$\Gamma_{rs}(0) = P_N \Lambda_N Q_N^T$$

with $\Lambda_N = \text{diag}(\lambda_1, \lambda_2, \dots, \lambda_N)$ and $\{\lambda_i\} : \lambda_1 \geq \lambda_2 \geq \dots \geq \lambda_\eta \geq \epsilon \geq \lambda_{\eta+1} \geq \dots \geq \lambda_N \geq 0$. Based on this decomposition, the rank η of $\Gamma_{rs}(0)$ is found as the index of the first nil singular value (for noisy matrices and the relative sizes of the singular values, the nil value is subject to interpretation and is here shown as ϵ).

If the order is chosen as η , the realization becomes

$$F = \Lambda_\eta^{-1/2} P_\eta^T \Gamma_{rs}(1) Q_\eta \Lambda_\eta^{-1/2}$$

$$H = \Lambda_\eta^{1/2} Q_\eta^T E_{m_i}$$

$$C = E_{m_o}^T P_\eta \Lambda_\eta^{1/2}$$

where $E_{m_i}^T = [I_{m_i}, \odot]$ and $E_{m_o}^T = [I_{m_o}, \odot]$. The terms I_{m_i} and I_{m_o} are identity matrices and the term $\odot$ is the null matrix of appropriate dimension. The construction of P_η and Q_η are from P_N and Q_N , respectively; further details can be found in [47].

The method may be considered to be a two-stage algorithm since the model order must be determined prior to calculating the final model. However, once the model order is chosen, all parameters are calculated in one step; thus, the algorithm is, in truth, a single stage algorithm. This approach to system identification, although simple, is not realistic for systems with significant amounts of noise. The model order selection, which is based on the singular values of the Hankel matrix, becomes unreliable under noisy conditions. Also the type of input used for probing is limited to that of a pulse. However, the models to be realized can be SISO, MIMO, or SIMO and are all minimal, which is an advantage over ARX methods.

Canonical Variate Analysis Methods The fundamental concept in the canonical variate analysis (CVA) approach is the *past* and *future* of the process [48], and is based on statistical principles. Associated with each time k is a past vector $\mathbf{p}_k$ consisting of the past outputs and inputs occurring prior to time k as well as a future

vector $\mathbf{f}_k$ consisting of outputs at time k or later,

$$\begin{aligned}\mathbf{p}_k &= [\mathbf{y}^T(k-1), \mathbf{y}^T(k-2), \dots, \mathbf{u}^T(k-1), \mathbf{u}^T(k-2), \dots]^T, \\ \mathbf{f}_k &= [\mathbf{y}^T(k), \mathbf{y}^T(k+1), \dots]^T.\end{aligned}\quad (2.16)$$

The vector processes $\mathbf{y}(k)$ and $\mathbf{u}(k)$ are assumed to be jointly stationary and the covariance matrices among the past $\mathbf{p}_k$ and the future $\mathbf{f}_k$ are denoted as Σ_{ff} , Σ_{pp} , and Σ_{fp} . Now suppose that for a specified state order η , we wish to determine η linear combinations of the past $\mathbf{p}_k$ which allow the optimal prediction of the future $\mathbf{f}_k$. The set of η linear combinations of the past $\mathbf{p}_k$ is denoted as an $\eta \times 1$ vector $\mathbf{m}_k$ and considered as η -order memory of the past. For any selection of the memory $\mathbf{m}_k$, the optimal linear prediction $\hat{\mathbf{f}}_k$ is the function [48]

$$\hat{\mathbf{f}}_k(\mathbf{m}_k) = \Sigma_{fm} \Sigma_{mm}^{-1} \mathbf{m}_k \quad (2.17)$$

of the reduced order memory $\mathbf{m}_k$. The prediction error is measured using the quadratic weighting

$$\mathbb{E}\{\|\mathbf{f}_k - \hat{\mathbf{f}}_k\|_{\Lambda}^2\} = \mathbb{E}\{(\mathbf{f}_k - \hat{\mathbf{f}}_k)^T \Lambda^\dagger (\mathbf{f}_k - \hat{\mathbf{f}}_k)\} \quad (2.18)$$

where $\mathbb{E}$ is the expectation operation and Λ is an arbitrary positive semidefinite symmetric matrix so that the pseudo-inverse $\Lambda^\dagger$ is an arbitrary quadratic weighting that is possibly singular. For system identification, the use of the weighting matrix $\Lambda = \Sigma_{ff}$ results in a near maximum likelihood system identification procedure.

In terms of these quantities, the following problem is stated. For a given order η , determine an optimal η -order memory

$$\mathbf{m}_k = J_\eta \mathbf{p}_k \quad (2.19)$$

by choosing the η rows of J_η such that the optimal linear predictor $\hat{\mathbf{f}}_k(\mathbf{m}_k)$ based on $\mathbf{m}_k$ minimizes the prediction error (2.18). The method to arrive at J_η is the subject of

the following discussion. The vector $\mathbf{m}_k$ is intentionally called 'memory' rather than 'state'. A given selection of memory $\mathbf{m}_k$ normally will not correspond to the state of any well defined η -order Markov process. For the system identification problem, this is not a problem since many orders of η will be considered and the one giving the best prediction will be chosen as the optimal order.

Theorem 2.1 (Larimore [48]) Consider the problem of choosing η linear combinations $\mathbf{m}_k = J_\eta \mathbf{p}_k$ of $\mathbf{p}_k$ for predicting $\mathbf{f}_k$, such that (2.18) is minimized where Σ_{pp} , and Λ are possibly singular positive semidefinite symmetric matrices with ranks m and n respectfully. Then the existence and uniqueness of solutions are completely characterized by the (Σ_{pp}, Λ) -generalized singular value decomposition which guarantees the existence of matrices J , L , and generalized singular values $\gamma_1, \dots, \gamma_q$ such that

$$\begin{aligned} J \Sigma_{pp} J^T &= I_m, \quad L \Lambda L^T = I_n, \\ J \Sigma_{pf} L^T &= \text{diag}(\gamma_1 \geq \dots \geq \gamma_q > 0, \dots, 0). \end{aligned} \quad (2.20)$$

The solution is given by choosing the rows of J_η if the η -th singular value satisfies $\gamma_\eta > \gamma_{\eta+1}$. If there are r repeated singular values equal to γ_η , then there is an arbitrary selection from among the corresponding singular vectors, i.e. rows of J . Thus, the condition that $\gamma_\eta > \gamma_{\eta+1}$ merely defines a decision point for model order selection. The model order η is to some degree arbitrary, however, the optimal state order can be determined by use of the Akaike information criterion. The minimum value is

$$\min_{\text{rank}(J_\eta \Sigma_{pp} J_\eta^T) = \eta} \mathbb{E}\{\|\mathbf{f}_k - \hat{\mathbf{f}}_k\|_{\Lambda}^2\} = \text{tr} \Lambda^\dagger \Sigma_{ff} - \gamma_1^2 - \dots - \gamma_\eta^2. \quad (2.21)$$

The vectors of variables $\mathbf{c} = J \mathbf{p}_k$ and $\mathbf{d} = L \mathbf{f}_k$ are canonical variables and (2.20) characterizes a *canonical variate analysis*. Now consider the system described by (2.10) and (2.11) where $\mathbf{u}(k)$ and $\mathbf{y}(k)$ are random processes. In [48] models of

noise are included; here we have omitted them for ease of presentation. We wish to model and predict the future of $y(k)$ by an η -state $x(k)$. The particular multivariate regression equations are expressed in terms of covariances, denoted by Σ , among various vectors as

$$\begin{bmatrix} F & H \\ C & D \end{bmatrix} = \Sigma \left[\begin{pmatrix} m(k+1) \\ y(k) \end{pmatrix}, \begin{pmatrix} m(k) \\ u(k) \end{pmatrix} \right] \Sigma^{-1} \left[\begin{pmatrix} m(k) \\ u(k) \end{pmatrix}, \begin{pmatrix} m(k) \\ u(k) \end{pmatrix} \right]. \quad (2.22)$$

Explicit computation is obtained by the substitution of $m_k = J_\eta p_k$.

To decide on the model state order or model structure, recent developments based upon entropy or information measures are used. Such methods were originally developed by Akaike [49] and involve the use of the Akaike Information Criterion (AIC) for deciding the appropriate order of a statistical model. In simplest terms the AIC is a procedure that minimizes negative entropy. The AIC for each order η is defined by

$$AIC(\eta) = -2 \log p(Y^N, U^N; \hat{\theta}_\eta) + 2M_\eta \quad (2.23)$$

where p is the likelihood function based on the observations (Y^N, U^N) at N time points, and where $\hat{\theta}_\eta$ is the maximum likelihood parameter estimate using an η -order model with M_η parameters. For more information on likelihood functions see [1, pages 181-190]. The model order η is chosen corresponding to the minimum value of the $AIC(\eta)$. The number of parameters in (2.10) and (2.11) is

$$M_\eta = (2\eta + m_i)m_o + m_i m_o + m_o(m_o + 1)/2$$

where m_i and m_o are the number of input and output variables, respectively. This result is developed by considering the size of the equivalent class of state space models having the same input/output and noise characteristics [50]. Thus the number of functionally independent parameters in a state-space model is far less than the number of elements in the various state-space matrices. Effectively, the AIC imposes a statistical penalty on the order (parameter size) of the model.

State Variable Analysis Methods An approach that is in some ways similar to CVA, referred to as State Variable Analysis (SVA), attempts to estimate the state of the system, and from the state, produce a realization of the state-space model. A particular formulation of an SVA algorithm is given in [51] and is briefly discussed as follows. First, a time-series state-vector basis for the dynamical system is realized as the intersection of the row spaces of two block Hankel matrices constructed with measured input/output (I/O) data. This step is done by repeated use of SVD. The second step solves a set of linear equations for F , H and C . The set of linear equations for step two is constructed from the basis vector estimated in step one.

The algorithm assumes that the data are consistent with the discrete-time state-space model of (2.10) and (2.11). The algorithm utilizes a Hankel matrix Υ ,

$$\Upsilon = \begin{bmatrix} \Upsilon_1 \\ \Upsilon_2 \end{bmatrix} \quad (2.24)$$

where

$$\Upsilon_1 = \begin{bmatrix} \mathbf{u}(k) & \mathbf{u}(k+1) & \cdots & \mathbf{u}(k+j-1) \\ \mathbf{y}(k) & \mathbf{y}(k+1) & \cdots & \mathbf{y}(k+j-1) \\ \mathbf{u}(k+1) & \mathbf{u}(k+2) & \cdots & \mathbf{u}(k+j) \\ \mathbf{y}(k+1) & \mathbf{y}(k+2) & \cdots & \mathbf{y}(k+j) \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{u}(k+i-1) & \mathbf{u}(k+i) & \cdots & \mathbf{u}(k+j+i-2) \\ \mathbf{y}(k+i-1) & \mathbf{y}(k+i) & \cdots & \mathbf{y}(k+j+i-2) \end{bmatrix}$$

and

$$\Upsilon_2 = \begin{bmatrix} \mathbf{u}(k+i) & \mathbf{u}(k+i+1) & \cdots & \mathbf{u}(k+j+i-1) \\ \mathbf{y}(k+i) & \mathbf{y}(k+i+1) & \cdots & \mathbf{y}(k+j+i-1) \\ \mathbf{u}(k+i+1) & \mathbf{u}(k+i+2) & \cdots & \mathbf{u}(k+j+i) \\ \mathbf{y}(k+i+1) & \mathbf{y}(k+i+2) & \cdots & \mathbf{y}(k+j+i) \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{u}(k+2i-1) & \mathbf{u}(k+2i) & \cdots & \mathbf{u}(k+j+2i-2) \\ \mathbf{y}(k+2i-1) & \mathbf{y}(k+2i) & \cdots & \mathbf{y}(k+j+2i-2) \end{bmatrix}$$

Application of singular value decomposition is used to estimate a basis, X , from which the time-series state vector can be obtained. It is shown in [51] that

$$\text{span}_{\text{row}}(X) = \text{span}_{\text{row}}(\Upsilon_1) \cap (\Upsilon_2),$$

where the basis, X , is given as

$$X = [x_{k+i} \ x_{k+i+1} \ \dots \ x_{k+i+j-2}].$$

Note, we have replaced $x(k)$ with subscript notation x_k . As stated in [51], i and j should be chosen sufficiently large (insuring sufficient information about the system), and $j \gg i$ so that computational load and noise sensitivity are reduced. With the state-vector basis determined the order of the system must be decided before the state-space matrices of (2.10) and (2.11) can be estimated. If the system is completely noise free the singular values of the SVD will reveal the exact order of the system. If noise is present one must observe the relative magnitudes of the singular values and determine an appropriate order for the model. This problem of the effect of noise on the singular values of the system Hankel matrix has been addressed in [52]. In [52] it is shown that small perturbations in the input and output matrices H and C can reveal those modes that are associated with the actual system and those modes associated with noise. Also, a selection procedure similar to the AIC may be employed for model order reduction.

With the order and resulting basis X established, the following set of linear equations may be solved in a least-squares sense for F , H , C and D .

$$\begin{bmatrix} X_{k+i+1}, \dots, X_{k+j+i-1} \\ y_{k+i}, \dots, y_{k+j+i-2} \end{bmatrix} = \begin{bmatrix} F & H \\ C & D \end{bmatrix} \begin{bmatrix} X_{k+i}, \dots, X_{k+j+i-2} \\ u_{k+i}, \dots, u_{k+j+i-2} \end{bmatrix}. \quad (2.25)$$

This method can be adapted for on-line identification.

This review has described some of the modern techniques of identification being used and it revealed issues that one must be concerned with when choosing a method. Each of the methods was presented in its simplest form. Each method has been extended to account for various types of noise, see, e.g., [53, 54]. Following is the development of the *discrete Prony* method. At the end of this chapter *discrete Prony* will be compared to the SVA method just described.

Prony Signal Identification for Discrete Control Systems

Introduction: Discrete Prony

The Prony method for system identification is based on a signal analysis method for approximating a signal with a weighted sum of exponentials. A complete survey of the development and uses of Prony methods in system identification is given in [20]. Since the original work of Prony [2], much has been accomplished in the way of system identification based on this analysis technique.

Most recently [20] has provided a general Prony approach for system identification where the probing input is piecewise continuous; between points of discontinuity, the input is characterized by input eigenvalues and input residues. The procedure is given in essentially two steps: a) modal (eigenvalue) identification and b) residue fitting. Within each of these steps there are additional steps that can be taken to aid in the final transfer function model optimal selection. The method results in estimates of initial condition residues, transfer-function residues, and system eigenvalues. The development in [20] is set in the continuous time domain and leads to a relatively complicated procedure in step b.

Unlike the method in [20] the development here is for those systems that are implemented using sampled-and-held inputs. This development leads to a straightforward procedure for step b.

System Characterization

System Model The system model shown in Figure 1 is a SISO system. Through partial fraction expansion the system model from (2.9) has the Laplace transform function represented in standard parallel form:

$$\hat{G}(s) = R_0 + \sum_{j=1}^{\eta} \frac{R_j}{s - \xi_j}. \quad (2.26)$$

In Figure 1 the initial condition terms are included explicitly in the summation preceding the output $\hat{y}(t)$, so that the input $u(t)$ can be taken as 0 for $t < 0$. The input $u(t)$ is sampled-and-held prior to its application to both the model and the system. The constant sample period is T . Although a zero-order-hold (ZOH) circuit is indicated in Figure 1, the approach that follows can be modified to account for other hold circuits. The ξ_j 's are the eigenvalues of the system model, R_0 is a feed-through gain, R_1 through R_η are the model residues, and E_1 through E_η are initial condition residues. The ξ_j 's are assumed to be distinct and can occur in complex conjugate pairs. Residues corresponding to complex conjugate eigenvalues also occur in complex conjugate pairs. The objective of the identification procedure is to find values of ξ_j 's, R_j 's, E_j 's and η so that the model's output $\hat{y}(t)$ is close as possible, in some appropriate sense, to the actual system output $y(t)$.

If there is a dc offset in the output, then some ξ_j will equal zero with the associated $E_j \neq 0$; and if there is no eigenvalue with value zero in $G(s)$, then the associated $R_j \cong 0$.

General System Input For the identification method that is developed in this thesis, the input $u(t)$ for $t \geq 0$ is assumed to be of the general form:

$$u(t) = \sum_{k=1}^q \sum_{l=1}^{m_k} c_{k,l} (e^{\mu_{k,l}(t-\tau_{k-1})}) [u_s(t - \tau_{k-1}) - u_s(t - \tau_k)], \quad (2.27)$$

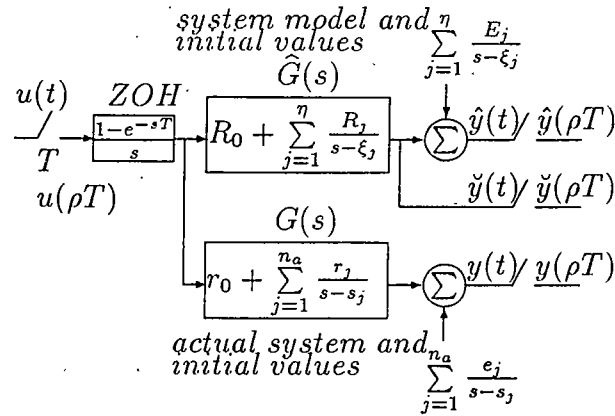


Figure 1. System and actual model in parallel form.

which is discontinuous at a finite number $q + 1$ points in time. Here the step function is represented by $u_s(\cdot)$. The k^{th} input time interval is characterized by $t \in [\tau_{k-1}, \tau_k)$ where $\tau_0 = 0$ without loss of generality. A total of $q + 1$ time intervals exist for $t \geq 0$, where the $(q + 1)^{th}$ time interval corresponds to $t \geq \tau_q$ in which $u(t) = 0$.

For each input interval, $(\tau_k - \tau_{k-1})/T$ is assumed to be an integer value. Let the integer M_k be defined by

$$M_k \triangleq \frac{\tau_k}{T}, \quad k = 0, 1, \dots, q.$$

The input time intervals can be characterized in terms of the M_k 's: letting $t = \rho T$, $\rho = 0, 1, \dots$, the input time interval is characterized by integer time $\rho \in [M_{k-1}, M_k)$ for $k = 1, 2, \dots, q$. During the k^{th} time interval, the input signal is characterized by the set of input eigenvalues $\{\mu_{k,l}\}_1^{m_k}$ with corresponding amplitude set $\{c_{k,l}\}_1^{m_k}$. The input eigenvalues can be selected to excite specific frequency ranges of interest. All values of the input signal parameters are assumed known.

Output Response Using $u(t)$ of (2.27) as the input in Figure 1, the model output $\hat{y}(t)$ is given by

$$\hat{y}(t) = \sum_{j=1}^{\eta} E_j e^{\xi_j t} + \check{y}(t), \quad (2.28)$$

where $\check{y}(t)$ is shown in Figure 1.

In order to employ the method of Prony in modal identification a thorough understanding of the sampled output signal in terms of the input and system modes is necessary. The analysis can be quite tedious because the system is being driven by a zero-order hold circuit and the input signal is piece-wise continuous. Therefore, for purposes of explanation, consider a first-order model

$$\hat{G}(s) = \frac{R}{s - \xi},$$

with initial conditions IC,

$$IC(s) = \frac{E}{s - \xi}.$$

Allow the input to be given by

$$u(t) = e^{\mu(t-\tau_{k-1})} [u_s(t - \tau_{k-1}) - u_s(t - \tau_k)]. \quad (2.29)$$

The following derivations make use of various properties and theorems of the z-transform and may be found in [46, pages 51-57, 86 and 152]. Let the z-transform operator be represented by $\mathcal{Z}\{\cdot\}$; then the z-transform of the input of (2.29) is

$$U(z) = \mathcal{Z}\{u(t)\} = z^{-M_{k-1}} [1 - z^{-\Delta_M} \gamma^{\Delta_M}] \frac{z}{z - \gamma}, \quad (2.30)$$

where

$$\Delta_M \triangleq M_k - M_{k-1} \text{ and } \gamma \triangleq e^{\mu T}.$$

The z-transform of IC(s) is

$$IC(z) = \mathcal{Z}\left\{\frac{E}{s - \xi}\right\} = E \frac{z}{z - \lambda}, \quad (2.31)$$

where

$$\lambda \triangleq e^{\xi T}.$$

Next, consider Figure 1 and (2.30) and (2.31); then the z-transform of $\hat{y}(t)$ is given by

$$\begin{aligned}\hat{Y}(z) &= IC(z) + U(z)(1 - z^{-1})\mathcal{Z}\left\{\frac{R}{s(s - \xi)}\right\} \\ &= E\frac{z}{z - \lambda} + z^{-M_{k-1}}[1 - z^{-\Delta_M}\gamma^{\Delta_M}]\frac{\frac{R}{\xi}(\lambda - 1)z}{(z - \lambda)(z - \gamma)}.\end{aligned}\quad (2.32)$$

There are three cases to consider when analyzing the output signal.

Case 1) $\rho: \rho T < M_{k-1}T$

It can be seen from (2.32) that the only part of $\hat{Y}(z)$ that has nonzero inverse when $\rho < M_{k-1}$ is

$$E\frac{z}{z - \lambda},$$

and thus

$$\hat{y}(\rho T) = E\lambda^\rho, \quad \rho < M_{k-1}.$$

Case 2) $\rho: M_{k-1}T \leq \rho T < M_k T$

In this case, those parts of $\hat{Y}(z)$ having nonzero inverse terms over the time interval are

$$E\frac{z}{z - \lambda} + z^{-M_{k-1}}\frac{\frac{R}{\xi}(\lambda - 1)z}{(z - \lambda)(z - \gamma)}$$

and thus

$$\hat{y}(\rho T) = E\lambda^\rho + \frac{R(\lambda - 1)}{\xi(\lambda - \gamma)}[\lambda^{\rho - M_{k-1}} - \gamma^{\rho - M_{k-1}}], \quad M_{k-1} \leq \rho < M_k. \quad (2.33)$$

Case 3) $\rho: \rho T \geq M_k T$

It can be seen from (2.32) and (2.33) that $\hat{y}(\rho T)$ takes on the following form

$$\hat{y}(\rho T) = E\lambda^\rho + \frac{R(\lambda - 1)}{\xi(\lambda - \gamma)}[\lambda^{\rho - M_{k-1}} - \gamma^{\Delta_M}\lambda^{\rho - M_k}], \quad \rho \geq M_k.$$

Hence, it can be seen that while the input is off the output signal contains only those modes associated with the system, whereas, while the input is on, as in case 2, the output signal is comprised of input and system modes. Further note that in case 3 the modal content of the output signal is comprised of linear combinations of the system mode, λ , where the constant multipliers of this system mode are a function of the past input mode, γ .

In general it can be said that during a given interval k the modal content of the output signal consists of linear combinations of both the input and system modes, and while the input is off, the modal content of the output signal consists of linear combinations of the system modes.

Thus, for any interval defined by $[\tau_{k-1}, \tau_k)$ such that the probing input is on, (i.e. for $k = 1, 2, \dots, q$), the part of $\hat{Y}(z)$ that has a nonzero inverse can be characterized by

$$[\hat{Y}(z)]_k \triangleq \left[\frac{n_{\hat{y}}(z^{-1})}{d_{\hat{y}}(z^{-1})} \right]_k = \frac{[n_{\hat{y}}(z^{-1})]_k}{d_{\hat{G}}(z^{-1}) [d_u(z^{-1})]_k}, \quad (2.34)$$

where $d_{\hat{G}}(z^{-1})$ is the characteristic polynomial of the model, $[d_u(z^{-1})]_k$ is the characteristic polynomial for the input over the k^{th} interval and $[n_{\hat{y}}(z^{-1})]_k$ is an appropriate numerator polynomial. Otherwise, for $k = q + 1$ or while the input is off

$$[\hat{Y}(z)]_k = \frac{[n_{\hat{y}}(z^{-1})]_k}{d_{\hat{G}}(z^{-1})}. \quad (2.35)$$

From (2.34) it can be seen that the modal content of the sampled output signal consists of a combination of input and system modes, whereas, from (2.35) the modal content of the sampled output signal consists only of those contributed by the system.

Prony Based Analysis

Basic Properties The objective of the first step of Prony analysis is to find the eigenvalues of the sampled output signal $y(t)$. We assume that the signal $y(t)$ is

sampled at a sample period T smaller than the Nyquist period. During the k^{th} time interval the sampled signal $\hat{y}(\rho T)$ can be rewritten in discrete-time form as

$$\hat{y}_\rho = \sum_{i=1}^{\nu_k} \beta_{k,i} z_i^{\rho - M_{k-1}}, \quad \rho = M_{k-1}, \dots, M_k - 1,$$

where ρ is integer time, $\hat{y}_\rho \equiv \hat{y}(\rho T)$, $\beta_{k,i} \in \mathbb{C}$ for $i = 1, 2, \dots, \nu_k$, $\nu_k \triangleq \eta + m_k$, and each z_i is a discrete-time eigenvalue of the system or of the input. It is well known that $\hat{y}_\rho$ satisfies its own characteristic equation, [2, 9]; thus from (2.34) and (2.35), it follows that

$$[d_{\hat{y}}(z^{-1})]_k \hat{y}_\rho \cong 0, \quad \rho = M_{k-1} + \nu_k, \dots, M_k - 1, \quad (2.36)$$

where the characteristic polynomial of (2.34) and (2.36) is defined as

$$[d_{\hat{y}}(z^{-1})]_k \triangleq 1 - (\phi_{k,1} z^{-1} + \phi_{k,2} z^{-2} + \dots + \phi_{k,\nu_k} z^{-\nu_k}). \quad (2.37)$$

Thus, for $\rho \in [M_{k-1} + \eta + m_k, M_k - 1]$, where the definition for ν_k has been substituted, we have from (2.36) and (2.37)

$$\hat{y}_\rho \cong \phi_{k,1} \hat{y}_{\rho-1} + \phi_{k,2} \hat{y}_{\rho-2} + \dots + \phi_{k,\eta+m_k} \hat{y}_{\rho-\eta-m_k}. \quad (2.38)$$

Allowing ρ to range over the defined interval and substituting y_ρ for $\hat{y}_\rho$ provides a system of over-determined linear equations (i.e., more parameters than necessary to describe the 'true' system) that can be solved in a least-squares sense for the coefficients $\phi_{k,i}$. Factoring (2.37) then results in estimates for the eigenvalues of the signal. By removing the eigenvalues $\{\mu_{k,l}\}_1^{m_k}$ associated with the input signal, the desired estimates for the eigenvalues $\{\xi_j\}_1^\eta$ of the system are obtained along with residual eigenvalues due to the over-determination.

Modal Identification Three different approaches to obtaining the set $\{\xi_j\}_1^\eta$ are described in detail in [20] and are referred to as i) Separate least-squares solutions; ii) Combined least-squares solution for system eigenvalues; and iii) Combined

least-squares solution for all eigenvalues. Equations (2.36) - (2.38) are the basis for method i.

The second approach provides a good degree of insight to the first stage of Prony analysis and will be discussed below. Method ii requires the construction of auxiliary equations with knowledge of the input signal and solves directly for the system eigenvalues, ξ_j 's. Let the characteristic polynomial of the system be given by

$$d_{\hat{G}}(z^{-1}) \triangleq 1 - (\psi_1 z^{-1} + \psi_2 z^{-2} + \dots + \psi_\eta z^{-\eta}) \quad (2.39)$$

and in factored form

$$d_{\hat{G}}(z^{-1}) \equiv \prod_{j=1}^{\eta} (1 - e^{\xi_j T} z^{-1}). \quad (2.40)$$

The characteristic polynomial of the input for the k^{th} interval is given by

$$[d_u(z^{-1})]_k \triangleq 1 + \alpha_{k,1} z^{-1} + \alpha_{k,2} z^{-2} + \dots + \alpha_{k,m_k} z^{-m_k}$$

and in factored form

$$[d_u(z^{-1})]_k \equiv \prod_{l=1}^{m_k} (1 - e^{\mu_{k,l} T} z^{-1}). \quad (2.41)$$

Over the k^{th} time interval, from (2.34) we have

$$d_{\hat{G}}(z^{-1}) ([d_u(z^{-1})]_k y_\rho) \cong 0$$

or

$$d_{\hat{G}}(z^{-1}) p_k(\rho) \cong 0, \quad (2.42)$$

where $p_k(\rho)$ is generated based on the known $[d_u(z^{-1})]_k$ of (2.41). The form of $p_k(\rho)$ is given by

$$p_k(\rho) = y_\rho + \alpha_{k,1} y_{\rho-1} + \alpha_{k,2} y_{\rho-2} + \dots + \alpha_{k,m_k} y_{\rho-m_k}, \quad (2.43)$$

where $\rho \in [M_{k-1} + m_k, M_k - 1]$. Using (2.39), (2.42) and (2.43), and appropriately narrowing the range on ρ , we have

$$p_k(\rho) \cong \psi_1 p_k(\rho - 1) + \psi_2 p_k(\rho - 2) + \dots + \psi_\eta p_k(\rho - \eta), \quad (2.44)$$

where $\rho \in [M_{k-1} + m_k + \eta, M_k - 1]$.

For some input interval, say $k = r$; if $M_r - 1$ is less than $M_{r-1} + m_r + \eta$, then the r^{th} interval of data cannot be used in the construction of the above set of equations. Also, for interval $k = q + 1$ or any interval where the input $u(t) = 0$, it follows from (2.35) and (2.39) that

$$y_\rho \cong \psi_1 y_{\rho-1} + \psi_2 y_{\rho-2} + \cdots + \psi_\eta y_{\rho-\eta}, \quad (2.45)$$

where

$$\rho : \begin{cases} \rho \in [M_{k-1} + \eta, M_k - 1] & \text{if } u(t) = 0, \\ \rho \geq M_{k-1} + \eta & \text{if } k = q + 1. \end{cases}$$

The unknowns in (2.44) and (2.45) are the ψ_i 's. For each time interval of sufficient width, and where $u(t) \neq 0$, a set of linear equations in the following form result:

$$\mathbf{v}_k \cong \Phi_k \psi, \quad (2.46)$$

where $\psi \in \mathbb{R}^\eta$ consists of ψ_i 's, the Toeplitz matrix $\Phi_k \in \mathbb{R}^{(M_k - M_{k-1} - m_k - \eta, \eta)}$ consists of appropriate ordering of $p_k(\rho - 1), \dots, p_k(\rho - \eta)$, and $\mathbf{v}_k \in \mathbb{R}^{(M_k - M_{k-1} - m_k - \eta)}$ consists of appropriate ordering of $p_k(\rho)$. For intervals where $u(t) = 0$ or $k = q + 1$, we have the same ψ , however, the Toeplitz matrix $\Phi_k \in \mathbb{R}^{(M_k - M_{k-1} - \eta, \eta)}$ and $\mathbf{v}_k \in \mathbb{R}^{(M_k - M_{k-1} - \eta)}$ each consist of appropriate ordering of y_ρ . As suggested in [20], it may be necessary for numerical reasons to row scale each interval set of equations such that the matrices have same order of magnitude. In any case, with all input intervals of sufficient length used in (2.46) the concatenation results in an over-determined set of equations:

$$\mathbf{v} \cong \Phi \psi.$$

To arrive at a set of parameters ψ_j 's, the following least-squares problem can be solved

$$\psi = \arg \min_{\psi} \|\mathbf{v} - \Phi \psi\|_2.$$

Once ψ is determined, a high precision root-finding algorithm or eigenvalue solving algorithm can be used to factor $d_{\hat{G}}(z^{-1})$ of (2.39) to obtain $z_j \triangleq e^{\xi_j T}$, (the roots of (2.40)). It is then a simple matter to solve for the set, $\{\xi_j\}_1^\eta$.

Residue Fitting With the first phase of Prony identification resulting in the set of ξ_j 's it remains to estimate the R_j 's and E_j 's. Equation (2.26) can be rewritten as

$$\hat{G}(s) = R_0 + \sum_{j=1}^{\eta} \hat{G}_j(s), \quad (2.47)$$

where $\hat{G}_j(s) \triangleq \frac{R_j}{s - \xi_j}$.

Each subsystem $\hat{G}_j(s)$ can be written in state-space parameters as

$$\hat{G}_j(s) = C_j(sI - A_j)^{-1}B_j + D_j,$$

where $A_j = \xi_j$, $D_j = 0$ and we are free to choose $B_j = 1$ and $C_j = R_j$ for all j . Next, recall that the system has been probed by an input signal which has been sampled-and-held, thus the discretization of $\hat{G}_j(s)$ (see (2.10) and (2.11)) is given by

$$\mathbf{x}_j((\rho + 1)T) = F_j(T)\mathbf{x}_j(\rho T) + H_j(T)u(\rho T) \quad (2.48)$$

and

$$\check{y}_j(\rho T) = R_j \mathbf{x}_j(\rho T). \quad (2.49)$$

Here we have

$$F_j(T) = e^{\xi_j T} \quad (2.50)$$

and

$$H_j(T) = \int_0^T e^{\xi_j t} dt. \quad (2.51)$$

If $\xi_j \neq 0$

$$H_j(T) = \frac{e^{\xi_j T} - 1}{\xi_j}$$

otherwise

$$H_j(T) = T.$$

From (2.28) and (2.49) the estimate of the output can be written as

$$\hat{y}(\rho T) = \sum_{j=1}^{\eta} \check{y}_j(\rho T) + R_0 u(\rho T) + \sum_{j=1}^{\eta} E_j e^{\xi_j \rho T},$$

where $\rho = 0, 1, 2, \dots, N$. The problem statement can be stated in the following form:

$$[R, E] = \arg \min_{[R, E]} \|y - \hat{y}\|_2. \quad (2.52)$$

Using (2.48) - (2.51), let ε be defined by

$$\varepsilon \triangleq y - [\Theta_1 \ \Theta_2] \begin{bmatrix} R \\ E \end{bmatrix}, \quad (2.53)$$

where

$$R \equiv [R_0 \ R_1 \ \dots \ R_{\eta}]^T,$$

$$E \equiv [E_1 \ E_2 \ \dots \ E_{\eta}]^T,$$

$$y \equiv [y(0) \ y(T) \ \dots \ y(NT)]^T,$$

$$\Theta_1 \equiv \begin{bmatrix} u(0) & x_1(0) & \dots & x_{\eta}(0) \\ u(T) & x_1(T) & \dots & x_{\eta}(T) \\ \vdots & \vdots & \ddots & \vdots \\ u(NT) & x_1(NT) & \dots & x_{\eta}(NT) \end{bmatrix}$$

and

$$\Theta_2 \equiv \begin{bmatrix} 1 & 1 & \dots & 1 \\ z_1 & z_2 & \dots & z_{\eta} \\ z_1^2 & z_2^2 & \dots & z_{\eta}^2 \\ \vdots & \vdots & \ddots & \vdots \\ z_1^N & z_1^N & \dots & z_{\eta}^N \end{bmatrix}.$$

Thus, (2.52) can be re-stated as

$$[R, E] = \arg \min_{[R, E] \in \mathbb{R}} \|\varepsilon\|_2. \quad (2.54)$$

Determination of Model Order η There are various methods of determining the optimal order selection η . Using the parsimony principle [42], the best model of a system is defined as the one that accurately represents a transfer function with a minimal number of parameters. Using Prony, the first phase estimates a large number of modes and then performs various reduction procedures to arrive at an optimal order selection. The first phase invariably places many discrete-poles in the 2nd and 3rd quadrants of the complex z-plane. These poles relate to modes that are near the Nyquist rate and often may be discarded.

Once a set of modes is selected the residues are fit to these modes as described above. An ordering algorithm is applied to this set of mode/residue pairs. An ordering scheme proposed in [55] provides a practical approach. The method sequentially compares subsets of mode/residue pairs by a given error criterion. The mode/residue pairs that provide the least amount of error are placed at the top of the order and new subsets are formed. An outline of the ordering algorithm used is as follows.

I. Define:

- (a) Each real eigenvalue and its associated residue constitute a mode/residue pair. For complex conjugate eigenvalues, we use the same terminology, mode/residue pair, to denote a complex conjugate pair of eigenvalues and their associated complex-conjugate residues. Let M denote the number of mode/residue pairs.
- (b) The ordering-criterion is based on the minimization of the relative error between the actual output and the estimated output over combinations of mode/residue pairs.
- (c) The error-criterion is the relative error between the actual output and the estimated output.

II. Let $k=1$.

- (a) Evaluate the relative error for each of the mode/residue pairs according to the error-criterion.
- (b) Order each of these mode/residue pairs according to the ordering-criterion.
- (c) Place the mode/residue pair associated with the minimum in a separate list defined as the sorted-set. The remaining $M-1$ residue/pairs are placed in a second list defined as the remaining-set.
- (d) $k = k + 1$.

III. While $k < M$

- (a) Construct $M - k$ combinations with each combination consisting of the sorted-set and one mode/residue pair of the remaining-set.
- (b) Evaluate the relative error for each of the $M - k$ combinations according to the error-criterion.
- (c) Order each of these combinations according to the ordering-criterion.
- (d) Place the mode/residue pair associated with the combination that gives the minimum in the sorted-set. The remaining $M - k$ mode/residue pairs constitute the remaining-set.
- (e) $k = k + 1$.

IV. end

The final choice of model order is established when the change in error from one pair to the next is insignificant. For more detail on this approach see [55].

A method based upon objective entropy or information measures can be used to optimally decide on the model state order [54]. The Akaike information criterion (AIC) [49] minimizes the Kullback discrimination information [56]. The resulting error criterion includes a penalty on the order number which lends itself well to automated optimal order selection. This approach is well suited for various identification approaches. In regard to Prony identification and model order selection, the reader is referred to [57].

An automated procedure to determine model order is not employed in the examples that follow. Instead, plots of the error in dB vs number of modes are constructed and used to note the point of diminishing return (no significant improvement on error) as the model order increases.

Examples

For our examples the linear system "benchmark" of [58] is used and is shown in Figure 2. Three examples are provided. The first example is noise free. The second example is the same as the first except for the addition of white noise to the measured output.

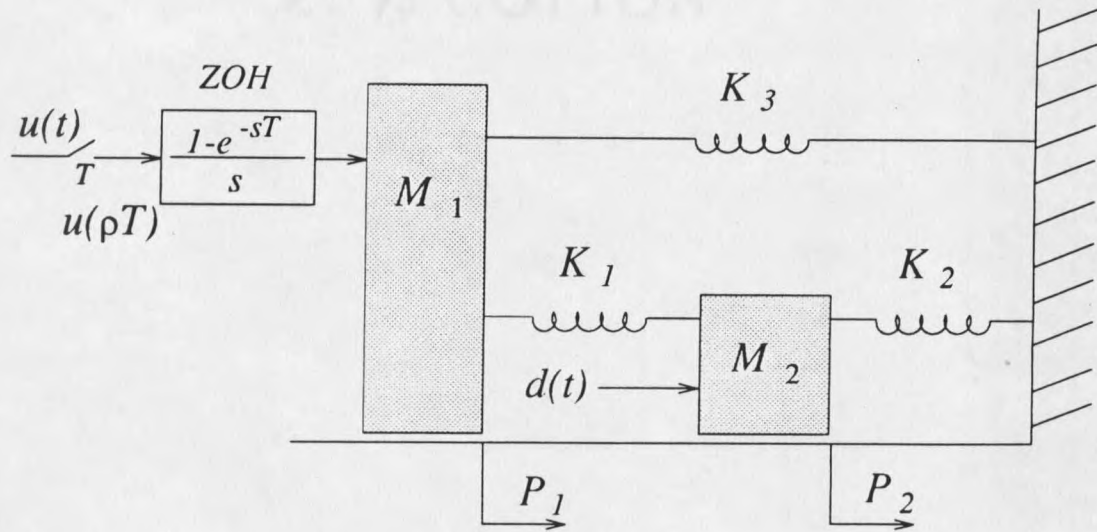


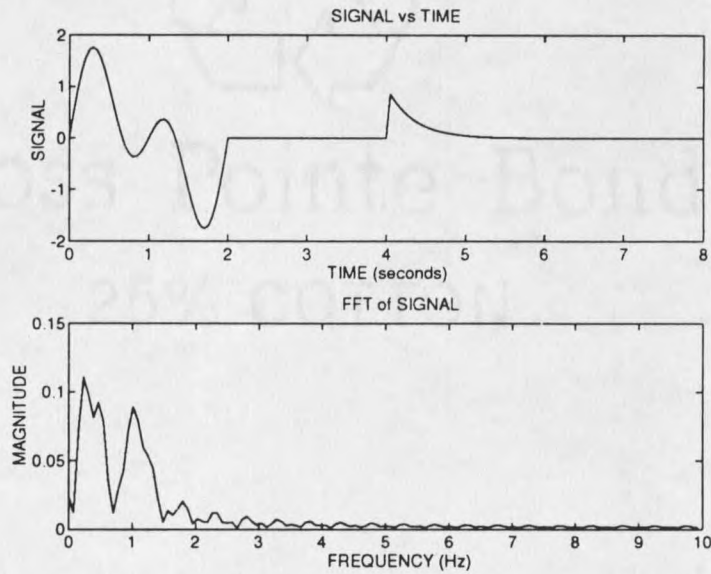
Figure 2. The linear mass/spring diagram for Examples 1-3.

In the third example, both the *discrete Prony* algorithm and an SVA method are applied and compared under conditions of additive white process noise. The system of Figure 2 conforms to that of Figure 1 for specified values of system parameters and initial conditions, as are listed in Table 1. The input drive signal for Examples 1 and 2, shown in Figure 3, is described in Table 2. The input drive signal was chosen in somewhat of an arbitrary fashion to show the variety of input drive signals the *discrete Prony* method can handle. The system is known to have modal content below the 3 Hz range, and the frequency content of the drive signal, as shown in Figure 3, is adequate for Examples 1 and 2. The system output $y(t)$ in all examples is $P_1(t)$ of Figure 2. In Examples 1 and 2, $d(t) = 0$ in Figure 2.

Example 1: No Noise Case For the first example white noise was not added to the measured output. An initial model order η of 30 was chosen prior to applying the Prony identification technique. After the first phase of Prony was complete, 14 of the 30 modes found were in the 1st or 4th quadrant of the z -plane.

j	s_j	r_j	e_j
1	-31.400	-0.9863	0.3000
2	-0.2500	0.0031	0.1000
3,4	-0.0500 $\pm 5.7276i$	0.3034 $\mp 0.0392i$	0.2000 $\mp 0.1000i$
5,6	-0.0500 $\pm 3.4913i$	0.1882 $\mp 0.0047i$	0.0500 $\mp 0.2000i$

Table 1. Actual system model data for Examples 1-3.

Figure 3. Probing input $u(t)$ and its Fourier transform equivalent for Examples 1-2.

$\tau_1 = 2$ seconds $m_1 = 4$ modes			$\tau_2 = 4$ seconds $m_2 = 0$ modes			$\tau_3 = 6$ seconds $m_3 = 1$ mode		
l	$\mu_{1,l}$	$c_{1,l}$	l	$\mu_{2,l}$	$c_{2,l}$	l	$\mu_{3,l}$	$c_{3,l}$
1	$3.1416i$	$-0.5000i$				1	-3.0000	+1.0000
2	$-3.1416i$	$+0.5000i$						
3	$6.2832i$	$-0.5000i$						
4	$-6.2832i$	$+0.5000i$						

Table 2. Input drive data for Examples 1-2.

j	ξ_j	R_j	E_j	error(dB)
1,2	$-5.0000e-02$ $\mp 5.7276e+00i$	$3.0338e-01$ $\pm 3.9152e-02i$	$2.0000e-01$ $\pm 1.0000e-01i$	-3.7748e+00
3,4	$-5.0000e-02$ $\mp 3.4914e+00i$	$1.8817e-01$ $\pm 4.7146e-03i$	$5.0000e-02$ $\pm 2.0000e-01i$	-2.0229e+01
5	$-2.5000e-01$	$3.1427e-03$	$1.0000e-01$	-2.6472e+01
6	$-3.1400e+01$	$-9.8626e-01$	$3.0000e-01$	-1.9910e+02
7,8	$-8.6376e-01$ $\mp 1.3699e+01i$	$-4.5645e-11$ $\pm 2.0300e-10i$	$-1.1734e-11$ $\pm 1.0299e-11$	-2.0103e+02
9,10	$-1.0742e+00$ $\mp 1.8757e+01i$	$-1.5061e-11$ $\pm 2.8731e-10i$	$-7.7889e-12$ $\pm 6.8718e-12i$	-2.0292e+02
11,12	$-1.2172e+00$ $\mp 2.3587e+01i$	$1.4187e-10$ $\pm 2.8776e-10i$	$-5.4991e-12$ $\pm 5.3120e-12i$	-2.0400e+02
13,14	$-1.3234e+00$ $\mp 2.8317e+01i$	$2.7987e-10$ $\pm 1.8675e-10i$	$-3.6150e-12$ $\pm 4.8013e-12i$	-2.0422e+02

Table 3. Data from Prony algorithm for Example 1.

The second phase of Prony fit system and initial condition residues to these modes. An ordering algorithm based on [55] was applied, and the results are given in Table 3.

A plot of the error in dB (column 5 of Table 3) is shown in Figure 4. From this graph a model order of 6 was selected. Figure 5 shows both actual and model output data when the input probing signal described by Table 2 is applied. There is negligible difference in the actual and estimated outputs since there is no noise added to the output of the system.

Example 2: Noisy Case For the second example white-Gaussian noise was added to the measured output. The signal-to-noise ratio of the output signal was 21.5 dB. An initial model order η of 30 was chosen prior to applying the Prony identification technique. After the first phase of Prony was complete, 15 of the 30 modes found were in the 1st or 4th quadrant of the z -plane. The second phase of Prony fit system and initial condition residues to these 15 modes.

A plot of the error in dB similar to Figure 4 is shown in Figure 6. From this

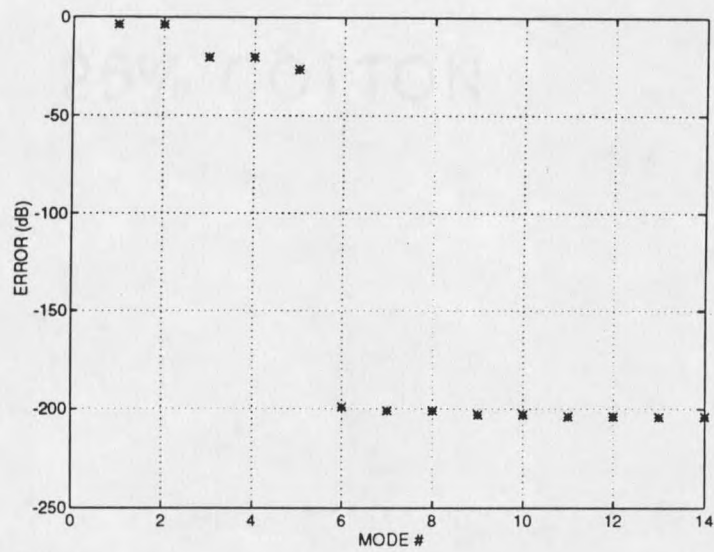


Figure 4. Error in dB for each additional mode set for Example 1.

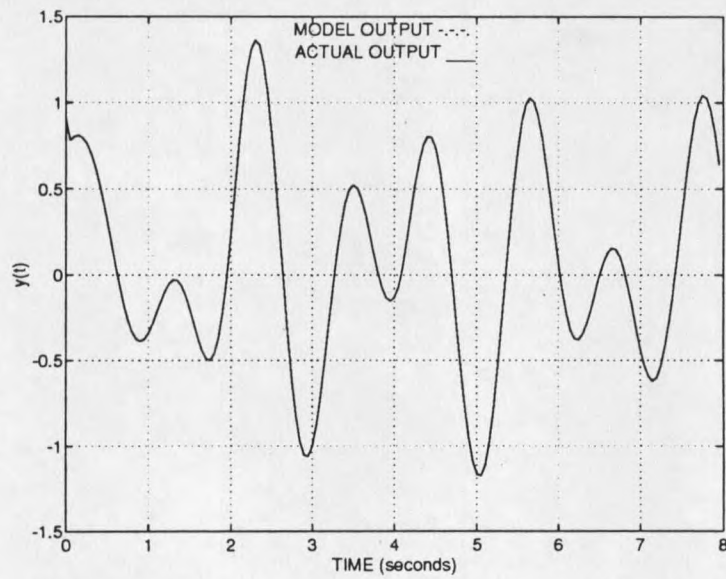


Figure 5. The output for both actual, $y(t)$ and model, $\hat{y}(t)$ for Example 1.

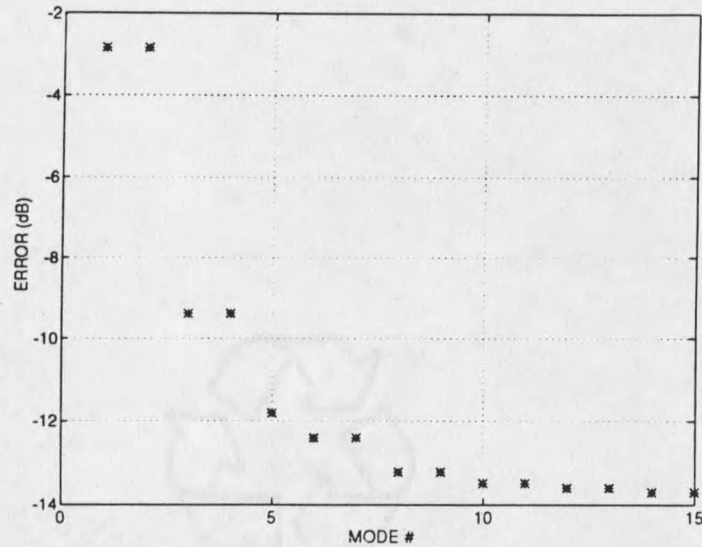


Figure 6. Error in dB for each additional mode set for Example 2.

graph a model order of 7 was selected due to the diminishing return in error with increased model order. Figure 7 shows both actual and model output data when the input probing signal described by Table 2 is applied. The error between the actual and estimated signal is -12.4 dB. Figure 8 shows both actual and model output data without additive noise at the output when the input probing signal described by Table 2 is applied. A plot of the s-domain poles of the actual system and model can be seen in Figure 9. Note that the dominant modes (the less damped modes) have been well identified which is an important feature from a dynamical control point of view. Depending on the type of control being used for system stabilization this model may or may not be adequate.

Example 3: SVA vs. Discrete Prony For a third example we compare *discrete Prony* identification results to those of the SVA method described at the beginning of this chapter which is based on the paper by Moonen, et al., [51]. This

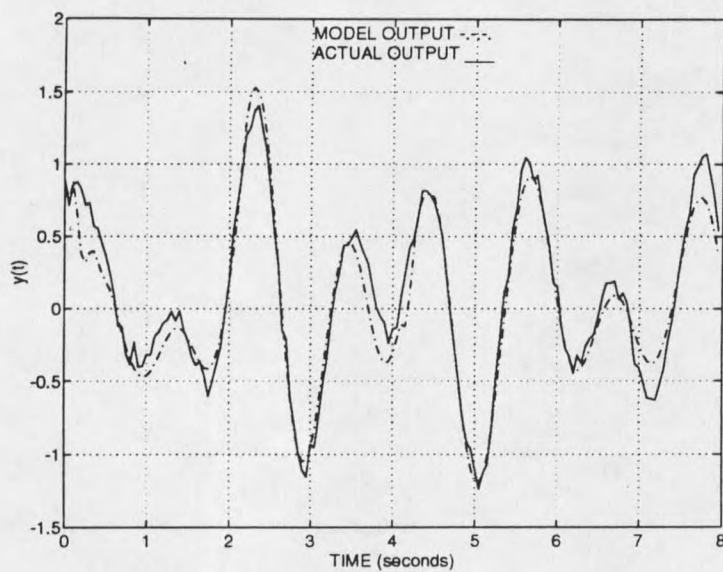


Figure 7. The output for both actual, $y(t)$ and model, $\hat{y}(t)$ for Example 2 with additive output noise.

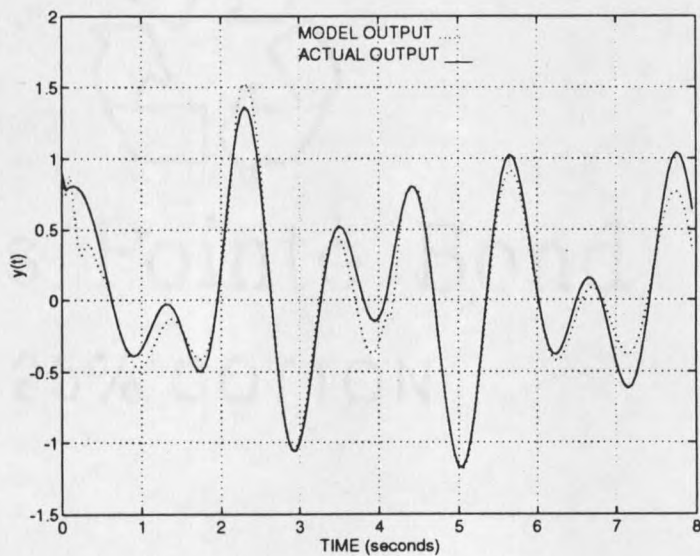


Figure 8. The output for both actual, $y(t)$ and model, $\hat{y}(t)$ for Example 2 with additive output noise removed.

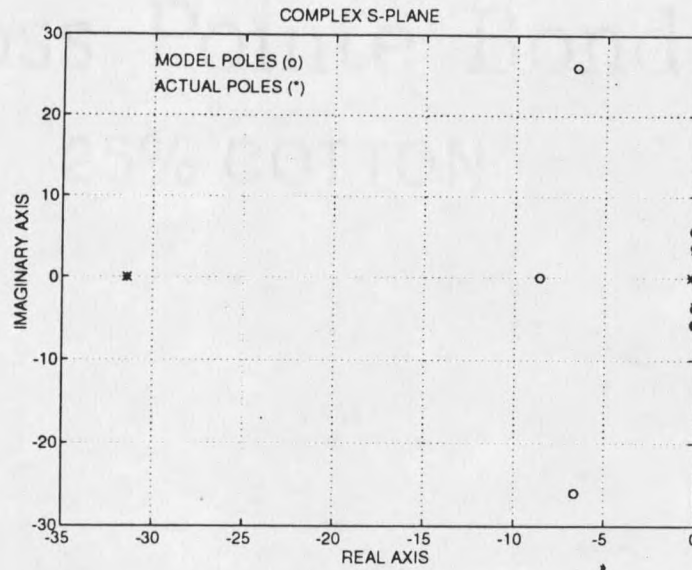


Figure 9. The poles of the s -domain for the actual system and the model for Example 2.

method is referred to as the *Moonen* algorithm. The *Moonen* algorithm was chosen because it parallels the method of *discrete Prony*. The *Moonen* algorithm estimates the states of the system over a time period, whereas Prony estimates the modes in the first step and then builds the states of the system prior to fitting the residues to the modes.

A different input probing signal was used in this example for the *discrete Prony* method compared to the previous examples. Here we chose a combination of 14 sinusoidal waves that span the bandwidth of interest, namely, the frequency range from 0 to 3 Hz. The input signal was on for 1 second and then it remained off. The parameter values for (2.27) are given as below and in Table 4:

- $q = 1$ interval
- $m_1 = 14$ input eigenvalues
- $\tau_1 = 1$ second

	$l = 1, 2$	$l = 3, 4$	$l = 5, 6$	$l = 7, 8$	$l = 9, 10$	$l = 11, 12$	$l = 13, 14$
$\mu_{1,l}/(2\pi)$	$\pm 0.25i$	$\pm 0.75i$	$\pm 1.00i$	$\pm 1.25i$	$\pm 1.75i$	$\pm 2i$	$\pm 2.5i$
$c_{1,l}$	$\mp 0.07i$	$\mp 0.07i$	$\mp 0.07i$	$\mp 0.14i$	$\mp 0.14i$	$\mp 0.07i$	$\mp 0.07i$

Table 4. Input drive data for *discrete Prony* of Example 3.

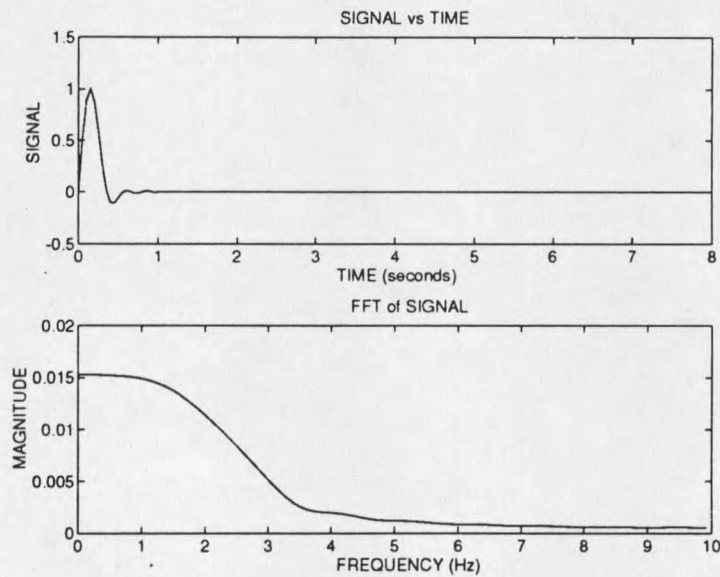


Figure 10. Probing input $u(t)$ and its Fourier transform equivalent for Example 3 using the *discrete Prony* algorithm.

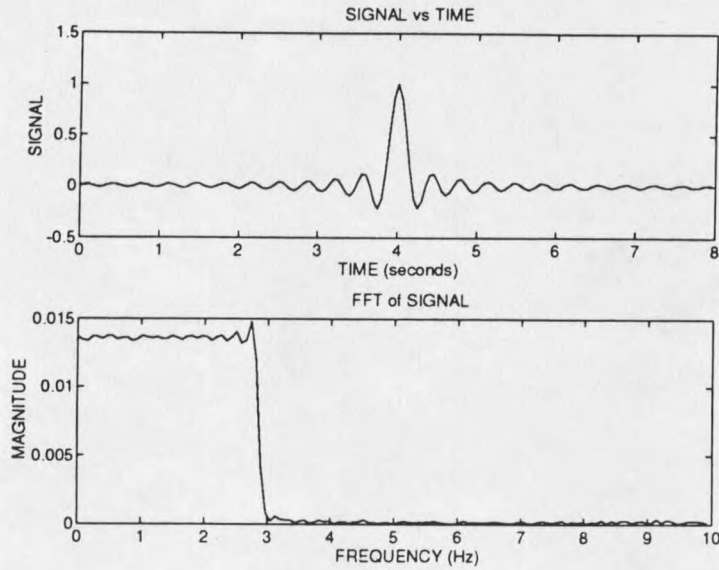


Figure 11. Probing input $u(t)$ and its Fourier transform equivalent for Example 3 using the *Moonen* algorithm.

For the *Moonen* method authors of [51] suggest a pseudo-random input drive signal that sufficiently excites all the modes of the system. Additionally, the signal is assumed to be on for the entire record length. Here we chose to use a SINC type input of (2.55) with energy content that is close to the input used in the *discrete Prony* method (compare the Figures of 10 and 11).

$$u_{\text{sinc}}(t) = m \frac{\sin(2\pi f(t - \Delta t))}{2\pi f(t - \Delta t)} \quad (2.55)$$

where $m = 1$, $f = 18$, and $\Delta t = 4$. Figure 11 shows the SINC type input used. To insure there was no rank deficiency in the Hankel matrix of (2.24) a small amount of normally distributed, zero mean noise was added to the SINC signal of Figure 11.

The *Moonen* algorithm was run for the same case as in Example 1, and the results were very similar to the *discrete Prony* algorithm. Note that there is no user interaction in deciding the model order as can be seen by Figure 12. The algorithm, after computing the rank of the Hankel matrix of (2.24), found the system to be

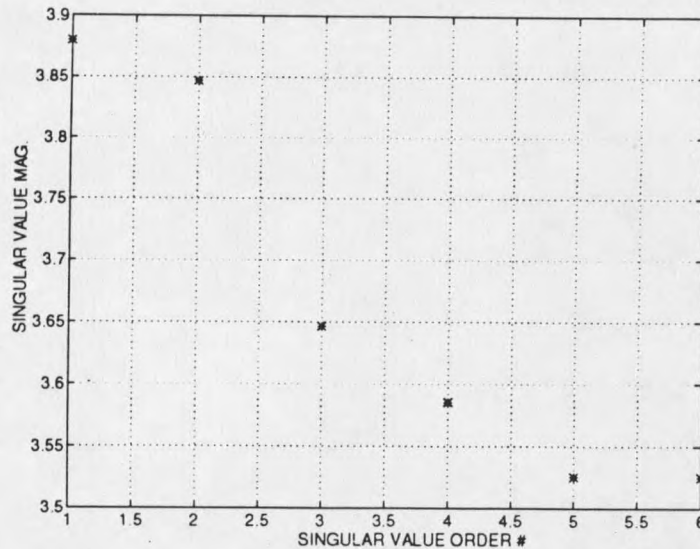


Figure 12. The singular values of the Hankel matrix for Example 3 using the *Moonen* algorithm, no noise case.

of order 6. However, once noise is added to the system, the order selection is not determined by simply computing the rank of this Hankel matrix.

With the input probing signal established for both algorithms the system was subjected to increasing amounts of system noise at a second input, $d(t)$, shown in Figure 2. The noise was normally distributed with zero mean. The variance of the noise input was adjusted to obtain several signal-to-noise ratios at the measured output; these values are given in column one of Table 5. Since the noise is entering at $d(t)$, the effect of the noise on the output is colored due to the dynamics of the system. Note the plot of singular values of the Hankel matrix for the *Moonen* algorithm associated with an SNR of 50 dB in Figure 13. The algorithm shows that the order selected is 23. However, through observation, it appears that the model order may be chosen as 6, which is at the first bend of singular values. This graph of singular values is typical for each of the cases shown in Table 5. From this table it is clear

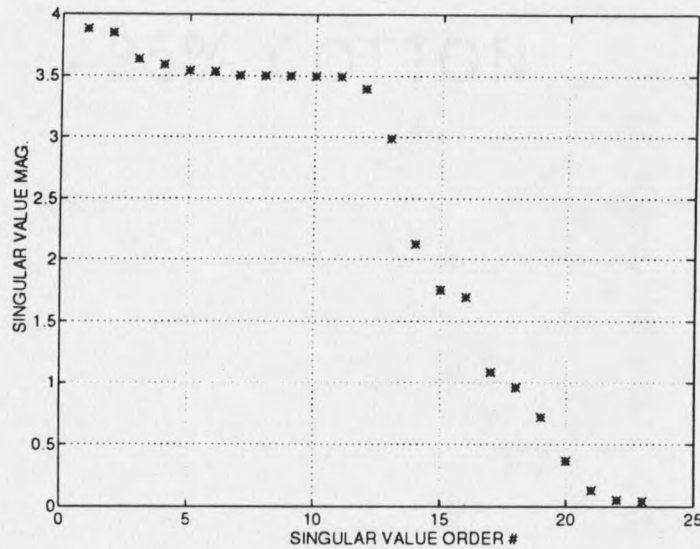


Figure 13. The singular values of the Hankel matrix for Example 3 using the *Moonen* algorithm with 50dB output SNR.

that each method performs relatively the same, where the *discrete Prony* method only does better in the worst noise case. The *Moonen* method has less computation and has less in the way of user interaction.

Noise at Output	<i>Moonen</i> Method		<i>discrete Prony</i> Method	
SNR (dB)	order	error (dB)	order	error (dB)
∞	6	-260.00	6	-199.00
50	6	-55.64	6	-30.85
30	6	-28.73	6	-27.87
20	6	-22.12	6	-21.38
10	6	-6.89	6	-10.85

Table 5. Example 3 error data for noise applied at $d(t)$.

Example 4: One Machine/Infinite Bus Power System As a final example, we consider a power system consisting of one machine, a transmission line and an infinite bus (refer to Figure 14). The infinite bus is an approximation to a large

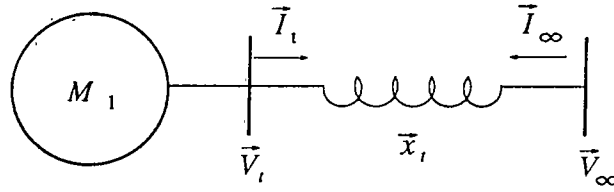


Figure 14. One-machine/infinite-bus power system diagram.

power grid connected to the one machine through a transmission line represented by $\bar{x}_t$. The per unit value of the infinite bus voltage is $1\angle 0^\circ$ and the per unit value of the injected current of the infinite bus is the negative of $\bar{I}_t$. In this example we use *discrete Prony* identification to obtain a low order model of the system for control purposes. The ultimate goal is to control speed deviation of the machine following a transmission line fault. This is done by affecting the field voltage of the synchronous machine.

Assuming the system remains in quasi steady-state we are able to write a set of algebraic equations describing the machine bus voltage and the current injected into the transmission line. A set of ordinary differential equations (ODE's) is used to describe the dynamics of the synchronous machine. The algebraic and differential equations are correlated through the so-called D-Q (Direct-Quadrature) axis relations. More detail on these equations is provided in Appendix A. The order of the set of ODE's is 4. The sampling period of the system is set at $T = 0.05$ seconds. The input probing signal for *discrete Prony* identification was a single pulse of amplitude 1 and duration 0.05 seconds and was applied at the reference input to the exciter. The exciter unit is a first-order low-pass filter with gain of 10 and a time-constant of 0.04 seconds. The exciter output is applied to the field voltage input, E_f , of the synchronous machine. The input to the power system stabilizer (PSS) unit was taken as the the speed deviation of the machine. A model of the system using *discrete Prony*

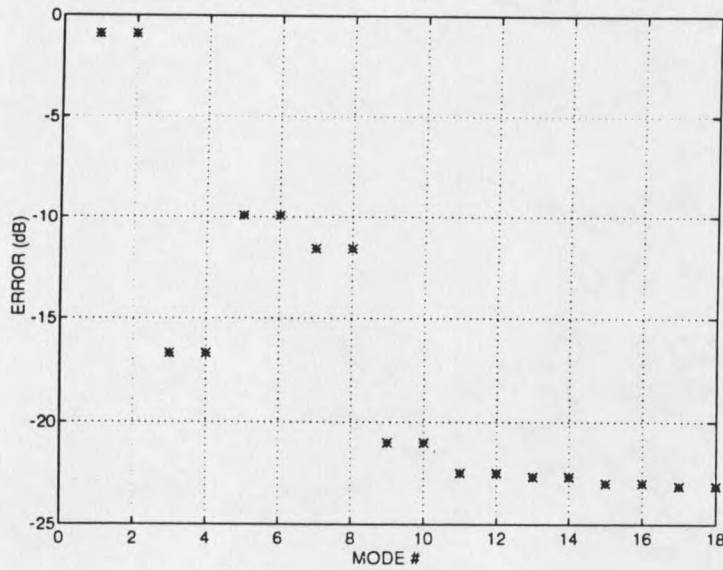


Figure 15. Residue ordering for example 4.

Eigenvalues		Residues	
-1.4056e+00	-1.7651e+00i	-2.3091e+00	+1.8099e+00i
-1.4056e+00	+1.7651e+00i	-2.3091e+00	-1.8099e+00i
-2.9920e-01	+7.2809e+00i	1.8320e+00	+1.1226e+00i
-2.9920e-01	-7.2809e+00i	1.8320e+00	-1.1226e+00i

Table 6. Final model for Example 4.

used the single pulse described above and the speed deviation of the machine as the input/output data. In the first phase of *discrete Prony* identification, 21 eigenvalues were identified. Of these 21, 18 were used in the second stage for residue fitting. A plot of the ordering by the residue method is shown in Figure 15. A model order of 4 was chosen (see Table 6 for the discrete model eigenvalues and residues) with an error in fitted to actual data of -16.7 dB. Using the 4th-order model of Table 6, an LQ gain of

$$K = -[4.8163 \ 2.7961 \ 0.64204 \ -1.4312]$$

was computed for the PSS unit. For state-estimation, a Kalman observer was used (details on the Kalman observer equations are given in the section containing the servo example of Chapter 4). For a detailed diagram of the control scheme, see Figure 16. The disturbance on the transmission line reactance was a change from

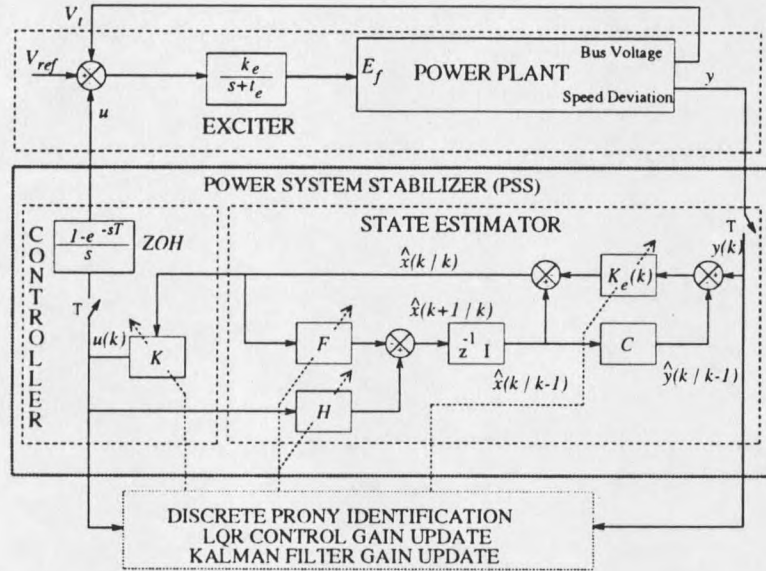


Figure 16. Detailed diagram of control scheme for Example 4.

$\vec{x}_t = j0.2 \text{ p.u.}$ to $\vec{x}_t = j0.35 \text{ p.u.}$ from time $t = 1$ second to $t = 2$ seconds. The results of two simulations are given in Figure 17 where the output signal $y(t)$ is the speed deviation of the machine. The first simulation is for the open-loop case, whereas the second simulation is for the LQG state-feedback case. The value for the LQG input weighting scalar R was chosen to be 10. As shown in Figure 17, the LQG approach using the *discrete Prony* model does improve the damping of the transient condition.

Discussion of Results

A review of various identification methods is given at the start of the chapter. The methods reviewed are based on autoregressive and state-space models. Characteristics

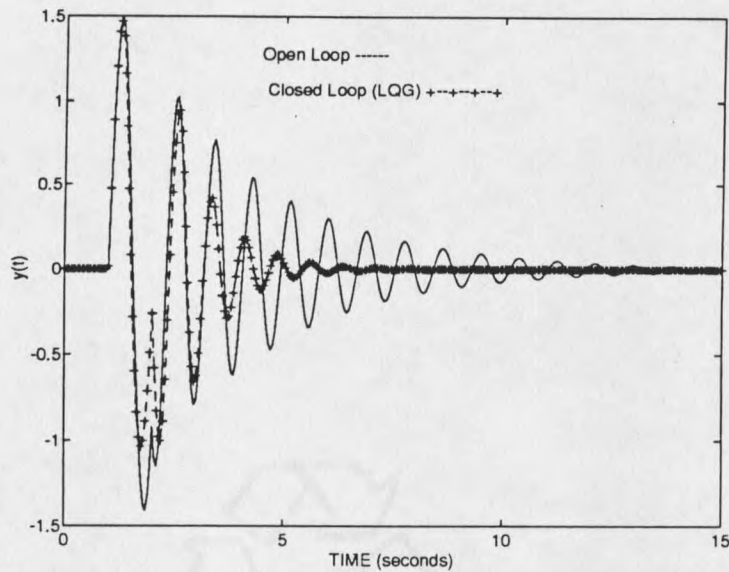


Figure 17. Transient simulation of example 4: Open loop vs. LQG.

of each method were discussed to establish a baseline for the new method developed in this chapter. The method of this chapter is classified as a Prony method because it has two main phases, paralleling the original Prony signal analysis method, in which eigenvalue estimates are generated in the first phase, and residue estimates are calculated in the second phase. Both phases involve linear least-squares solutions that can be obtained with high accuracy using singular value decomposition techniques. The detailed steps used in the two Prony phases of this chapter differ considerably from those of previous Prony-based methods because of the sampled-and-held nature of the input signal. An interesting feature of including the sample-and-hold on the input is that the resulting second Prony phase is actually simplified from the corresponding second Prony phase of [20].

The *discrete Prony* method is for single input/single output systems. The AR methods have this same limitation as opposed to the state-space methods which can handle the multi-input/multi-output case. One feature that is not dealt with in the

AR method and some of the state-space methods is that of establishing an estimate of initial conditions. The state-space method based on SVA does allow for the estimation of initial states of the system. The striking difference of the Prony method from those reviewed is the fact that modes of the system are estimated first. The user may be able to use prior knowledge of the system modes to eliminate extraneous modes from the set. This feature allows for reduced numerical calculations and potential for more accurate residue fitting. The final feature of the *discrete Prony* method is the use of a generalized class of input probing signals that are sampled and held. Prior to being sampled and held, the input signal is piecewise continuous and is characterized by input eigenvalues and input residues between points of discontinuity. The sample period T is the same as that used to collect the sampled system input and output data, which lends itself well to digitally controlled systems. By appropriate selection of the input signal, its energy can be allocated to particular frequency ranges of interest with regard to transfer function identification. Most of the other methods utilize a pseudo-random input signal that essentially has much of its energy at frequency ranges that are not of interest. This type of signal will often not adequately excite those modes that are of interest.

The examples show that with the addition of measured output noise, the approach developed in this chapter can generate fairly good estimates of initial condition residues, transfer function residues, and system eigenvalues. Example 3 shows that the *discrete Prony* method performs relatively well under colored noise environments as compared with other methods. While the effects of system noise on the method presented here remains to be analytically examined, there is a foundation of previous work (e.g., [59, 7, 10, 11, 12, 60, 13]) to build on.

The last example demonstrated the *discrete Prony* identification use in power system stabilization (damping). A low-order model was obtained and used for LQG

state-feedback control. The LQG control based on the *discrete Prony* identified model provided significant damping. In practice, it was found that higher-order models obtained from *discrete Prony* identification were more difficult to utilize in this non-adaptive LQG state-feedback configuration.

CHAPTER 3

SET-MEMBERSHIP APPROACH TO CONTROL DESIGN

In the previous chapter we developed an identification method for digitally controlled systems. Traditionally, the problem of system identification and control design is formulated in an entirely stochastic framework in which the system and its parameters are described by stochastic models. Although the development in Chapter 2 was not formulated in a stochastic framework, a salient feature is that the control input can be easily constructed such that specific frequency ranges of interest are excited, which gives it the ability to overcome (compensate for) the effects of noise in the system. In the ideal case, it is assumed that there exist parameters that exactly describe the true system, which is essentially what many identification schemes assume. This is the case for the *discrete Prony* method, and even though the type of input used can compensate for effects of noise on the estimated model there will always be uncertainty in the parameters estimated.

Once a model is determined one then tries to find a controller which minimizes the expected value of a loss function. This is known as the dual-control problem. A heuristic approach to solving this problem is the traditional adaptive control system in which a model with unknown parameters is assumed for the system. Parameter estimates are obtained recursively in real time, and the controller is designed as if the parameter estimates were in fact the correct parameters for describing the plant. This is known as the *certainty equivalence principle*. Even in the ideal case, the transient errors between the identified model and the true system can be large enough to lead to significant reduction in system performance.

To overcome these shortcomings, a *set-membership approach* to control design is suggested in [36]. The idea of set-membership was first introduced by Fogel [61]. The concept is based on the idea of first estimating a parameter set which, in the ideal case, contains the true parameters of the system. Let this membership set be represented by $\mathcal{M}$. A control scheme based on this model set is then proposed. Note, the control scheme assumes that each plant within the model set as described by the parameter set estimate is equally likely to be the true plant. Therefore, the controller, if it exists, must stabilize all plants within the membership set $\mathcal{M}$ and as such is considered *robust*. A diagram of such an identification/control scheme is depicted in Figure 18.

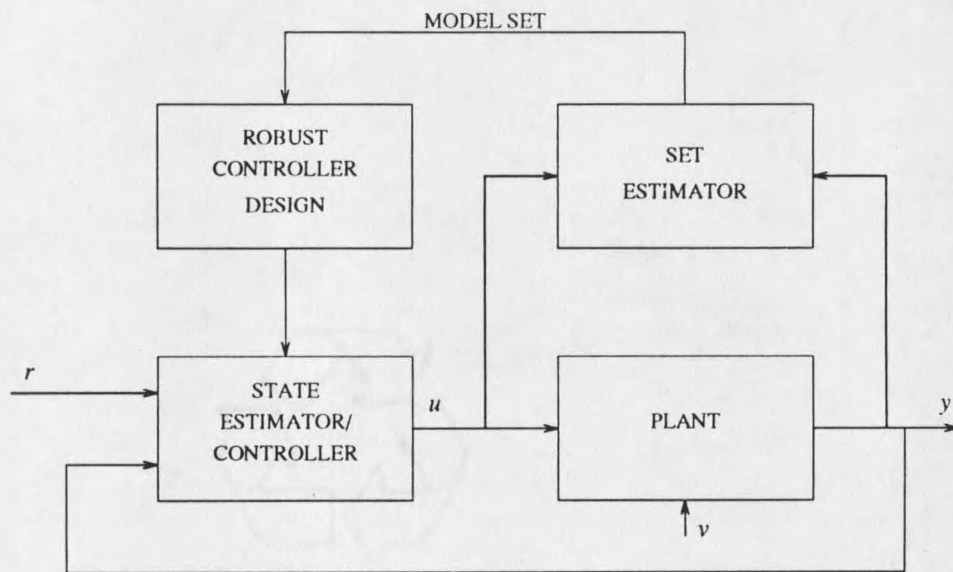


Figure 18. Set-membership adaptive control scheme diagram.

Thus, unlike traditional adaptive control schemes, where parameters are estimated from input/output data and assumed to be equivalent to the true parameters, the set-membership approach, in a sense, selects the worst-case set of parameters from the membership set for the design of the optimal control.

Before describing any particular control strategy for the set-membership ap-

proach we will describe some identification methods that produce membership sets.

Ellipsoid Set Estimation

Two methods of set membership identification are described, each producing a set of real-valued parameters of dimension n that are described by a bounded hyper-ellipsoid in the space $\mathbb{R}^{n \times n}$. The first method of identification assumes no models of uncertainty and arrives at an ellipsoid set which is directly affected by an assumed known bound on the disturbance/noise that acts on the system. The second method of identification assumes a model of non-parametric uncertainty in the model of the system and uses an identification scheme that produces an ellipsoid set of parameters. As expected, this set is directly affected by the model of uncertainty assumed for the system.

Before continuing, consider the following definition of an ellipsoid set.

$$\Omega = \{ \theta : (\theta - \theta_0)^T \Gamma (\theta - \theta_0) \leq 1, \Gamma = \Gamma^T > 0 \}.$$

Geometrically, this ellipsoid set has its center at θ_0 , and the matrix Γ gives its size and orientation; i.e., the square roots of the reciprocals of the eigenvalues of Γ are the lengths of the semi-axes, and the eigenvectors of Γ give their direction.

Disturbance/Noise Induced Ellipsoid

Cheung, et al., [62], have developed what they refer to as the *Optimal Volume Ellipsoid* algorithm (OVE). The algorithm is recursive and is derived for set estimation of a SISO linear time-invariant system with bounded noise. Cast in a recursive framework, where a minimal volume ellipsoid results at each recursion, the algorithm extends a result due to Khachian [63] in which a technique was developed to solve a class of linear programming problems (see [64] for extensions of Khachian's results).

Consider the the SISO ARX model

$$\begin{aligned} y(k) &= -\sum_{i=1}^{n_a} a_i y(k-i) + \sum_{j=1}^{n_b} b_j u(k-j) + \nu(k) \\ &= \theta^T \phi(k) + \nu(k) \end{aligned} \quad (3.1)$$

where $\theta^T = [a_1, \dots, a_{n_a}, b_1, \dots, b_{n_b}]$ is the parameter vector to be estimated; the regression vector containing the past input and output values is given by $\phi(k) = [-y(k-1), \dots, -y(k-n_a), u(k-1), \dots, u(k-n_b)]^T$. The term $\nu(k)$ is a sequence of bounded disturbances/noise corrupting the system output with $|\nu(k)| \leq \gamma$ for all $k \geq 0$. It is assumed that the number of parameters is known a priori as well as the bound on the disturbance; thus we are given n_a , n_b , and γ . The total number of parameters is $n = n_a + n_b$.

Let $\mathcal{F} \in \mathbb{R}^n$ be a set such that all $\theta \in \mathcal{F}$ are feasible parameter estimates of the plant which are consistent with the measurements. That is

$$\mathcal{F} = \{\theta : |y(k) - \theta^T \phi(k)| \leq \gamma, \quad k = 0, \dots, N\}. \quad (3.2)$$

The membership $\mathcal{M}$ is defined by (3.2). The problem of parameter set estimation is to find $\mathcal{F}$ explicitly in the parameter space. In general, $\mathcal{F}$ is an irregular convex set; therefore, the algorithm finds the smallest ellipsoid that contains the set $\mathcal{F}$ where the hyper-volume of an ellipsoid is used to measure "smallness". Consistency of measurement provides a pair of constraints which establishes the framework for a minimal volume ellipsoid. The first constraint comes directly from (3.2) and the second is in terms of the predicted set $\mathcal{F}_{k+1}$ which are given by

$$\begin{aligned} \text{constraint1} &: & |y(k) - \theta^T \phi(k)| &\leq \gamma \\ \text{constraint2} &: & \mathcal{F}_{k+1} &= \{\theta : |y(k+1) - \theta^T \phi(k+1)| \leq \gamma\}. \end{aligned}$$

Geometrically, $\mathcal{F}_{k+1}$ is the region between two parallel hyper-planes defined by *constraint 2*. The set estimation problem is then stated as: Given an ellipsoid Ω_k , find

another ellipsoid Ω_{k+1} with minimal volume, such that Ω_{k+1} contains $\Omega_k \cap \mathcal{F}_{k+1}$, for $k = 0, \dots, N$, where N is the number of data records. Mathematically, the optimization problem becomes

$$\min\{\text{vol}(\Omega_{k+1}) : \Omega_{k+1} \supset \Omega_k \cap \mathcal{F}_{k+1}\}.$$

The authors of [62] define Ω_k and Ω_{k+1} as

$$\Omega_k \triangleq \{\theta : (\theta - \theta_k)^T \Gamma_k (\theta - \theta_k) \leq 1; \theta \in \mathbb{R}^n\} \quad (3.3)$$

and

$$\Omega_{k+1} \triangleq \{\theta : (\theta - \theta_{k+1})^T \Gamma_{k+1} (\theta - \theta_{k+1}) \leq 1; \theta \in \mathbb{R}^n\},$$

where Γ is symmetric and positive definite and θ_k is the center estimate of the ellipsoid at time k . The algorithm amounts to recursively estimating θ_k and Γ_k at each time instant k . The algorithm is initialized by assuming that θ_0 and Γ_0 are such that the ellipsoid Ω_0 includes the true parameters of the system. This is accomplished by selecting the eigenvalues of Γ_0 such that Ω_0 is sufficiently large. For details and attributes of the OVE algorithm see [62] and references therein.

Non-parametric Uncertainty Induced Ellipsoid

The method of identification outlined below is found in [36]. The model set $\mathcal{M}$ is defined as follows:

$$\mathcal{M} \triangleq \{y = Gu : G \in \mathcal{G}\} \quad (3.4)$$

and

$$\mathcal{G} \triangleq \{G_\theta(1 + \Delta_G W_G) : \theta \in \Omega_{\text{prior}}, \|\Delta_G\|_{H_\infty} \leq 1\}, \quad (3.5)$$

where $G_\theta(z)$ is a parametric z-transform transfer function with parameters $\theta \in \Omega_{\text{prior}}$, referred to as the prior parameter set. The system $\Delta_G W_G$ is referred to as the multiplicative non-parametric uncertainty, which is a dynamic uncertainty described

by an uncertain but unity bounded stable transfer function $\Delta_G(z)$ and a known stable transfer function $W_G(z)$. Having knowledge of W_G is precisely the assumption made in robust control design, e.g., [28]. The H_∞ norm for z -transform functions is defined as

$$\|G\|_{H_\infty} \triangleq \sup_{|\omega| \leq \pi} |G(e^{j\omega})|.$$

Solving for Δ_G in (3.5) in terms of G and θ , we get

$$\Delta_G(z) = \frac{G(z) - G_\theta(z)}{W_G(z)G_\theta(z)}.$$

We define

$$\Omega^* \triangleq \left\{ \theta : \left\| \frac{G(z) - G_\theta(z)}{W_G(z)G_\theta(z)} \right\|_{H_\infty} \leq 1 \right\}$$

and refer to Ω^* as the true-plant set because it does not depend on the data set but rather on the true but unknown system G . Since the decomposition of G into G_θ and Δ_G is not unique, it is not possible to consider a "true" parameter value for the plant. Therefore, we want to find a set estimate,

$$\hat{\Omega} \triangleq \hat{\Omega}^* \cap \Theta_{prior},$$

where $\hat{\Omega}^*$ is an estimate of Ω^* . As a result, $\hat{\Omega}$, is a set estimate of *all* possible parameter values consistent with the assumption that the true system G is in the model set $\mathcal{G}$.

We further characterize the parametric transfer function $G_\theta(z)$ by using the standard ARX form of (3.1).

Before we state a main result of [36] we must define several terms. Let $\mathcal{E}_k(\cdot)$ be defined as the sample-mean operator,

$$\mathcal{E}_k(x) \triangleq \frac{1}{k} \sum_{j=1}^k x(j), \quad (3.6)$$

and $\|x\|_{k2}$ be defined as the truncated l_2 -norm of a sequence,

$$\|x\|_{k2} \triangleq \left(\sum_{j=1}^k x(j)^2 \right)^{\frac{1}{2}}.$$

Hence,

$$\frac{1}{k} \|x\|_{k2}^2 = \frac{1}{k} \sum_{j=1}^k x(j)^2 = \mathcal{E}_k(xx)$$

where x is an scalar value. Next, we define the following regression vectors:

$$\begin{aligned}\phi_y &\triangleq [-z^{-1}y \ \cdots \ -z^{-n_a}y]^T, \\ \phi_u &\triangleq [-z^{-1}u \ \cdots \ -z^{-n_b}u]^T, \\ \phi &\triangleq \begin{bmatrix} \phi_y \\ \phi_u \end{bmatrix}, \\ \alpha_k &\triangleq \mathcal{E}_k(yy), \\ \beta_k &\triangleq \mathcal{E}_k(\phi y), \\ \Gamma_k &\triangleq \mathcal{E}_k(\phi\phi^T) - \begin{bmatrix} 0 & 0 \\ 0 & \mathcal{E}_k((W_G\phi_u)(W_G\phi_u)^T) \end{bmatrix},\end{aligned}$$

where $\alpha_k \in \mathbb{R}$, $\beta_k \in \mathbb{R}^\eta$, and $\Gamma_k \in \mathbb{R}^{\eta \times \eta}$, with $\eta = n_a + n_b$. By the definition of $\mathcal{E}_k$ given in (3.6) we see that $\mathcal{E}_k(yy)$ is computed by first performing the multiplication out term-by-term before taking the sample mean. To compute $\mathcal{E}_k(\phi\phi^T)$, multiply ϕ and ϕ^T to produce a matrix of sequences, then take the sample mean of each element. Finally, by letting $G_\theta(z) = B_\theta(z)/A_\theta(z)$ where

$$\begin{aligned}B_\theta(z) &= b_1 z^{-1} + \cdots + b_{n_b} z^{-n_b}, \\ A_\theta(z) &= 1 + a_1 z^{-1} + \cdots + a_{n_a} z^{-n_a}\end{aligned}$$

we can state the primary result of Lau's and give the solution for estimation of Ω^* .

Theorem 3.1 (Lau [36]) *Suppose the measured data $\{y, u : j = 1, \dots, N\}$ is generated from $y = Gu$ as defined in (3.4). Then the following holds:*

$$\Omega^* \subseteq \Omega[N] \subseteq \Omega_k, \ \forall k \in [1, N], \ \forall N \in \mathbb{N}$$

where $\Omega[N]$ and Ω_k are given by,

$$\begin{aligned}\Omega_k &\triangleq \{\theta : \|A_\theta y - B_\theta u\|_{k2} \leq \|W_G B_\theta u\|_{k2}\} \\ \Omega[N] &\triangleq \bigcap_{k=1}^N \Omega_k.\end{aligned}$$

An estimate of the parameters can now be obtained from Ω_k since Ω^* is a subset of Ω_k . Lau goes on to show that:

I. Ω_k can be expressed in quadratic form as

$$\Omega_k = \{ \theta : \theta^T \Gamma_k \theta - 2\beta_k^T \theta + \alpha_k \leq 0 \}.$$

II. If Γ_k^{-1} exists, then

$$\Omega_k = \{ \theta : (\theta - \hat{\theta}_k)^T \Gamma_k (\theta - \hat{\theta}_k) \leq V_k \}, \quad (3.7)$$

where

$$\hat{\theta}_k = \Gamma_k^{-1} \beta_k$$

$$V_k = \beta_k^T \Gamma_k^{-1} \beta_k - \alpha_k.$$

Assuming V_k is nonzero we can divide through the inequality of (3.7) by V_k and obtain an expression similar to the ellipsoid expression found in (3.3). Note, $\hat{\theta}_k$ is identical to the ordinary least-squares estimate when $W_G = 0$ (no nonparametric dynamics), i.e.,

$$\theta_{LS} = (\mathcal{E}_k(\phi\phi^T))^{-1} \mathcal{E}_k(\phi y).$$

This gives the set estimate Ω_k a nice geometric interpretation because it is centered at the estimate one usually gets by ignoring the nonparametric uncertainty and applying least-squares estimation. The center of the ellipsoid estimate of Cheung does not necessarily have as its center the least-squares estimate. For ellipsoid uncertainty algorithms similar to Cheung that have the least-squares estimate as their center see [65].

Linear Quadratic Regulator

This section includes a tutorial on linear quadratic regulator (LQR) control theory. We first describe the standard cost function that is commonly used in control theory for the continuous time system. We introduce the algebraic Riccati equation and its role in optimal state feedback control. Lastly, the discrete counterpart to the continuous time solution is given. Following this section we combine the ideas here and in the previous sections to state a set-membership control method in the form of a minimaximization problem.

Continuous-Time LQR

A common objective for the feedback regulation problem is to minimize the energy of the output signal over time. This is accomplished by determining a control signal $\mathbf{u}(t)$ that optimally minimizes the integral of $\mathbf{y}(t)^T \mathbf{y}(t)$ over time. The control signal that accomplishes this is referred to as the optimal control and is represented by $\mathbf{u}^*(t)$. However, without any penalty on the magnitude of the control signal $\mathbf{u}(t)$, it is possible that $\mathbf{u}(t)$ may reach amplitudes that are not acceptable in the actual system. To introduce trade-off between magnitudes of $\mathbf{u}(t)$ and $\mathbf{y}(t)$ we place weights on both $\mathbf{u}(t)$ and $\mathbf{y}(t)$ and construct what is commonly referred to as the LQR cost function which when minimized over time gives the optimal LQR state-feedback gain K_{LQR} . Essentially, the objective is to keep the magnitude of the output $\mathbf{y}(t)$ small while not using too much control effort.

For the remainder of this and other chapters, K will be considered the same as K_{LQR} unless otherwise noted. Following is a summary of the standard LQR results. Given a linear time-invariant system represented by (2.7) the quadratic regulator problem is to find a control $\mathbf{u}(t)$ to minimize the cost

$$J = \int_0^\infty [\mathbf{u}(t)^T R \mathbf{u}(t) + \mathbf{x}(t)^T Q \mathbf{x}(t)] dt. \quad (3.8)$$

Note that J is an explicit function of $\mathbf{y}(t)$ under the condition that $\mathbf{y}(t) = C\mathbf{x}(t)$ and $Q = C^T C$. In general, the weighting matrix Q should be positive-semidefinite, and R should be positive-definite. Under the assumptions that (A, B) is stabilizable and $(Q^{\frac{1}{2}}, A) = (C, A)$ is detectable, this problem has the well-known solution [66]

$$\mathbf{u}^*(t) = -K\mathbf{x}(t), \quad (3.9)$$

where the state-feedback gain K is given by

$$K = R^{-1}B^T P \quad (3.10)$$

and P denotes the unique positive-definite solution of the algebraic Riccati equation

$$A^T P + PA - PBR^{-1}B^T P + Q = 0. \quad (3.11)$$

With the optimal control $\mathbf{u}^*(t)$, the LQR cost is

$$J = \mathbf{x}(0)^T P \mathbf{x}(0).$$

Since the close-loop system must be stable for the cost to be finite, there exists a positive-definite matrix L which solves the associated Lyapunov equation,

$$(A - BK)^T L + L(A - BK) + S = 0 \quad (3.12)$$

for any positive-semidefinite symmetric matrix S . In particular, if $S = \mathbf{x}(0)\mathbf{x}(0)^T$, then

$$L = \int_0^\infty \left[e^{(A-BK)^T t} S e^{(A-BK)t} \right] dt, \quad (3.13)$$

which is sometimes referred to as the controllability grammian [29]. Finally, the stable closed-loop solution of (2.7) for the control gain given in (3.10) is simply

$$\mathbf{x}(t) = e^{(A-BK)t} \mathbf{x}(0).$$

Discrete-Time LQR

Given the discrete-time linear system of (2.10) and (2.11) the problem is to find $\mathbf{u}(k)$ so that

$$J = \sum_{k=0}^N [\mathbf{x}(k)^T Q \mathbf{x}(k) + \mathbf{u}(k)^T R \mathbf{u}(k)] \quad (3.14)$$

is minimized. The optimal solution [46] is given by the state-feedback

$$\mathbf{u}^*(k) = -K(k)\mathbf{x}(k), \quad (3.15)$$

where the state-feedback gain K is a time-varying gain given by

$$K(k) = [R + H^T P(k+1)H]^{-1} H^T P(k+1)F \quad (3.16)$$

and P obeys the discrete Riccati equation

$$P(k) = F^T P(k+1)F - F^T P(k+1)H [R + H^T P(k+1)H]^{-1} H^T P(k+1)F + Q \quad (3.17)$$

with boundary condition

$$P(N) = Q.$$

Note that even though $K(k)$ is time-varying, it can be precomputed as long as the length N is known. This is because K is not a function of the initial state $\mathbf{x}(0)$. The LQR cost for the optimal control is

$$J = \mathbf{x}(0)^T P(0)\mathbf{x}(0).$$

For the infinite-horizon problem, where $N \rightarrow \infty$, the constant gain solution is the optimum. In steady-state, $P(k)$ becomes $P(k+1)$ and the Riccati equation reduces to

$$P_\infty = F^T P_\infty F - F^T P_\infty H [R + H^T P_\infty H]^{-1} H^T P_\infty F + Q. \quad (3.18)$$

The optimal control is

$$\mathbf{u}(k) = -K_\infty \mathbf{x}(k), \quad (3.19)$$

where

$$K_{\infty} = [R + H^T P_{\infty} H]^{-1} H^T P_{\infty} F. \quad (3.20)$$

The cost associated with this control law is

$$J_{\infty} = \mathbf{x}(0)^T P_{\infty} \mathbf{x}(0).$$

Minimaximization Problems

We now state the general set-membership approach to control design. Let us first consider the continuous time system of (2.7) and (2.8). We assume that the plant to be controlled is known to belong to a set $\mathcal{M}$ as described by Ω of (3.3). Specifically, this means $\theta \in \Omega$, where θ is a parameter vector that parameterizes the plant. We want to find a control $\mathbf{u} : \mathbb{R}_+ \mapsto \mathbb{R}^n$ that solves the following minimax problem:

$$\min_{\mathbf{u}} \max_{\theta \in \Omega} J \quad (3.21)$$

where J is the usual quadratic cost given in (3.8). More precisely, the goal is to find a saddle-point $(\mathbf{u}^*, \theta^*)$ such that

$$J(\mathbf{u}^*, \theta) \leq J(\mathbf{u}^*, \theta^*) \leq J(\mathbf{u}, \theta^*) \quad \forall \mathbf{u} \text{ and } \theta \in \Omega. \quad (3.22)$$

From a game-theoretic point of view, the first player, $\theta \in \Omega$, attempts to maximize the cost and the second player, $\mathbf{u} \in \mathbb{R}^n$, attempts to minimize the cost. Since no particular form is assumed for the control $\mathbf{u}$, such as linear state-feedback, the minimization is over all possible $\mathbf{u}$'s, i.e., all time functions. Note that this same assumption is made initially for the linear quadratic optimal control problem where the plant is known, but the optimal control in that case exhibits the form of linear state-feedback control, (3.9). When this control design is implemented in conjunction

with a set-membership identifier, as described above, a new controller can be designed each time the set estimate, $\widehat{\mathcal{M}}$, is updated.

The minimax problem formulated in [36] is a good example where the first player is a vector θ as defined in (3.3) which describes a parametric uncertainty for the output matrix C of (2.11) and the second player is u of form shown in (3.9).

Example: Lau

The formulation of the problem is as follows. Consider the strictly proper system ($D = 0$) of (2.7) and (2.8) of an uncertain system $\mathcal{M}$ where A , B and $x(0) = x_0$ are given. The elements of $C \in \mathbb{R}^{m_0 \times n_a}$ are in the set Ω defined as:

$$\Omega = \{ \theta : (\theta - \theta_0)^T \Gamma (\theta - \theta_0) \leq 1 \}, \quad (3.23)$$

where Γ is symmetric and positive definite and θ_0 is the center of the ellipsoid. In general, we can take the rows of C , turn them into column vectors and stack them on top of each other to form a $\mathbb{R}^{m_0 n_a}$ column vector c such that $\theta = c \in \Omega$, i.e.,

$$C^T = [c_1 \ c_2 \ \cdots \ c_{m_0}] \in \mathbb{R}^{m_0 \times n_a}$$

$$c = \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_{m_0} \end{bmatrix} \in \mathbb{R}^{m_0 n_a \times 1}.$$

For a given control, $u : \mathbb{R}_+ \mapsto \mathbb{R}^{m_i}$, and a fixed $c \in \Omega$, the objective is defined to be

$$J(u, c) \triangleq \int_0^\infty [u(t)^T R u(t) + y(t)^T y(t)] dt \quad (3.24)$$

where $R \in \mathbb{R}^{m_i \times m_i}$ is a known weighting matrix. Equation (3.24) can be obtained from (3.8) with $Q = C^T C$ and $y(t) = Cx(t)$. We assume that (A, B) is controllable (at least stabilizable) and (C, A) is observable (at least detectable) for all c in Ω . The robust control problem is to find a control u that solves the following minimax

problem:

$$\min_{\mathbf{u} \in \mathbb{R}^m} \max_{\mathbf{c} \in \Omega} J(\mathbf{u}, \mathbf{c}). \quad (3.25)$$

Rewriting (3.24) as

$$J(\mathbf{u}, \mathbf{c}) \triangleq \int_0^\infty [\mathbf{u}(t)^T R \mathbf{u}(t) + \mathbf{x}(t)^T C^T C \mathbf{x}(t)] dt \quad (3.26)$$

leads to another interpretation. We can say that we are designing a controller for a set of uncertain plant cost objectives; in contrast, the standard LQR design is to find a controller for fixed weighting matrices, Q and R . In [36] $(\mathbf{u}^*, \mathbf{c}^*)$ is shown to exist such that

$$J(\mathbf{u}^*, \mathbf{c}) \leq J(\mathbf{u}^*, \mathbf{c}^*) \leq J(\mathbf{u}, \mathbf{c}^*) \quad \forall \mathbf{u} \text{ and } \mathbf{c} \in \Omega \quad (3.27)$$

and that the optimal control is actually the state-feedback LQR control of (3.9), i.e., $\mathbf{u}^*(t) = \mathbf{u}_{LQR}(\mathbf{c}^*)$. Lau proves existence of $(\mathbf{u}^*, \mathbf{c}^*)$ by fixed point theory and shows that the solution must reside on the boundary of the ellipse; two methods for finding the saddle-point solution are given. The first method is an iterative fixed-point algorithm for which there are no guarantees of convergence. The second approach is based on linear matrix inequality methods (LMI, see [67]) where the feedback gain K of (3.10) is computed directly. The method is guaranteed to converge to within any arbitrary degree of accuracy to the solution $(\mathbf{u}^*, \mathbf{c}^*)$. For details see [36] and references therein.

Example: Mills

An approach based on the calculus of variation is proposed in [68]. The approach uses a parameter-change norm, σ , which was first proposed in [69]

$$\sigma^2 = \Delta_\theta^T \Sigma^{-2} \Delta_\theta. \quad (3.28)$$

Δ_θ is a vector of plant parameter changes from nominal. Σ is a symmetric weighting matrix, typically a diagonal matrix of standard deviations of the parameters about the

nominal, mean values. A larger element of Σ allows more variation in the corresponding parameter for a given norm σ . The quadratic norm (3.28) therefore has Gaussian interpretation. Physical system uncertainty often closely follows and is modeled with Gaussian distribution.

Of particular interest is a minimax problem proposed in [68] which iteratively determines the worst-case plant and optimal controller for a set of plants defined by the parameter-change norm (3.28). The cost function is given as

$$J = \min_{u_{LQR}} \max_{\Delta\theta(\sigma)} \int_0^\infty (\mathbf{x}(t)^T Q \mathbf{x}(t) + \mathbf{u}(t)^T R \mathbf{u}(t)) dt$$

with standard assumptions on Q and R . The parameter-change norm σ is assumed a priori. The procedure first involves the maximization for a given linear quadratic control u_{LQR} over the set of plants related to the equality constraint defined by (3.28). Necessary conditions are provided for achieving a maximization. The maximization procedure is the result of standard methods used in the calculus of variations. After the plant associated with the maximization is determined the procedure continues by computing the optimal LQR feedback control u_{LQR} . The algorithm proceeds until a saddle-point is found. Conditions for obtaining a saddle-point are discussed in [68].

Proposed Minimaximization Problems

While Lau has considered the problem where uncertainty is only in the output matrix C , it is natural to consider the problem where uncertainty in the matrices A and B can be placed in the form of ellipsoidal bounds on the associate parameters. Furthermore, unlike the approach taken in [68], it is worth considering the case where the parameters are allowed to vary over the entire space defined by the ellipsoidal bound. In addition, current *ellipsoid* identification schemes are restricted to SISO systems as seen in [65, 62, 36, 70]. A natural minimax problem to consider then is for the SISO system. Next, we construct the minimax problem for both the continuous time and discrete

time system where the uncertainty in the numerator and denominator parameters is contained in a bounded ellipsoidal domain.

Continuous Plant Consider the strictly proper case of the system of (2.9) (i.e., set $b_{n_b+1} = 0$ and $n_b \leq n_a$) where there is uncertainty in the numerator and denominator parameters. In observable form (2.7) and (2.8) are represented as

$$A \triangleq \begin{bmatrix} 0 & \cdots & 0 & -a_1 \\ \vdots & & & \vdots \\ I & & & -a_{n_a} \end{bmatrix}, \quad B \triangleq \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_{n_b} \\ 0^{(n_a-n_b)} \end{bmatrix}$$

and

$$C \triangleq [0 \quad \cdots \quad 0 \quad 1].$$

Define the parameter set θ_{AB} by

$$\theta_{AB}^T \triangleq [a_1 \quad a_2 \quad \cdots \quad a_{n_a} \quad b_1 \quad b_2 \quad \cdots \quad b_{n_b}]^T. \quad (3.29)$$

The parameter uncertainty set in θ_{AB} is defined as follows:

$$\Omega \triangleq \{\theta_{AB} : (\theta_{AB} - \theta_o)^T \Gamma (\theta_{AB} - \theta_o) \leq 1\}, \quad (3.30)$$

where θ_o contains the nominal values of the system parameters defined as

$$\theta_o^T \triangleq [\theta_{o1} \quad \theta_{o2} \quad \cdots \quad \theta_{o\{n_a+n_b\}}] \quad (3.31)$$

and matrix Γ defines the bound and is symmetric and positive definite given by

$$\Gamma \triangleq \begin{bmatrix} \gamma_1 & \gamma_2 & \gamma_3 & \cdots & \gamma_{\eta-1} & \gamma_{\eta} \\ \gamma_2 & \gamma_{\eta+1} & \gamma_{\eta+2} & \cdots & \gamma_{2\eta-2} & \gamma_{2\eta-1} \\ \gamma_3 & \gamma_{\eta+2} & \gamma_{2\eta} & \gamma_{2\eta+1} & \cdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \gamma_{\eta-1} & \gamma_{2\eta-2} & \cdots & \cdots & \cdots & \vdots \\ \gamma_{\eta} & \gamma_{2\eta-1} & \cdots & \cdots & \cdots & \gamma_{\frac{\eta(\eta+1)}{2}} \end{bmatrix}, \quad (3.32)$$

where $\eta \triangleq n_a + n_b$.

The problem is to find a control signal $u(t)$ that stabilizes $G(s)$ of (2.9) for all $\theta_{AB} \in \Omega$ and is optimum in the sense of the minimaximization of the quadratic cost function

$$J(u, \theta_{AB}) \triangleq \int_0^\infty [u(t)^T R u(t) + y^2(t)] dt \quad (3.33)$$

where $R > 0$ is a known weighting scalar. In general, the existence of such an optimal control signal will be dependent on the size of Γ and the stabilizability of the system (2.9) over the parameter space $\theta_{AB} \in \Omega$. From dynamic game theory [71], $J(u^*, \theta_{AB}^*)$ is optimum if and only if

$$J(u^*, \theta_{AB}) \leq J(u^*, \theta_{AB}^*) \leq J(u, \theta_{AB}^*) \quad (3.34)$$

for all $u : \mathbb{R}_+ \rightarrow \mathbb{R}$ and $\theta_{AB} \in \Omega$. Note that the type of control has not been established and therefore can either be linear or nonlinear. However, from LQR control theory, the second inequality in (3.34) is true if and only if

$$u^* = u_{LQR}(\theta_{AB}^*),$$

where $u_{LQR}(\theta_{AB}^*)$ denotes the LQR control designed for the plant with $\theta_{AB} = \theta_{AB}^* \in \Omega$. Thus, if a saddle point exists, the type of control implemented must be of the state feedback form given by (3.9).

Discrete Plant The discrete counterpart to the continuous time SISO system is now stated. If the sampling rate is given by T and $z \triangleq e^{Ts}$ the pulse-transfer function is simply given by

$$G(z) = \frac{b_1 z^{-1} + \dots + b_{n_b} z^{-n_b}}{1 + a_1 z^{-1} + \dots + a_{n_a} z^{-n_a}} \quad (3.35)$$

where $n_b \leq n_a$. In observable form (2.10) and (2.11) are represented as

$$F \triangleq \begin{bmatrix} 0 & \cdots & 0 & -a_{n_a} \\ \vdots & \vdots & \vdots & -a_{n_a-1} \\ I & & & \vdots \\ & & & -a_1 \end{bmatrix}, \quad H \triangleq \begin{bmatrix} 0^{(n_a-n_b)} \\ b_{n_b} \\ b_{n_b-1} \\ \vdots \\ b_1 \end{bmatrix}$$

and

$$C \triangleq \begin{bmatrix} 0 & \cdots & 0 & 1 \end{bmatrix}.$$

It is clear that the a_i 's and b_i 's of (3.35) differ from those of the continuous case of (3.29). However, the parameters in a_i and b_i can be assigned in the same way as in the continuous case, as also can the parameter uncertainty set Ω (3.30). Replacing the subscripts AB with FH we obtain the parameter uncertainty set in θ_{FH} as follows:

$$\Omega \triangleq \{\theta_{FH} : (\theta_{FH} - \theta_o)^T \Gamma (\theta_{FH} - \theta_o) \leq 1\}, \quad (3.36)$$

where θ_o contains the nominal values of the system parameters. The nominal parameter vector and the matrix Γ are defined in the same way as in the continuous time case.

The problem is to find a control signal $u(k)$ that stabilizes $G(z)$ of (3.35) for all $\theta_{FH} \in \Omega$ and is optimum in the sense of the minimaximization of the quadratic cost function

$$J(u, \theta_{FH}) \triangleq \sum_{k=0}^{\infty} [u(k)^T R u(k) + y^2(k)] \quad (3.37)$$

where $R > 0$ is a known weighting scalar.

Discussion of Results

This chapter has served to introduce two minimaximization problems. The review of linear quadratic control theory provided the necessary tools to establish the statement of the problem. Motivation for these problems is primarily due to the efforts of Lau,

Fogel and Cheung [32, 61, 62] in the area of system identification and the efforts of Lau and Mills [72, 68] in the area of set-membership control. From standard results in linear quadratic control theory we see that, in order to determine the saddle point solution, the control strategy for the minimaximization problem must be LQR linear state-feedback. The problems are different from those of Lau and Mills in that the uncertain parameters are the parameters of the state-space matrices A and B (or F and H) and are allowed to vary within a bounded ellipsoidal domain. The ellipsoidal domain is determined through identification schemes similar to the method of Cheung [62]. The following chapter proposes a fixed-point iterative algorithm to find the desired saddle-point solution. The iterative solutions for the continuous and the discrete time systems are very similar.

CHAPTER 4

MINIMAXIMIZATION FOR SINGLE-INPUT SINGLE-OUTPUT SYSTEMS

In this chapter we derive an iterative algorithm for finding a saddle-point to the minimax problem developed in the previous chapter. Both the continuous-time and discrete-time SISO plants are considered where the system parameters are allowed to vary over the entire ellipsoidal domain defined by (3.30) and (3.36), respectively.

Continuous-Time System Problem Statement

The system is described by (2.7)-(2.9) where we set $b_{n_b+1} = 0$ and $n_b \leq n_a$, and by (3.29) and (3.30). As discussed in Chapter 3, the form of the control that is of interest here is linear state feedback

$$u(t) = -K\mathbf{x}(t).$$

The set of stabilizing controllers is defined by

$$\mathcal{K} \triangleq \{K : K \text{ stabilizes } G(s) \forall \theta_{AB} \in \Omega\}.$$

Those cases where $\mathcal{K}$ is nonempty are of interest.

Statement of Problem: Find a controller $K^* \in \mathcal{K}$ that stabilizes $G(s) \forall \theta_{AB} \in \Omega$ and is optimum in the sense of the minimaximization of the quadratic cost function

$$J(K, \theta_{AB}) = \int_0^\infty \mathbf{x}^T(t)(K^T R K + C^T C)\mathbf{x}(t)dt, \quad (4.1)$$

where R is symmetric and positive definite, and $\mathbf{x}(t) = e^{(A-BK)t}\mathbf{x}(0)$. Since the system is SISO, $R \in \mathbb{R}$ is scalar and must be greater than 0. This problem is equivalent

to finding a saddle point solution. Note that (4.1) is equivalent to (3.8) under the condition that $\mathbf{y}(t) = C\mathbf{x}(t)$ and $Q = C^T C$.

As shown in Chapter 3, $J(K^*, \theta_{AB}^*)$ (the saddle point of interest) is optimum if and only if

$$J(K^*, \theta_{AB}) \leq J(K^*, \theta_{AB}^*) \leq J(K, \theta_{AB}^*). \quad (4.2)$$

Following, we show that the solution may not be unique, however, if there are multiple solutions, all optimal LQR costs must be equal. If it is the case that more than one solution exists, then it does not matter which solution is chosen in the final design.

Theorem 4.1 (Uniqueness of Solution): *If there are more than one (K, θ) which satisfy (4.2) then their associated LQR costs must be equal and any one of the solutions can be used to compute the minimax control.*

Proof: (This proof follows from [36]). Assume there exist (K_1, θ_1) and (K_2, θ_2) such that

$$J(K_1, \theta) \leq J(K_1, \theta_1) \leq J(K, \theta_1) \quad \forall K, \theta \quad (4.3)$$

and

$$J(K_2, \theta) \leq J(K_2, \theta_2) \leq J(K, \theta_2) \quad \forall K, \theta. \quad (4.4)$$

Setting $\theta = \theta_2$ and $K = K_2$ in (4.3), we get

$$J(K_1, \theta_2) \leq J(K_1, \theta_1) \leq J(K_2, \theta_1).$$

From (4.4)

$$J(K_2, \theta_1) \leq J(K_2, \theta_2) \Rightarrow J(K_1, \theta_1) \leq J(K_2, \theta_2) \quad (4.5)$$

Similarly, setting $\theta = \theta_1$ and $K = K_1$ in (4.4), we get

$$J(K_2, \theta_1) \leq J(K_2, \theta_2) \leq J(K_1, \theta_2).$$

From (4.3)

$$J(K_1, \theta_2) \leq J(K_1, \theta_1) \Rightarrow J(K_2, \theta_2) \leq J(K_1, \theta_1). \quad (4.6)$$

Equations (4.5) and (4.6) imply $J(K_1, \theta_1) = J(K_2, \theta_2)$. It follows that if there are k solutions $J(K_k, \theta_k)$ that satisfy (4.2) then $J(K_i, \theta_i) = J(K_j, \theta_j)$ for any $i, j = 1, 2, \dots, k$. Thus, any one of the solutions (K_i, θ_i) for $i = 1, 2, \dots, k$ may be used for the final control design. $\square$

Assuming a saddle point exists, it is necessarily true that

$$\theta_{AB}^* = \arg \max_{\theta_{AB} \in \Omega} \int_0^\infty \mathbf{x}^T(t) (K^{*T} R K^* + C^T C) \mathbf{x}(t) dt. \quad (4.7)$$

This fact is used in the following section. From this viewpoint we seek a two-step iterative algorithm, that first solves the LQR minimization for K , and second solves (4.7) for θ_{AB} , such that the algorithm converges to (K^*, θ_{AB}^*) .

Some Useful Results from LQR Control Theory

Since we are seeking a controller in the form of standard LQR state-feedback it can be shown that the minimization of (4.1) over $K \in \mathcal{K}$ yields the following sets of identities.

First Set:

$$K^T R K + C^T C = -[(A - BK)^T P + P(A - BK)] \quad (4.8)$$

for any set $\theta_{AB} \in \Omega$ where P , symmetric and positive definite, is the solution to the standard Riccati equation of (3.11) and the feedback gain K is given by (3.10). We also know from LQR theory that for a given $[A, B]$, the optimal cost is given by

$$\min_{K \in \mathcal{K}} J(K, \theta_{AB}) = \mathbf{x}(0)^T P \mathbf{x}(0). \quad (4.9)$$

Second Set:

Using the identity $\mathbf{z}^T \Phi \mathbf{z} \triangleq \text{tr}\{\Phi \mathbf{z} \mathbf{z}^T\}$, where $\text{tr} \equiv \text{trace}$, the standard linear operator, the cost function from (4.1) can be written as

$$J(K, \theta_{AB}) = \text{tr} \left[\int_0^\infty Z(K) \mathbf{x}(t) \mathbf{x}^T(t) dt \right], \quad (4.10)$$

where $Z(K)$ is given by

$$Z(K) \triangleq K^T R K + C^T C,$$

and $\mathbf{x}(t)$ is a function of θ_{AB} . Using the fact that $Z(K)$ is independent of time, (4.10) becomes

$$J(K, \theta_{AB}) = \text{tr} \left[Z(K) \int_0^\infty \mathbf{x}(t) \mathbf{x}^T(t) dt \right]. \quad (4.11)$$

Since the closed-loop system must be stable for the cost to be finite, there exists a positive definite matrix L which solves the associated Lyapunov equation,

$$(A - BK)L + L(A - BK)^T + \mathbf{x}_0 \mathbf{x}_0^T = 0 \quad (4.12)$$

where $\mathbf{x}_0 \triangleq \mathbf{x}(0)$. The solution L of (4.12) is actually a function of θ_{AB} and is given by

$$L = \int_0^\infty \mathbf{x}(t) \mathbf{x}^T(t) dt. \quad (4.13)$$

Equation (4.13) is sometimes referred to as the controllability grammian. Thus, (4.11) becomes

$$J(K, \theta_{AB}) = \text{tr} [Z(K)L]. \quad (4.14)$$

These sets of identities lead to two possible iterative approaches for finding $[K^*, \theta_{AB}^*]$ and are given in the following sections.

Iterative Solution Approaches

The first approach follows from the first set of identities using equations (4.8)-(4.9).

Algorithm 1:

Let $[A_{(j)}, B_{(j)}]$ be given at the j^{th} iterate.

- I. (minimize) Solve the Riccati equation of (4.15) and the feedback control law of (4.16) for $P_{(j)}$ and $K_{(j)}$, respectively.

$$A_{(j)}^T P_{(j)} + P_{(j)} A_{(j)} - P_{(j)}^T B_{(j)} R^{-1} B_{(j)}^T P_{(j)} + C^T C = 0. \quad (4.15)$$

$$K_{(j)} = R^{-1} B_{(j)}^T P_{(j)}. \quad (4.16)$$

- II. (maximize) Solve (4.17) for $[A_{(j+1)}, B_{(j+1)}]$.

$$[A_{(j+1)}, B_{(j+1)}] = \arg \max_{\theta_{AB} \in \Omega} \mathbf{x}_0^T P \mathbf{x}_0 \quad (4.17)$$

subject to

$$(A - BK_{(j)})^T P + P(A - BK_{(j)}) + K_{(j)}^T R K_{(j)} + C^T C = 0.$$

where $P = P(A, B)$ in (4.17) because of the above constraint.

- III. (check convergence) If $\|[A_{(j+1)}, B_{(j+1)}] - [A_{(j)}, B_{(j)}]\|$ is less than some desired tolerance, then stop. Otherwise, let $[A_{(j+1)}, B_{(j+1)}] \rightarrow [A_{(j)}, B_{(j)}]$ and continue at step I.

The algorithm may be initiated by selecting the nominal parameter values of Ω for $[A_{(0)}, B_{(0)}]$.

The second approach follows from the second set of identities using equations (4.12)-(4.14) and also equations (3.10) and (3.11).

Algorithm 2:

Let $[A_{(j)}, B_{(j)}]$ be given at the j^{th} iterate.

- I. (minimize) Solve the Riccati equation of (4.18) and the feedback control law of (4.19) for $P_{(j)}$ and $K_{(j)}$, respectively.

$$A_{(j)}^T P_{(j)} + P_{(j)} A_{(j)} - P_{(j)}^T B_{(j)} R^{-1} B_{(j)}^T P_{(j)} + C^T C = 0. \quad (4.18)$$

$$K_{(j)} = R^{-1} B_{(j)}^T P_{(j)}. \quad (4.19)$$

- II. (maximize) Solve (4.20) for $[A_{(j+1)}, B_{(j+1)}]$.

$$[A_{(j+1)}, B_{(j+1)}] = \arg \max_{\theta_{AB} \in \Omega} -tr [Z(K_{(j)})L] \quad (4.20)$$

subject to

$$(A - BK_{(j)})L + L(A - BK_{(j)})^T + x_0 x_0^T = 0.$$

where $L = L(A, B, x_0)$ in (4.20) because of the above constraint.

- III. (check convergence) If $\|[A_{(j+1)}, B_{(j+1)}] - [A_{(j)}, B_{(j)}]\|$ is less than some desired tolerance, then stop. Otherwise, let $[A_{(j+1)}, B_{(j+1)}] \rightarrow [A_{(j)}, B_{(j)}]$ and continue at step I.

Remarks:

- In step III of both iterative approaches, the norm on the differences would actually be performed using the equivalent vector of parameters $\theta_{AB}^{(j+1)}$ and $\theta_{AB}^{(j)}$.
- The first iterative approach is dependent on the initial conditions through the final optimal cost equation of (4.9); the second iterative approach is dependent on the initial conditions through the controllability grammian of (4.12). However, this dependence on x_0 can be removed if we start with the assumption

that $\mathbf{x}_0$ is a random vector with known mean $\bar{\mathbf{m}}$ and covariance V and the objective in (4.1) is an expectation over $\mathbf{x}_0$. In that case, $\mathbf{x}_0 \mathbf{x}_0^T$ may be replaced by $V + \bar{\mathbf{m}} \bar{\mathbf{m}}^T$. Another alternative is to use the worst initial condition of unit magnitude as in [69].

- Equivalence of each iterative approach is implied by virtue of existence of the LQR minimization solution and thus the existence of the associated controllability grammian of (4.12).
- The maximization in step II of **Algorithm 1** may be stated more precisely.

Let the symmetric matrix P be defined as

$$P \triangleq \begin{bmatrix} p_1 & p_2 & p_3 & \cdots & p_{n_a-1} & p_{n_a} \\ p_2 & p_{n_a+1} & p_{n_a+2} & \cdots & p_{2n_a-2} & p_{2n_a-1} \\ p_3 & p_{n_a+2} & p_{2n_a} & p_{2n_a+1} & \cdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ p_{n_a-1} & p_{2n_a-2} & \cdots & \cdots & \cdots & \vdots \\ p_{n_a} & p_{2n_a-1} & \cdots & \cdots & \cdots & p_{\frac{n_a(n_a+1)}{2}} \end{bmatrix}$$

Next, let $f(\xi) \triangleq \mathbf{x}_0^T P \mathbf{x}_0$ where ξ is given as

$$\xi \triangleq \left[a_1 \ a_2 \ \cdots \ a_{n_a} \ b_1 \ b_2 \ \cdots \ b_{n_b} \ p_1 \ p_2 \ \cdots \ p_{\frac{n_a(n_a+1)}{2}} \right]^T.$$

Solve (4.21) for $\xi_{(k+1)}$.

$$\xi_{(k+1)} = \arg \max_{\xi \in \mathbb{R}^N} f(\xi) \quad (4.21)$$

subject to

$$\theta_{AB} \in \Omega \quad (4.22)$$

and

$$(A - BK_{(j)})^T P + P(A - BK_{(j)}) + K_{(j)}^T R K_{(j)} + C^T C = \mathbf{0}, \quad (4.23)$$

where $N \triangleq n_a + n_b + \frac{n_a(n_a+1)}{2}$.

- Similarly, the maximization in step II of Algorithm 2 may be stated more precisely.

Let the symmetric matrix L be defined as

$$L \triangleq \begin{bmatrix} l_1 & l_2 & l_3 & \cdots & l_{n_a-1} & l_{n_a} \\ l_2 & l_{n_a+1} & l_{n_a+2} & \cdots & l_{2n_a-2} & l_{2n_a-1} \\ l_3 & l_{n_a+2} & l_{2n_a} & l_{2n_a+1} & \cdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ l_{n_a-1} & l_{2n_a-2} & \cdots & \cdots & \cdots & \vdots \\ l_{n_a} & l_{2n_a-1} & \cdots & \cdots & \cdots & l_{\frac{n_a(n_a+1)}{2}} \end{bmatrix}$$

Next, let $f(\xi) \triangleq \text{tr} [Z(K_{(j)})L]$ where ξ is given as

$$\xi \triangleq \begin{bmatrix} a_1 & a_2 & \cdots & a_{n_a} & b_1 & b_2 & \cdots & b_{n_b} & l_1 & l_2 & \cdots & l_{\frac{n_a(n_a+1)}{2}} \end{bmatrix}^T.$$

Solve (4.24) for $\xi_{(k+1)}$.

$$\xi_{(k+1)} = \arg \max_{\xi \in \mathbb{R}^N} f(\xi) \quad (4.24)$$

subject to

$$\theta_{AB} \in \Omega$$

and

$$(A - BK_{(j)})L + L(A - BK_{(j)})^T + \mathbf{x}_0 \mathbf{x}_0^T = \mathbf{0}.$$

where $N \triangleq n_a + n_b + \frac{n_a(n_a+1)}{2}$.

- The primary difference between the algorithms is that the first algorithm results from a parameterization in the elements of the Riccati matrix P and the second results from a parameterization in the elements of the grammian matrix L .
- With these simple changes in notation we still check for convergence using $\theta_{AB}^{(j+1)}$ and $\theta_{AB}^{(j)}$.

Although it is true that multiple saddle points exhibit equivalent LQR costs, it is possible for the algorithm to converge to a local saddle-point and not necessarily

a global saddle-point. For this reason, it is necessary to apply K^* to all plants within the plant set Ω to insure that (K^*, θ_{AB}^*) is a global maximum or stated another way, to insure that the control gain K^* does not cause destabilization for any other plants. It is not practical to test the optimal controller on all plants; therefore, a selection of plants from the plant set Ω must be chosen to determine the likelihood that all plants in Ω are stabilized by K^* . One approach is to select N_e sub-ellipsoids of the bounding ellipsoid Ω such that $\theta_0 = \Omega_1 \subset \Omega_2 \subset \dots \subset \Omega_{N_e} \subset \Omega$. Note that the limiting case Ω_1 is actually the center of the ellipsoid Ω and is simply given by the nominal parameters θ_0 . For each sub-ellipsoid, Ω_i , select N_b plants, θ_{AB}^{ij} , $j = 1, 2, \dots, N_b$, on the boundary, apply the optimal gain K^* , and compute the LQR cost $J(K^*, \theta_{AB}^{ij})$. This procedure checks a total of $N_e N_b$ plants within Ω . If (K^*, θ_{AB}^*) is a global maximum then according to (4.2) the following must hold

$$J(K^*, \theta_{AB}^{ij}) \leq J(K^*, \theta_{AB}^*).$$

Examples

In all examples, unless otherwise stated, the convergence criterion is given by

$$\|\theta_{AB}^{(j+1)} - \theta_{AB}^{(j)}\|_2 \leq tol$$

where the tolerance is $tol = 0.0001$.

Example 1: Two System Parameters (Stabilizable) The first example is for a first-order system with two uncertain parameters, a_1 and b_1 . The output matrix $C = 1$, the LQR weight $R = r_1$, and the initial condition $x(0) = x_0$. The positive symmetric matrix Γ of (3.32) has three parameters, γ_1, γ_2 and γ_3 while the nominal vector θ_0 of (3.31) has two parameters θ_{01} and θ_{02} . The Riccati matrix of

(4.15) is a scalar, i.e., $P = p_1$ and the feedback gain of (4.16) is scalar, i.e., $K = k_1$.

The number of parameters to maximize over is $N = 3$, thus, ξ of (4.21) is given by $\xi = [a_1 \ b_1 \ p_1]^T \triangleq [\xi_1 \ \xi_2 \ \xi_3]^T$. The function of (4.21) reduces to

$$f(\xi) = \xi_3 x_0^2.$$

The inequality constraint given by (4.22) is given by

$$[(\xi_1 - \theta_{01}), (\xi_2 - \theta_{02})]^T \Gamma [(\xi_1 - \theta_{01}), (\xi_2 - \theta_{02})] - 1 \leq 0 \quad (4.25)$$

and the equality constraint given by (4.23) is simplified to

$$-2(\xi_1 + \xi_2 k_1) \xi_3 + r_1 k_1^2 + 1 = 0. \quad (4.26)$$

While the existence of a solution has not been shown, it is worthwhile to consider where the solution may reside. If the domain of ξ is convex, then the function $f(\xi)$, which is convex by linearity, will have its solution at an extreme point of the domain [73, Theorem 3, page 119]. Consider the following definition of convexity.

Definition 4.1 *A set Ω_c in $\mathbb{R}^n$ is said to be convex if for every $\xi_a, \xi_b \in \Omega_c$ and every real number α , $0 < \alpha < 1$, the point $\alpha \xi_a + (1 - \alpha) \xi_b \in \Omega_c$.*

Let the domain defined by the equality constraint (4.26) be represented by Ω_ξ . The domain of ξ is $\Omega_N \triangleq \Omega \cap \Omega_\xi$ where Ω is the ellipsoidal domain given by the inequality constraint (4.25) and is defined in (3.30). It is well known that the intersection of any collection of convex sets is also convex [73, see Appendix B]. In order for Ω_N to be convex it is sufficient to show Ω_ξ to be convex. While it would be helpful for Ω_N to be convex, this is, in general, not true. It can be seen by further analysis of (4.26) that Ω_ξ is, in general, not convex and thus Ω_N is, in general, not convex. We show this through theorem and proof.

Theorem 4.2 In general, Ω_ξ , defined by (4.26), is not convex and therefore, $\Omega_N \triangleq \Omega \cap \Omega_\xi$, where Ω is defined by (4.25), is, in general, not convex.

Proof: {Proof by counterexample} Assume there exist two vectors ξ_a and $\xi_b \in \Omega_\xi$ such that $\alpha\xi_a + (1 - \alpha)\xi_b \in \Omega_\xi$ for any scalar α where $0 < \alpha < 1$. From (4.26), this is equivalent to saying

$$h(\alpha\xi_a + (1 - \alpha)\xi_b) = d \text{ for any } \alpha : 0 < \alpha < 1 \quad (4.27)$$

where $h(\xi) \triangleq (\xi_1 + \xi_2 k_1)\xi_3$ and $d \triangleq \frac{r_1 k_1^2 + 1}{2}$. Defining $\xi_a \triangleq [\xi_{a1} \ \xi_{a2} \ \xi_{a3}]^T$ and $\xi_b \triangleq [\xi_{b1} \ \xi_{b2} \ \xi_{b3}]^T$ and using the fact that $h(\xi) = d$, $\forall \xi \in \Omega_\xi$, it can be shown, with some algebraic manipulation, that (4.27) reduces to

$$h(\alpha\xi_a + (1 - \alpha)\xi_b) = \left(\alpha(1 - \alpha) \left(\frac{\xi_{a3}^2 + \xi_{b3}^2}{\xi_{a3}\xi_{b3}} \right) + (1 - 2\alpha) \right) d$$

for any α such that $0 < \alpha < 1$. Hence, using (4.27) again, we obtain

$$\begin{aligned} \alpha &= 1 - \left(\frac{\xi_{a3}^2 + \xi_{b3}^2}{2\xi_{a3}\xi_{b3}} \right)^{-1} \\ &= 1 - \left(\frac{(\xi_{a1} + \xi_{a2}k_1)^2 + (\xi_{b1} + \xi_{b2}k_1)^2}{2(\xi_{a1} + \xi_{a2}k_1)(\xi_{b1} + \xi_{b2}k_1)} \right)^{-1} \end{aligned} \quad (4.28)$$

Note that (4.28) is independent of r_1 . Equation (4.28) is generally not true for any arbitrary selection of $\xi_a, \xi_b \in \Omega_\xi$ and for any $\alpha \in (0, 1)$, as can be seen by the following example.

Let $\Gamma = I_2 \times 2$, the weight $r_1 = 1$, the nominal values $[\theta_{01}, \theta_{02}] = [2, 0.2]^T$, and the initial condition $x_0 = 1$. Further, let $[\xi_{a1}, \xi_{a2}] = [1 \ 1]$ and $[\xi_{b1}, \xi_{b2}] = [3 \ 1]$ (although each point satisfies the inequality constraint (4.25), this is not a requirement since the intersection of Ω and Ω_ξ has not yet been taken). We see that (4.28) is only true for $\alpha = \frac{1}{5}$ and not for any arbitrary $\alpha \in (0, 1)$. Therefore, in general, Ω_ξ as defined in (4.26) is not convex. It follows that, in general, the intersection of the

convex domain Ω with the non-convex domain Ω_ξ is also non-convex. $\square$

Remark: This observation does not preclude the possibility that a saddle point exists on the boundary of Ω .

Indeed, for this example, i.e., $\Gamma = I_2 \times 2$, the weight $r_1 = 1$, the nominal values $[\theta_{01}, \theta_{02}] = [2, 0.2]^T$, and the initial condition $x_0 = 1$, the system exhibits a saddle point on the boundary of Ω . The saddle point is shown in Figure 19. **Algorithm 1** converged in 15 iterations to the *worst-case* plant parameters $\theta_{AB}^* = [1.0002, 0.1337]^T$. The optimal gain associated with these parameters is $K^* = 0.0664$ and the LQR cost is $J^* = 0.4967$.

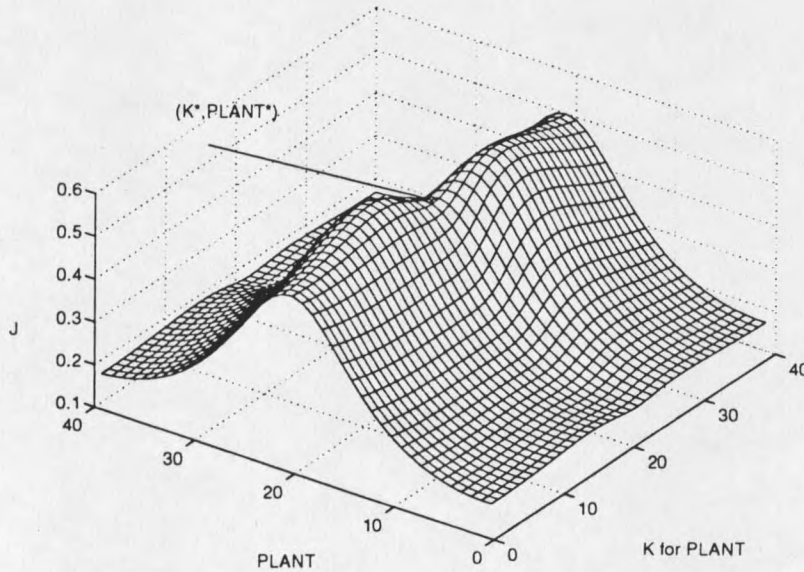


Figure 19. Illustration of saddle point on the boundary of Ω for Example 1.

In Figure 20, “PLANT” is equivalent to θ_{AB} of (4.2). The horizontal axis represents 40 plants uniformly distributed along the boundary of Ω . The vertical axis is the LQR cost for a controller/plant pair. The optimal cost, $J(K^*, \theta_{AB}^*)$, is shown as a horizontal dashed line. The curve above the optimal cost line is for the optimal

plant θ_{AB}^* but with the application of the LQR gains for the plants distributed around the perimeter of Ω . The curve below the optimal cost line is for the plants distributed around the perimeter of Ω and with the optimal control gain K^* applied. Clearly, Figure 20 reveals that the saddle point condition of (4.2) is at least satisfied for those plants around the perimeter of Ω .

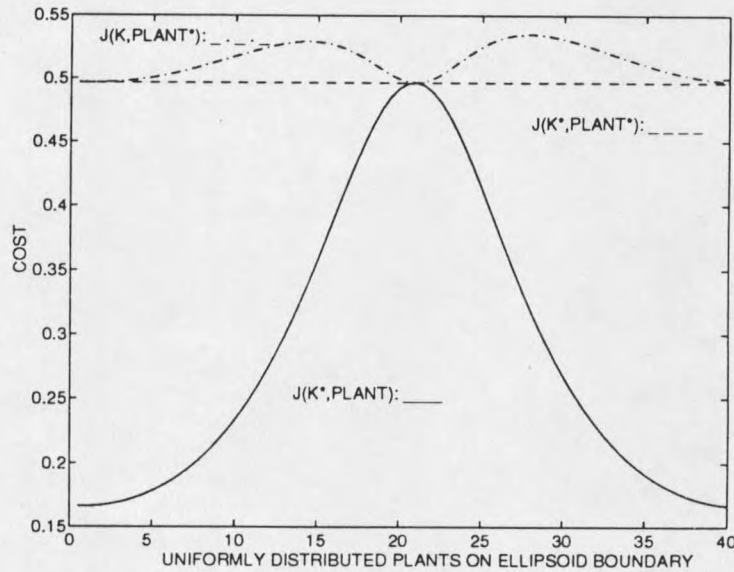


Figure 20. Illustration in two dimension of saddle point condition for Example 1.

Remark: The controllability matrix, $\mathcal{C} = b_1$, is singular for some plants in the ellipsoid, Ω . Although the system is not controllable it is at least stabilizable. If $\theta_0 = [0, 0.2]^T$, the system is not controllable nor is it stabilizable for some of the plants in the ellipsoid. For this system **Algorithm 1** fails to converge. It is well known that the solution to the standard Riccati equation exists as long as the system is at least stabilizable and detectable. Since **Algorithm 1** solves a Riccati equation, it is *necessary* for all plants in Ω to be at least stabilizable. To support this observation consider the following example. We obtain the same results using **Algorithm 2**.

Example 2: Two System Parameters (Controllable, Unstable) The

following example is an uncertain system where all plants in the region Ω are controllable, yet are possibly unstable. The description of the uncertain system is summarized below.

- $\Gamma = I_2 \times 2$
- $r_1 = 1$
- $\theta_0 = [0.1 \ 2]^T$
- $x_0 = 1$

After 14 iterations Algorithm 1 converged to the following optimal values:

- $\theta_{AB}^* = [0.4627 \ 1.1733]^T$
- $K^* = 1.4692$
- $J(K^*, \theta_{AB}^*) = 1.2522$

These results are shown graphically in Figure 21. Note that the cost increases sharply as other optimal LQR control gains are applied to the plants that are near to the plant given by θ_{AB}^* . This is really of no concern, however, the question remains as to whether the gain given by K^* results in larger costs than $J(K^*, \theta_{AB}^*)$ when applied to plants in the interior of Ω .

Figure 22 shows the LQR cost for plants located around the perimeter of ellipsoids contained in Ω . The size of the interior ellipsoids are monotonically decreasing in size. The limiting case is the single point corresponding to the center of the ellipsoid. The limiting point becomes a horizontal line at an LQR cost of 0.5198. As is evident, the true saddle point is located on the boundary of Ω . For several other examples

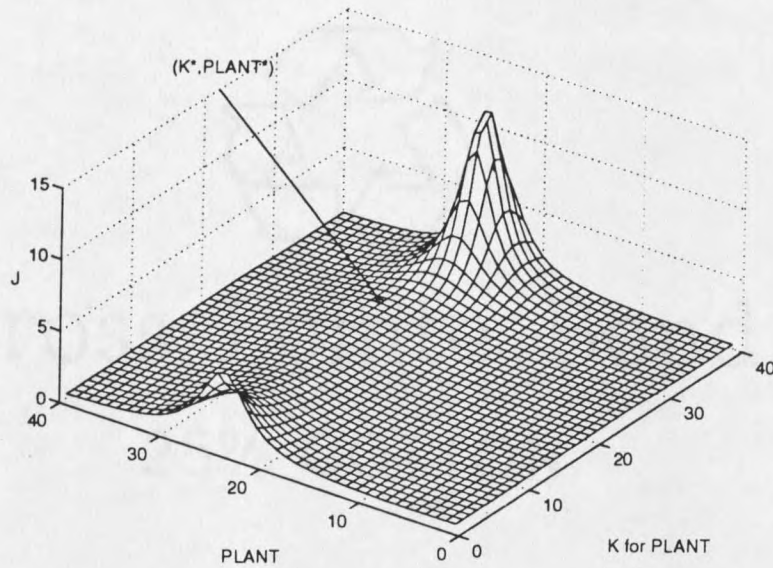


Figure 21. Illustration of saddle point on the boundary of Ω for Example 2.

we noted that all the solutions tended to the boundary of the ellipsoid. A proof for existence of solution is not provided for the continuous time problem.

Discrete-Time System Problem Statement

The system is described by (3.35) and (3.36). As discussed in Chapter 3, the form of the control that is of interest here is linear state feedback

$$u(k) = -Kx(k).$$

where $K = K_\infty$ of (3.20). The set of stabilizing controllers is defined by

$$\mathcal{K} \triangleq \{K : K \text{ stabilizes } G(z) \forall \theta_{FH} \in \Omega\}.$$

Those cases where $\mathcal{K}$ is nonempty are of interest.

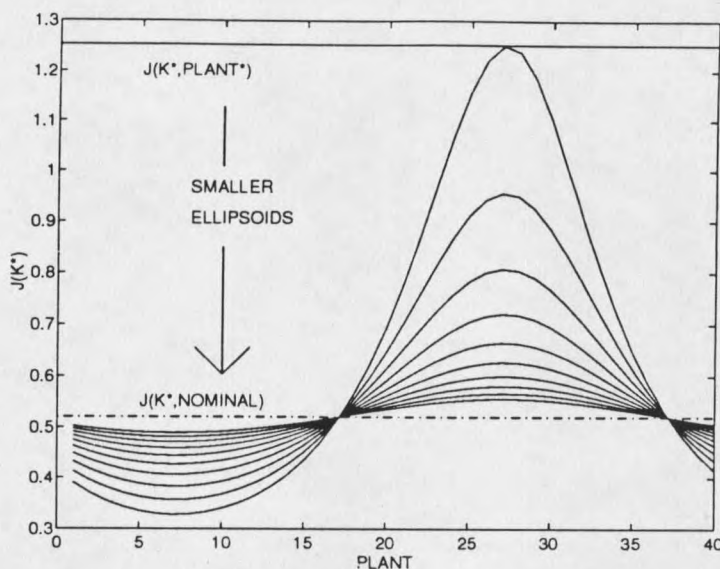


Figure 22. LQR costs for plants within interior of uncertainty region using $K^* = 1.4692$; Example 2.

Statement of Problem: Find a controller $K^* \in \mathcal{K}$ that stabilizes $G(z) \forall \theta_{FH} \in \Omega$ and is optimum in the sense of the minimaximization of the quadratic cost function

$$J(K, \theta_{FH}) = \sum_{k=0}^{N \rightarrow \infty} [\mathbf{x}(k)^T (K^T R K + C^T C) \mathbf{x}(k)] \quad (4.29)$$

where R is symmetric and positive definite. Since the system is SISO, $R \in \mathbb{R}$ is scalar and must be greater than 0. This problem is equivalent to finding a saddle point solution. Note that (4.29) is equivalent to (3.14) under the condition that $\mathbf{y}(k) = C\mathbf{x}(k)$ and $Q = C^T C$.

As shown in Chapter 3, $J(K^*, \theta_{FH}^*)$ (the saddle point of interest) is optimum if and only if

$$J(K^*, \theta_{FH}) \leq J(K^*, \theta_{FH}^*) \leq J(K, \theta_{FH}^*). \quad (4.30)$$

As in the case for the continuous time system **Theorem 4.1** holds for the discrete case. Assuming a saddle point exists, it is necessarily true that

$$\theta_{FH}^* = \arg \max_{\theta_{FH} \in \Omega} \sum_{k=0}^{N \rightarrow \infty} [\mathbf{x}(k)^T (K^T R K + C^T C) \mathbf{x}(k)]. \quad (4.31)$$

This fact is used in the following section. From this viewpoint we seek a two-step iterative algorithm, that first solves the LQR minimization for K , and second solves (4.31) for θ_{FH} , such that the algorithm converges to (K^*, θ_{FH}^*) .

Iterative Solution Approach

The iterative approach that follows is similar to **Algorithm 1** of the continuous-time problem. In this algorithm we make use of (3.18), the infinite-horizon discrete-time Riccati equation. We also make use of the discrete-time Lyapunov equation

$$(F - HK)^T P (F - HK) - P + K^T R K + C^T C = 0. \quad (4.32)$$

For a stable matrix $F - HK$, there exists a positive definite matrix P which satisfies (4.32) [46].

Algorithm 3:

Let $[F_{(j)}, H_{(j)}]$ be given at the j^{th} iterate.

- I. (minimize) Solve the Riccati equation of (4.33) and the feedback control law of (4.34) for $P_{(j)}$ and $K_{(j)}$, respectively.

$$P_{(j)} = F^T P_{(j)} F - F^T P_{(j)} H [R + H^T P_{(j)} H]^{-1} H^T P_{(j)} F + C^T C \quad (4.33)$$

$$K_{(j)} = [R + H^T P_{(j)} H]^{-1} H^T P_{(j)} F. \quad (4.34)$$

- II. (maximize) Solve (4.35) for $[F_{(j+1)}, H_{(j+1)}]$.

$$[F_{(j+1)}, H_{(j+1)}] = \arg \max_{\theta_{FH} \in \Omega} \mathbf{x}_0^T P \mathbf{x}_0 \quad (4.35)$$

subject to

$$(F - HK_{(j)})^T P (F - HK_{(j)}) - P + K_{(j)}^T R K_{(j)} + C^T C = 0.$$

- III. (check convergence) If $\|[F_{(j+1)}, H_{(j+1)}] - [F_{(j)}, H_{(j)}]\|$ is less than some desired tolerance, then stop. Otherwise, let $[F_{(j+1)}, H_{(j+1)}] \leftarrow [F_{(j)}, H_{(j)}]$ and continue at step I.

The algorithm may be initiated by selecting the nominal parameter values of Ω for $[F_{(0)}, H_{(0)}]$.

On The Nature Of The Solution Of Algorithm 3

From the previous sections dealing with the continuous time problem we saw that the examples provided had solutions on the boundary of the ellipsoid. Here we look at **Algorithm 3** for the discrete problem.

Consider the first-order problem where there is only uncertainty in the parameter of F and the parameter of H is constant. The uncertain parameter is simply a symmetric interval about some nominal value a_0 . The state-space equations are given by

$$x(k+1) = ax(k) + bu(k) \quad (4.36)$$

and

$$y(k) = x(k).$$

First consider the minimization portion of **Algorithm 3**. The Riccati equation of (4.33) for the solution to the control law is given by

$$p = a^2 p - \frac{a^2 b^2 p^2}{r + b^2 p} + 1,$$

which simplifies to a quadratic polynomial in p

$$b^2 p^2 + ((1 - a^2)r - b^2)p - r = 0.$$

Using the positive solution to this polynomial for any value of a in the corresponding uncertainty region, we know from (4.34) that

$$K = \frac{abp}{r + b^2p}.$$

Thus K is a scalar which may be positive or negative depending on the the values of a and b . Next, consider the maximization portion of Algorithm 3. Equation (4.35) reduces to

$$a = \arg \max_{a,p} px_0^2 \quad (4.37)$$

subject to

$$(a - a_0)^2 \gamma \leq 1$$

and

$$((a - bK)^2 - 1)p + rK^2 + 1 = 0. \quad (4.38)$$

Solving (4.38) for p we arrive at

$$p = \frac{1 + rK^2}{1 - (a - bK)^2}.$$

If we assume that K stabilizes (4.36) for all values of a , we see that $|(a - bK)|$ is always less than 1. Since $r > 0$ we note that $p > 0$ and is finite. Next, if we substitute p above into (4.37) we obtain

$$a = \arg \max_a \frac{(1 + rK^2)x_0^2}{1 - (a - bK)^2}$$

which is equivalent to

$$a = \arg \max_a (a - bK)^2.$$

The function being maximized is a quadratic function. Furthermore, since we have assumed that K stabilizes (4.36) over all a then we know that $|(a - bK)| < 1$ which implies $(a - bK)^2$ is a convex function. Hence, the solution resides on the boundary [73, Theorem 3, page 119].

As expected, the existence of the solution to the maximization problem relies on the condition that K stabilizes (4.36) for all values of a . A sufficient condition for the existence of the maximization problem is shown in **Theorem 4.3**.

Theorem 4.3 (Existence of Maximization in Algorithm 3): *If the gain K given in (4.34) stabilizes all plants $\theta_{FH} \in \Omega$ and furthermore, for all plants $\theta_{FH} \in \Omega$ it is true that $\sigma_M(F - HK) < 1$ (σ_M denotes maximum singular value) then*

$$\max_{\theta_{FH} \in \Omega} x_0^T P x_0$$

subject to

$$(F - HK)^T P (F - HK) - P + K^T R K + C^T C = 0$$

exists.

Before proving this theorem, we shall make use of the following lemmas.

Lemma 4.1 *For any $n \times n$ matrices P and Y such that $P > 0$, $Y \geq 0$, the following inequalities are satisfied:*

$$\lambda_m(Y) \text{tr}(P) \leq \text{tr}(PY) \leq \lambda_M(Y) \text{tr}(P)$$

where λ_m and λ_M denote the minimum and maximum eigenvalues, respectively.

Lemma 4.1 was taken from [74] and the proof can be found in [75].

Lemma 4.2 *For the discrete Lyapunov equation $X^T P X - P + Z = 0$ and $Z \geq 0$, the following inequality is true*

$$\frac{\text{tr}(Z)}{1 - \lambda_m(X X^T)} \leq \text{tr}(P) \leq \frac{\text{tr}(Z)}{1 - \lambda_M(X X^T)}.$$

The proof of **Lemma 4.2** is given in [76].

Proof: (Theorem 4.3) Making use of the fact $\mathbf{x}_0^T P \mathbf{x}_0 = \text{tr}(P \mathbf{x}_0 \mathbf{x}_0^T)$ and Lemma 4.1 where $Y \triangleq \mathbf{x}_0 \mathbf{x}_0^T$, we see that

$$\max_{\theta_{FH} \in \Omega} \mathbf{x}_0^T P \mathbf{x}_0$$

subject to

$$(F - HK)^T P (F - HK) - P + K^T R K + C^T C = 0$$

is less than or equal to

$$\max_{\theta_{FH} \in \Omega} \text{tr}(P) \lambda_M(Y)$$

subject to

$$(F - HK)^T P (F - HK) - P + K^T R K + C^T C = 0.$$

Using the results of Lemma 4.2 and letting $X \triangleq F - HK$ and $Z \triangleq K^T R K + C^T C$ we obtain the following.

$$\lambda_M(Y) \max_{\theta_{FH} \in \Omega} \text{tr}(P)$$

subject to

$$X^T P X - P + Z = 0$$

is less than or equal to

$$\lambda_M(Y) \max_{\theta_{FH} \in \Omega} \frac{\text{tr}(Z)}{1 - \lambda_M(X X^T)}.$$

Since $\sigma_m(X) \leq \min_i |\lambda_i(X)| \leq \max_i |\lambda_i(X)| \leq \sigma_M(X)$ for $i = 1, 2, \dots, n$ [77], where the definition of singular value is given by $\sigma \triangleq \sqrt{\lambda(X X^T)}$, we see that

$$\max_{\theta_{FH} \in \Omega} \frac{\text{tr}(Z)}{1 - \lambda_M(X X^T)} \equiv \max_{\theta_{FH} \in \Omega} \frac{\text{tr}(Z)}{1 - \sigma_M^2(X)}.$$

Thus, if $\sigma_M(F - HK) < 1 \quad \forall \quad \theta_{FH} \in \Omega$ then the original maximization problem has a bounded solution. Hence, a sufficient condition for the existence of solution to the maximization problem of Algorithm 3 has been shown. $\square$

To show for a given K that it stabilizes all plants $\theta_{FH} \in \Omega$ is not a trivial matter and is left for future research. **Theorem 4.3** does not tell us where the solution resides and this is a subject of continuing research.

Set-membership Control Problem

Following is an example of set-membership identification and control problem. The ellipsoid bounding algorithm of [62] is used for set-membership identification. The example is in discrete-time; therefore, we use **Algorithm 3** for the robust control design. The states of the system are estimated using a Kalman filter. Implementation of a set-membership control problem raises an interesting question when one is employing a Kalman filter for state estimation; which plant within the uncertainty region does one select for use in the Kalman filter? For this example we take the parameters of the center of the ellipsoid for use in the Kalman filter. The reasoning behind this selection is that the center of the ellipsoid is "likely" closer to the actual plant than the worst-case plant associated with the robust control design and will likely offer a better estimate of the true system states.

Second Order Servo Example Consider the controlled plant (this example is found in [78, page 385])

$$\mathbf{x}(k+1) = \begin{bmatrix} 1.5 & 1 \\ -0.5 & 0 \end{bmatrix} \mathbf{x}(k) + \begin{bmatrix} 1 \\ 0.6 \end{bmatrix} [u(k) + w(k)]$$

$$y(k) = [1 \ 0] \mathbf{x}(k) + v(k)$$

where $w(k)$ and $v(k)$ are independent Gaussian noise disturbances. The object of this servo control is to determine an optimum controller

$$u(k) = -K^T \mathbf{x}(k) + Vr(k)$$

minimizing the cost function

$$J = \sum_{k=1}^{\infty} [\Psi^T(k) Q \Psi(k) + R(u(k) - u_{ss})^2]$$

where

$$\Psi^T(k) = [x_1(k) \ x_2(k) \ r(k)],$$

$$Q = \begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix},$$

$$u_{ss} = u(\infty),$$

and

$$R = 1.$$

The command variable $r(k)$ may take the values $(-1, 0, 1)$. The sampling period $T = 1$. The variable V is a scaling factor determined from the condition of zero steady-state control error. The scaling factor is adjusted depending on the gain vector K used for state feedback control. We note that this servo example is unusual in the way we are using the variable V for adjusting the steady-state error. A more popular method is to remove the gain block V and introduce an integrator into the forward path of the system (see, e.g., [46, page 850]). In order to keep the example simple and to follow the example as given in [78, page 385], we did not introduce the integrator.

The states are estimated using a Kalman filter. The associated equations are given below and can be found in numerous sources, see for example [46, 78]. We use F_0 and H_0 to denote the nominal plant given by the center of the bounding ellipsoid. Using steps (a) and (b) below $\hat{x}(k+1|k+1)$ can be solved given the Kalman gain $K_e(k+1)$ and the initial estimate $\hat{x}(k|k)$.

$$(a) \ \hat{x}(k+1|k) = F_0 \hat{x}(k|k) + H_0 u(k),$$

$$(b) \hat{x}(k+1|k+1) = \hat{x}(k+1|k) + K_e(k+1)[y(k+1) - C\hat{x}(k+1|k)].$$

For an initial Riccati matrix $P_e(k|k)$ the Kalman gain $K_e(k+1)$ is solved using steps (c) and (d) below. The Riccati matrix P_e is updated in step (e).

$$(c) P_e(k+1|k) = F_0 P_e(k|k) F_0^T + H_0 Q_e H_0^T,$$

$$(d) K_e(k+1) = P_e(k+1|k) C^T [C P_e(k+1|k) C^T + R_e]^{-1},$$

$$(e) P_e(k+1|k+1) = [I - K_e(k+1)C] P_e(k+1|k).$$

The adjustable weights which are the dual to the weights seen in the linear quadratic cost function are R_e and Q_e . These weights are often set to the approximate values of the variance of the measurement noise $v(k)$ and process noise $w(k)$, respectively. The Kalman gain is proportional to Q_e/R_e and as such R_e and Q_e are often adjusted to obtain better state estimation. Sophisticated methods such as *Loop Transfer Recovery* can be employed for adjusting the Kalman weighting factors [44].

A diagram showing the set-membership robust control scheme is shown in Figure 23.

For our example the process noise $w(k)$ is negligible; the measurement noise $v(k)$ has a variance of 0.01 and mean of 0.0. When calculating the controller gain K we make use of the fact that the solution is invariant to the elements of the third column of Q above. The solution of the Riccati equation of (3.18) may be carried out with the simpler matrix

$$Q = \begin{bmatrix} 1 & 0 \\ 0 & 0 \end{bmatrix}.$$

Using the actual values of the plant the solution of (3.20) gives the LQR gain

$$K = [0.7886 \quad 0.6119]^T.$$

The scale factor V in the control equation is computed from the condition of the zero steady-state control error. For a value of k approaching infinity it holds that

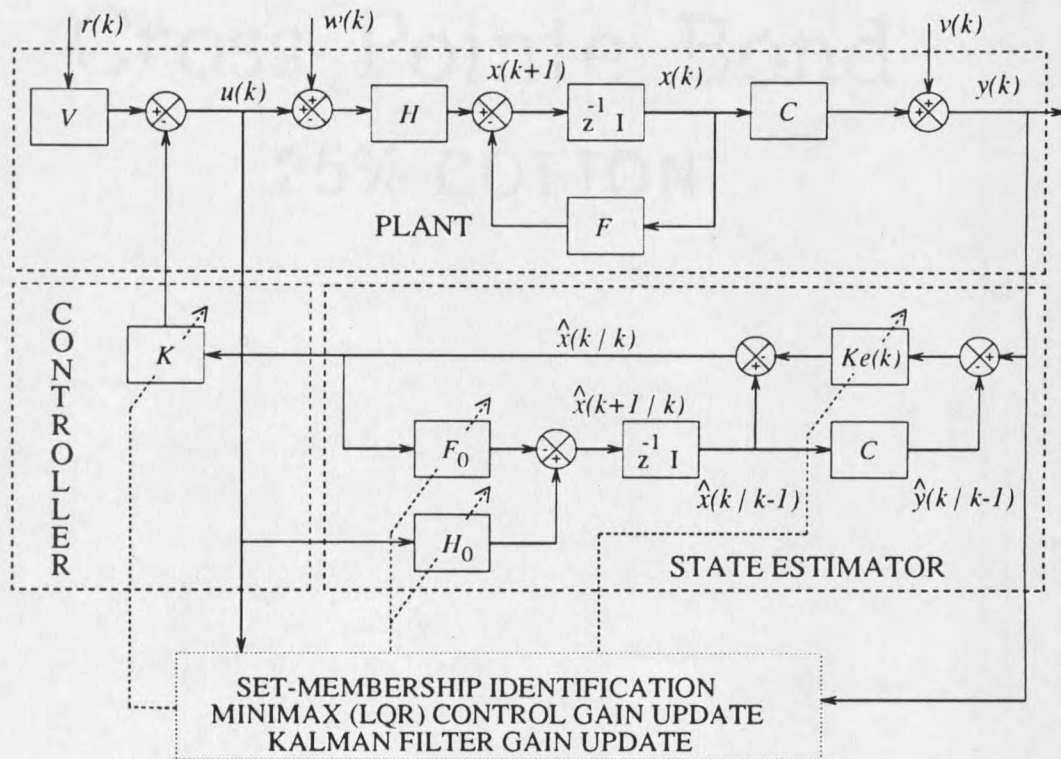


Figure 23. Set-membership adaptive control detailed block diagram.

parameter	lower	nominal	upper
$-a_1$	-2.0489	-1.4958	-0.9428
$-a_2$	-0.0451	0.4952	1.0355
b_1	0.2991	0.9610	1.6230
b_2	-0.0876	0.6156	1.3187

Table 7. Interval bounds on parameters of Example 1 using OVE algorithm.

$\mathbf{x}(k+1) = \mathbf{x}(k)$ and $x_1(k) = y(k) = r(k)$. Allowing $r(k)$ to take on the value of 1, the following set of equations can be solved for V and the steady-state value for $x_2(k)$. Let $Fcl = F - HK^T$ be the 2×2 closed-loop control matrix with elements $fcl_{1,1}$, $fcl_{1,2}$, $fcl_{2,1}$ and $fcl_{2,2}$, and let $H^T = [b_1 \ b_2]$. The solution to V and $x_2(k)$ in steady-state is given by

$$\begin{bmatrix} V \\ x_2(k) \end{bmatrix} = \begin{bmatrix} b_1 & fcl_{1,2} \\ b_2 & fcl_{2,2} - 1 \end{bmatrix}^{-1} \begin{bmatrix} 1 - fcl_{1,1} \\ -fcl_{2,1} \end{bmatrix}. \quad (4.39)$$

Using actual values of the plant we find that $V = 0.4826$ and $x_2(k) = -0.5000$. The initial starting value of $\mathbf{x}$ for calculation of the final cost was taken as a random sample from the Kalman filter estimates of the plant states before the feedback loop was closed. The initial starting value of $\mathbf{x}$ was also used in **Algorithm 3** for the worst-case gain calculation. In the first case, system identification was not used and actual parameter values of the plant in the Kalman filter were used for state-estimation. The initial starting value of the states was $\mathbf{x}(0) = [0.2325 \ 0.8409]$. The estimated cost using actual plant values is $J = 0.8118$. The bound on the noise sequence $v(k)$ is $v_{max} = 0.328229$ which was used in the ellipsoid bounding technique of Cheung, et al., referred to as the OVE algorithm. The input probing signal was adjusted to obtain a signal to noise ratio at the output of approximately 65 dB. Using the OVE algorithm we determined a bounding ellipsoid which had an associated bounding interval given in Table 7. After 9 iterations of **Algorithm 3** the LQR gain was found to be

$$K = [1.2852 \quad 1.0291]^T$$

and the worst-case LQR cost was $J = 2.5627$. To calculate V and the steady-state value of $x_2(k)$ we defined $F_{cl} \triangleq F_0 - H_0 K^T$ and $H_0 \triangleq [b_1 \ b_2]$, where F_0 and H_0 are the appropriate matrices using the nominal values given in Table 7. The solution to V and $x_2(k)$ in steady-state is given by (4.39) and the values are $V = 0.7749$ and $x_2(k) = -0.4955$. The results for this servo example can be seen in Figure 24.

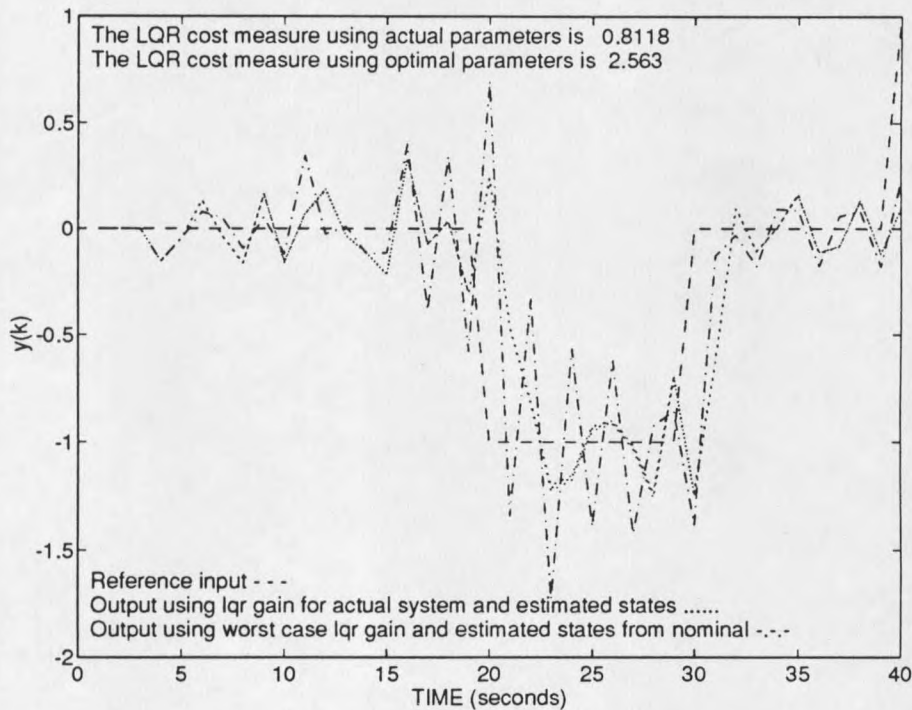


Figure 24. Servomechanism problem for standard LQG and set-membership control.

Discussion of Results

Prior to the development of the minimax algorithms of Chapter 4 the statement of the problems are established, and in particular we assume state-feedback control. In Chapter 3 state-feedback control is shown to be the necessary type of control to reach the optimal minimax solution. It is shown in **Theorem 4.1** that if a solution exists for the minimax problem, that in the case of nonunique saddle point solutions, the associated costs will be the same and that any one of the solutions may be used in the final design. Two algorithms are developed for the continuous-time problem where the uncertainty on the model parameters is in the form of an ellipsoidal domain. The algorithms are equivalent yet differ in the choice of Lyapunov equations used. While **Algorithm 2** more naturally follows from the original problem statement, we choose to use **Algorithm 1** due to the more commonly used measure of final cost for the LQ problem. In practice it can be seen that either algorithm gives the same result, however no examples are provided in this chapter. It is noted that there is a dependence on the initial conditions of the systems. This dependence on $\mathbf{x}_0$ can be removed if we start with the assumption that $\mathbf{x}_0$ is a random vector with known mean $\bar{\mathbf{m}}$ and covariance V and the objective in (4.1) is an expectation over $\mathbf{x}_0$. In that case, $\mathbf{x}_0 \mathbf{x}_0^T$ may be replaced by $V + \bar{\mathbf{m}} \bar{\mathbf{m}}^T$. Another alternative is to use the worst initial condition of unit magnitude as in [69]. Since the solution is not guaranteed to be a global saddle point a method is offered to divide the uncertainty region into a grid and methodically check to determine the nature of the solution.

In the examples provided it is shown that the maximization portion of the algorithms are performed over a region that is not necessarily convex. Because the region is not convex we are unable to make definitive remarks as to the existence and

location of the solution. In the low-order examples using **Algorithm 1** we find that the solutions reside on the boundary of the uncertainty domain. We also show that it is necessary for all plants in the uncertainty region to be at least stabilizable. This necessary condition is shown in **Example 2**.

For the discrete-time problem, **Algorithm 3** is developed along the lines as was **Algorithm 1**. The first-order problem, where the uncertainty is only in one parameter, namely a of the state matrix F , is reviewed for analytical purposes. In this analysis, it is shown that the sufficient condition for the existence of solution to the maximization portion in **Algorithm 3** is simply that K stabilizes the uncertain plant over all possible values of a . We carry this analysis further to the generalized higher-order case in **Theorem 4.3**. This analysis provides necessary conditions for existence of solution. In particular, **Theorem 4.3** shows that if the maximum singular value of the closed-loop system is less than 1 for all plants in the uncertainty set, the maximization step of **Algorithm 3** is bounded, and therefore, the solution exists.

Finally, Chapter 4 concludes in a complete example of Set-membership control. A second-order servo example is provided where the system is considered a *black box*. The OVE algorithm of Cheung [62] is implemented for system identification. This algorithm provides the nominal model with parametric uncertainty in the form of an ellipsoid. Since the system is considered to be a *black box* we employ a Kalman filter for state estimation. The nominal model from the OVE algorithm is used in the Kalman filter. The complete uncertain model of the OVE algorithm is used in **Algorithm 3** to arrive at an optimal control gain K . Once this gain is computed the feedback loop is closed and the reference signal for tracking is allowed to change from discrete values of -1, 0 and 1. The system remains stable and the worst performance of the system is shown by way of time simulation.

CHAPTER 5

CONCLUSION

A review of various identification methods is given in Chapter 2. Characteristics of each method were discussed to establish a baseline for the method based on Prony signal analysis. The method of identification developed in Chapter 2 is classified as a Prony method because it has two main phases, paralleling the original Prony signal analysis method; in which eigenvalue estimates are generated in the first phase, and residue estimates are calculated in the second phase. Both phases involve linear least-squares solutions that can be obtained with high accuracy using singular value decomposition techniques. The detailed steps used in the two Prony phases of Chapter 2 differ considerably from those of previous Prony-based methods because of the sampled-and-held nature of the input signal.

An interesting feature of including the sample-and-hold on the input is that the resulting second Prony phase is actually simplified from the corresponding second Prony phase of [20]. A second feature is that of establishing an estimate of initial conditions. The striking difference of the Prony method from those reviewed is the fact that modes of the system are estimated first. The user may be able to use prior knowledge of the system modes to eliminate extraneous modes from the set. This feature allows for reduced numerical calculations and potential for more accurate residue fitting.

The final feature of the *discrete Prony* method is the use of a generalized class of input probing signals that are sampled and held. Prior to being sampled and held, the input signal is piecewise continuous and is characterized by input eigenvalues and

input residues between points of discontinuity. The sample period T is the same as that used to collect the sampled system input and output data, which lends itself well to digitally controlled systems. By appropriate selection of the input signal, its energy can be allocated to particular frequency ranges of interest with regard to transfer function identification. Most other methods utilize a pseudo-random input signal that essentially has much of its energy at frequencies that are outside the bandwidth of the system being identified. This type of signal will often not adequately excite some modes of the system.

Examples show that with the addition of measured output noise, the approach developed in this dissertation can generate fairly good estimates of initial condition residues, transfer function residues, and system eigenvalues. Example 3 of Chapter 2 shows that the *discrete Prony* method performs relatively well under colored noise environments as compared with other methods. The last example demonstrates the model obtained from *discrete Prony* identification and used for LQG state-feedback control provides significant damping while the power system experiences a transmission line fault. In practice, it was found that higher order models obtained from *discrete Prony* identification were more difficult to utilize in this non-adaptive LQG state-feedback configuration.

Chapter 3 served to introduce two minimaximization problems. The review of linear quadratic control theory provided the necessary tools to establish the statement of the problem. Motivation for the robust control problems of this thesis is primarily due to the efforts of Lau, Fogel and Cheung [32, 61, 62] in the area of system identification and the efforts of Lau and Mills [72, 68] in the area of set-membership control. From standard results in linear quadratic control theory, it is shown that in order to determine the saddle point solution, the control strategy for the minimaximization problem must be LQR linear state-feedback. The problems are different from those of

Lau and Mills in that the uncertain parameters are the parameters of the state-space matrices A and B and are allowed to vary within a bounded ellipsoidal domain. The ellipsoidal domain of interest may be determined through an identification scheme based on the method of Cheung [62].

In Chapter 4, it is shown in **Theorem 4.1** that if a solution exists for the minimax problem, that in the case of nonunique saddle point solutions, the associated costs will be the same and that any one of the solutions may be used in the final design. Two algorithms are developed for the continuous-time problem where the uncertainty on the model parameters is in the form of an ellipsoidal domain. One algorithm is provided for the discrete-time problem.

It is noted that there is a dependence on the initial conditions of the systems. This dependence on $\mathbf{x}_0$ can be removed if we start with the assumption that $\mathbf{x}_0$ is a random vector with known mean $\bar{\mathbf{m}}$ and covariance V and the objective in (4.1) is an expectation over $\mathbf{x}_0$. In that case, $\mathbf{x}_0 \mathbf{x}_0^T$ may be replaced by $V + \bar{\mathbf{m}} \bar{\mathbf{m}}^T$. Another alternative is to use the worst initial condition of unit magnitude as in [69]. Since the solution is not guaranteed to be a global saddle point, a method is offered to divide the uncertainty region into a grid and methodically check to determine the nature of the solution.

In the examples provided it is shown that the maximization portion of the algorithms are performed over a region that is not necessarily convex. Because the region is not convex, we are unable to make definitive remarks as to the existence and location of the solution. By way of example, we see that the solutions tend to reside on the boundary of the uncertainty domain.

Concerning the discrete-time algorithm we look in more detail at the existence of solution to the maximization portion in **Algorithm 3**. The first-order problem, where the uncertainty is only in one parameter, namely a of the state matrix F ,

is reviewed for analytical purposes. In this analysis, it is shown that the sufficient condition for the existence of solution to the maximization portion in **Algorithm 3** is simply that K stabilizes the uncertain plant over all possible values of a . We carry this analysis further to the generalized higher order case in **Theorem 4.3**. This analysis provides a sufficient condition for existence of solution. In particular, **Theorem 4.3** shows that if the maximum singular value of the closed-loop system is less than 1 for all plants in the uncertainty set, the maximization step of **Algorithm 3** is bounded, and therefore, a solution exists.

Chapter 4 concludes in a complete example of Set-membership control. The problem considered is a second-order servo control system. The OVE algorithm of Cheung [62] is implemented for system identification. We employ a Kalman filter for state estimation using the nominal values from the OVE identified model. The complete uncertain model of the OVE algorithm is used in **Algorithm 3** to arrive at an optimal control gain K . Once this gain is computed the feedback loop is closed and the reference signal for tracking is allowed to change from discrete values of -1, 0 and 1. The system remains stable and the worst performance of the system is shown by way of time simulation.

CHAPTER 6

FUTURE RESEARCH

During the course of this research, some problems have not been fully addressed. This, of course, is the nature of research. Following is a list of possible areas for future research in which significant contributions still can be made:

1. The discrete form of Prony identification assumes no repeated eigenvalues in the system. While some may consider the likelihood of a real dynamical system having repeated eigenvalues small, it is still worthwhile investigating changes to the algorithm to account for this possibility.
2. The effects of system noise on the *discrete Prony* method presented remains to be analytically examined. There is a foundation of previous work (e.g., [59, 7, 10, 11, 12, 60, 13]) to build on. One may pay particular attention to the higher-order statistical approaches as presented in [13]. The methods of higher-order statistics found in [13] focus more on the signal analysis problem; it would be worthwhile to incorporate these methods in system identification.
3. The algorithms presented in Chapter 4 provide a foundation to work on for this type of robust control problem. While a sufficient condition for the solution to the maximization problem has been stated, it becomes quite obvious that the far more difficult problem is that of developing a technique for determining whether or not a given state-feedback gain K stabilizes all plants in the uncertainty domain.

Methods of stability analysis based on Kharitonov's theorem [79] have been developed for the continuous-time problems where the uncertainty is in the form

of bounds on the parameters of characteristic polynomials in Laplace variable s . In order to describe Kharitonov's Theorem for robust stability, we first define four fixed polynomials associated with an interval polynomial family $\mathcal{P}$.

Definition 6.1 (Kharitonov Polynomials): *Associated with the interval polynomial $p(s, a) = \sum_{i=0}^n [a_i^-, a_i^+] s^i$ are the four fixed Kharitonov Polynomials,*

$$K_1(s) = a_0^- + a_1^- s + a_2^+ s^2 + a_3^+ s^3 + a_4^- s^4 + a_5^- s^5 + a_6^+ s^6 + \dots;$$

$$K_2(s) = a_0^+ + a_1^+ s + a_2^- s^2 + a_3^- s^3 + a_4^+ s^4 + a_5^+ s^5 + a_6^- s^6 + \dots;$$

$$K_3(s) = a_0^+ + a_1^- s + a_2^- s^2 + a_3^+ s^3 + a_4^+ s^4 + a_5^- s^5 + a_6^- s^6 + \dots;$$

$$K_4(s) = a_0^- + a_1^+ s + a_2^+ s^2 + a_3^- s^3 + a_4^- s^4 + a_5^+ s^5 + a_6^+ s^6 + \dots;$$

We now state the celebrated theorem of Kharitonov [80] (the proof can be found in either [80] or [79]).

Theorem 6.1 (Kharitonov, 1978): *An interval polynomial family $\mathcal{P}$ with invariant degree is robustly stable if and only if its four Kharitonov polynomials are stable.*

If a comparable method of the Kharitonov analysis for the discrete-characteristic polynomial $p(z, a) = \sum_{i=0}^n [a_i^-, a_i^+] z^i$ were available, we could simply bound the ellipsoidal domain with a multi-dimensional box before attempting the maximization procedure of **Algorithm 3**. This approach would obviously be conservative but would certainly provide the sufficient condition for existence of solution as stated in **Theorem 4.3**. Unfortunately, as shown by Barmish [79, page 219], Kharitonov's Theorem does not generalize to the discrete case. Hence, this may or may not be an avenue of research for further investigation into stability analysis of gain K over the uncertainty region. Other results and references stated in [79] may provide the necessary tools for further investigation.

Another possible approach may be to consider the actual computation of

$$\max_{\theta_{FH} \in \Omega} \frac{\text{tr}(K^T R K + C^T C)}{1 - \sigma_M^2(F - HK)}.$$

This is equivalent to considering

$$\max_{\theta_{FH} \in \Omega} \sigma_M(F - HK).$$

4. The OVE identification method along with the discrete minimax algorithm of Chapter 4 were unsuccessfully applied to the same problem outlined in Chapter 2, Example 4. The minimax algorithm was unable to converge for a 4th-order uncertainty model obtained from the OVE algorithm. Reasons for nonconvergence and possible solutions is a path for further investigation.
5. Another area of research is to follow the uncertainty modeling approach of Lau [36] where, prior to identification, a model of unstructured (nonparametric) uncertainty is included. This approach to uncertainty modeling is widely accepted for robust control [29]. The minimax problem will have to account for both structured and unstructured uncertainty. This control problem is sometimes referred to as mixed robust control, see for example [81, 82].
6. Robust identification is a fairly new and emerging area of research. Although this subject is only discussed in this thesis, it promises to have considerable developments in the near future. Other methods of identification that provide different types of parametric uncertainty regions may be investigated.

REFERENCES CITED

- [1] L. Ljung, *System Identification: Theory for the User*. Englewood Cliffs, New Jersey: Prentice-Hall Inc., 1987.
- [2] G. R. B. deProny, "Essai experimentale et analytique," *J. l'Ecole Polytech (Paris)*, vol. 1, pp. 24-76, 1795.
- [3] A. J. Poggio, M. L. V. Blaricum, E. K. Miller, and R. Mittra, "Evaluation of a processing technique for transient data," *IEEE Trans. on Antennas and Propagation*, vol. AP-26, Jan. 1978.
- [4] R. Kumaresan and D. W. Tufts, "Improved spectral resolution iii: efficient realization," in *Proceedings of the IEEE*, vol. 68, Oct. 1980.
- [5] D. W. Tufts and R. Kumaresan, "Singular value decomposition and improved frequency estimation using linear prediction," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. ASSP-30, Aug. 1982.
- [6] D. W. Tufts and R. Kumaresan, "Estimation of frequencies of multiple sinusoids: making linear prediction perform like maximum likelihood," in *Proceedings of the IEEE*, vol. 70, Sept. 1982.
- [7] R. Kumaresan and D. W. Tufts, "Estimating the parameters of exponentially damped sinusoids and pole-zero modeling in noise," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. ASSP-30, pp. 833-840, Dec. 1982.
- [8] D. W. Tufts, A. C. Kot, and R. J. Vaccaro, "The threshold effect in signal processing algorithms which use an estimated subspace," *SVD and Signal Processing, II, Algorithms, Analysis and Applications*, pp. 301-320, 1991.
- [9] L. L. Scharf, *Statistical Signal Processing: Detection, Estimation, and Time Series Analysis*. Reading, Massachusetts: Addison-Wesley Publishing Company, 1991.
- [10] R. Kumaresan, D. W. Tufts, and L. L. Scharf, "A Prony method for noisy data: choosing the signal components and selecting the order in exponential signal models," in *Proceedings of the IEEE*, vol. 72, pp. 230-233, Feb. 1984.
- [11] B. Porat and B. Friedlander, "On the accuracy of the Kumaresan-Tufts method for estimating complex damped exponentials," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. ASSP-35, pp. 231-235, Feb. 1987.
- [12] P. Barone, "Some practical remarks on the extended Prony's method of spectrum analysis," in *Proceedings of the IEEE*, vol. 76, pp. 284-285, March 1988.
- [13] C. K. Papadopoulos and C. L. Nikias, "Parameter estimation of exponentially damped sinusoids using higher order statistics," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. 38, pp. 1424 - 1435, Aug. 1990.

- [14] J. F. Hauer, "The use of Prony analysis to determine modal content and equivalent models for measured output power system response," in *IEEE Symposium on Eigenanalysis and Frequency Domain Methods for System Dynamic Performance*, no. 90TH0292-3 PWR, pp. 105-115, 1989.
- [15] J. F. Hauer, C. J. Demeure, and L. L. Scharf, "Initial results in Prony analysis of power system response signals," *IEEE Trans. on Power Systems*, pp. 80-89, Feb. 1990.
- [16] D. J. Trudnowski, J. R. Smith, T. A. Short, and D. A. Pierre, "An application of Prony methods in PSS design for multimachine systems," *IEEE Trans. on Power Systems*, pp. 118-126, 1991.
- [17] D. A. Pierre, D. J. Trudnowski, and J. F. Hauer, "Identifying linear reduced-order models for systems with arbitrary initial conditions using Prony signal analysis," *IEEE Trans. on Automatic Control*, pp. 831-835, June 1992.
- [18] P. A. Emmanuel, *Damping of Low Frequency Oscillations in an AC/DC Power System Using HVDC Converter Control*. PhD thesis, Montana State University, Bozeman, MT 59717, October 1991.
- [19] D. J. Trudnowski, M. K. Donnelly, and J. F. Hauer, "Advances in the identification of electro-mechanical system models using Prony analysis," *IEEE Trans. on Power Systems*, pending.
- [20] D. A. Pierre, J. R. Smith, D. J. Trudnowski, and J. W. Pierre, "General formulation of a Prony based method for simultaneous identification of transfer functions and initial conditions," in *Proc., IEEE Conf. on Decision and Control*, pp. 1686-1691, Dec. 1992.
- [21] H. Nyquist, "Regeneration theory," *Bell Syst. Tech. J.*, vol. 11, pp. 126-147, 1932.
- [22] H. W. Bode, *Network Analysis and Feedback Amplifier Design*. Princeton, NJ: Van Norstrand, 1945.
- [23] V. G. Boltyanskii, R. V. Gamkrelidze, and L. S. Pontryagin, "On the theory of optimal processes," *Doklady Akad. Nauk S.S.S.R.*, vol. 110, pp. 7-10, 1956.
- [24] L. S. Pontryagin, V. G. Boltyanskii, R. V. Gamkrelidze, and E. F. Mishchenko, *The Mathematical Theory of Optimal Processes*. (Authorized Translation from the Russian; K. N. Trirogoff, Translator; L. W. Neustadt, Editor). New York, New York: John Wiley & Sons, Inc., 1962.
- [25] R. E. Bellman, *Dynamic Programming*. Princeton, New York: Princeton University Press, 1957.

- [26] R. E. Kalman and J. E. Bertram, "General synthesis procedure for computer control of single and multi-loop linear systems," *Trans. AIEE*, vol. 77, no. II, pp. 602-609, 1959.
- [27] J. C. Doyle and G. Stein, "Multivariable feedback design: concepts for a classical modern synthesis," *IEEE Trans. on Automatic Control*, vol. 26, pp. 4-16, 1981.
- [28] J. C. Doyle, B. A. Francis, and A. R. Tannenbaum, *Feedback Control Theory*. New York, New York: Macmillan Publishing Company, 1992.
- [29] R. L. Daily, "Lecture notes for the workshop on H_∞ and μ methods for robust control," in *Proc., IEEE American Control Conf.*, May 1990.
- [30] D. C. McFarlane and K. Glover, *Robust Controller Design Using Normalized Coprime Factor Plant Descriptions*. Lecture Notes in Control and Information Sciences, Berlin, Germany: Springer-Verlag, 1990.
- [31] M. F. Cheung, S. Yurkovich, and K. M. Passino, "An optimal volume ellipsoid algorithm for parameter set estimation," *IEEE Trans. on Automatic Control*, vol. 38, pp. 1292-1296, Aug. 1993.
- [32] M. K. Lau, R. L. Kosut, and S. Boyd, "Parameter set estimation of systems with uncertain non-parametric dynamics and disturbances," in *Proc., IEEE Conf. on Decision and Control*, pp. 3162-3167, Dec. 1990.
- [33] R. L. Kosut, M. K. Lau, and S. P. Boyd, "Set-membership identification of systems with parametric and non-parametric uncertainty," *IEEE Trans. on Automatic Control*, vol. 37, pp. 929-941, July 1992.
- [34] D. P. Bertsekas and I. B. Rhodes, "Recursive state estimation for a set-membership description of uncertainty," *IEEE Trans. on Automatic Control*, vol. 16, pp. 117-128, April 1971.
- [35] M. Milanese and A. Vicino, "Optimal estimation theory for dynamic system with set membership uncertainty: an overview," *Automatica*, vol. 27, no. 6, pp. 997-1009, 1991.
- [36] M. K. Lau, *Set Membership Approach to System Identification and Control Design*. PhD thesis, Stanford University, Stanford, CA 94305, March 1993.
- [37] M. K. Lau, S. Boyd, R. L. Kosut, and G. F. Franklin, "Robust control design for ellipsoidal plant set," in *Proc., IEEE Conf. on Decision and Control*, no. W1-10, pp. 291-296, Dec. 1991.
- [38] A. Friedman, *Differential Games*. New York, New York: John Wiley & Sons, Inc., 1971.

- [39] A. E. Bryson, Jr. and Y. Ho, *Applied Optimal Control: Optimization, Estimation, and Control*. Washington, DC: Hemisphere Publishing Corporation, 1975.
- [40] T. Başar and P. Bernhard, *H^∞ -Optimal Control and Related Minimax Design Problems*. Boston, Massachusetts: Birkhäuser, 1991.
- [41] I. Sadighi, *System Identification Methods for Adaptive Control*. PhD thesis, Montana State University, Bozeman, MT 59717, Oct. 1989.
- [42] T. Soderstrom and P. Stoica, *System Identification*. Englewood Cliffs, New Jersey: Prentice-Hall Inc., 1989.
- [43] R. E. Kalman, "On the general theory of control systems," in *Proc. First IFAC Congress, Moscow*, vol. 1, pp. 481-492, 1960.
- [44] J. M. Maciejowski, *Multivariable Feedback Design*. Reading, Massachusetts: Addison-Wesley Publishing Company, 1989.
- [45] P. K. Sinha, *Multivariable Control: An Introduction*. Electrical Engineering and Electronics/19, New York, New York: Marcel Dekker, Inc., 1984.
- [46] K. Ogata, *Discrete-Time Control Systems*. Englewood Cliffs, New Jersey: Prentice-Hall Inc., 1987.
- [47] I. Kamwa, R. Grondin, J. Dickinson, and S. Fortin, "A minimal realization approach to reduced-order modeling and modal analysis for power system response signals," in *IEEE/PES Summer Meeting*, no. 92SM490-3 PWRS, July 1992.
- [48] W. E. Larimore, "Canonical variate analysis in identification, filtering, and adaptive control," in *Proc., IEEE Conf. on Decision and Control*, Dec. 1990.
- [49] H. Akaike, "A new look at the statistical model identification," *IEEE Trans. on Automatic Control*, vol. AC-19, pp. 716-722, Dec. 1974.
- [50] J. V. Candy, T. E. Bullock, and M. E. Warren, "Invariant description of the stochastic realization," *Automatica*, vol. 15, pp. 493-495, 1979.
- [51] M. Moonen, B. De Moor, L. Vandenberghe, and J. Vandewalle, "On- and off-line identification of linear state-space models," *Int. J. of Control*, vol. 49, no. 1, pp. 219-232, 1989.
- [52] W. Gawronski, F. Y. Hadaegh, and R. E. Scheid, "A Hankel singular value approach for identification of low level dynamics in flexible structures," in *Proc., IEEE Conf. on Decision and Control*, pp. 2575-2579, Dec. 1990.
- [53] M. Moonen and J. Vandewalle, "QSVD approach to on- and off-line state-space identification," *Int. J. of Control*, vol. 51, no. 5, pp. 1133-1146, 1990.

- [54] W. E. Larimore, "Automated and objective system identification based on canonical variables," in *Proc., IEEE Conference on Aerospace Control Systems*, May 1993.
- [55] R. R. Hocking and R. N. Leslie, "Selection of the best subset in regression analysis," *Technometrics*, vol. 9, pp. 531-540, Nov. 1967.
- [56] S. Kullback, *Information Theory and Statistics*. Mineola, New York: Dover, 1959.
- [57] D. J. Trudnowski, "Order reduction of large-scale linear oscillatory system models," in *IEEE/PES Winter Meeting*, no. 93WM209-7 PWRS, Feb. 1993.
- [58] F. Fatehi and D. A. Pierre, "The robustness of an advanced adaptive controller against system noise," in *Proc., IEEE Conf. on Decision and Control*, Dec. 1993.
- [59] T. L. Henderson, "Geometric methods for determining system poles from transient response," *IEEE Trans. on Acoustics, Speech, and Signal Processing*, vol. ASSP-29, pp. 982-988, Oct. 1981.
- [60] B. J. Bujanowski, J. W. Pierre, S. Hietpas, T. Sharpe, and D. A. Pierre, "A comparison of several system identification methods with applications to power systems," in *Proc., 36th Midwest Symposium on Circuits and Systems*, Aug. 1993.
- [61] E. Fogel, "System identification via membership set constraints with energy constrained noise," *Automatica*, vol. 24, no. 5, pp. 752-758, 1979.
- [62] M. Cheung, S. Yurkovich, and K. M. Passino, "An optimal volume ellipsoid algorithm for parameter set estimation," in *Proc., IEEE Conf. on Decision and Control*, Dec. 1991.
- [63] L. G. Khachian, "A polynomial algorithm in linear programming," Translated in: *Soviet Mathematics Doklady*, vol. 20, pp. 191-194, 1979.
- [64] D. Goldfarb and M. J. Todd, "Modifications and implementation of the ellipsoid algorithm for linear programming," *Mathematical Programming*, vol. 23, pp. 1-19, March 1982.
- [65] E. Fogel and Y. F. Huang, "On the value of information in system identification - bounded noise case," *Automatica*, vol. 18, no. 2, pp. 229-238, 1982.
- [66] J. J. D'Azzo and C. H. Houpis, *Linear Control System Analysis and Design*. New York, New York: McGraw-Hill Book Company, Inc., 1981.
- [67] S. Boyd, L. E. Ghaoui, E. Feron, V. Balakrishnan, and M. Grant, *Generalized Eigenvalue Minimization Problems Arising in Control Theory*. Stanford, CA: In preparation, Stanford University, 1993.

- [68] R. A. Mills, *Robust Controller and Estimator Design Using Minimax Methods*. PhD thesis, Stanford University, Stanford, CA 94305, March 1992.
- [69] L. E. Ghaoui, A. Carrier, and A. E. B. Jr., "Linear-quadratic minimax controllers," *AIAA J. of Guidance, Control and Dynamics*, vol. 15, pp. 953-961, 1992.
- [70] R. L. Kosut, M. Lau, and S. Boyd, "Identification of systems with parametric and non-parametric uncertainty," in *Proc., IEEE American Control Conf.*, no. FA11, pp. 2412-2417, 1990.
- [71] T. Başar, "A dynamic games approach to controller design: disturbance rejection in discrete time," in *Proc., IEEE Conf. on Decision and Control*, pp. 407-414, Dec 1989.
- [72] M. K. Lau, S. Boyd, R. L. Kosut, and G. F. Franklin, "A robust control design for FIR plants with parameter set uncertainty," in *Proc., IEEE American Control Conf.*, no. WA3, pp. 83-88, 1991.
- [73] D. G. Luenberger, *Introduction to Linear and Nonlinear Programming*. Reading, Massachusetts: Addison-Wesley Publishing Company, 1965.
- [74] B. H. Kwon, M. J. Youn, and Z. Bien, "On bounds of the Riccati and Lyapunov matrix equations," *IEEE Trans. on Automatic Control*, vol. 30, pp. 1134-1135, Nov. 1985.
- [75] M. Toda and R. V. Patel, "Bounds on estimation errors of discrete-time filters under modeling uncertainty," *IEEE Trans. on Automatic Control*, vol. 25, pp. 1115-1121, 1980.
- [76] T. Mori, N. Fukuma, and M. Kuwahara, "On the discrete Lyapunov matrix equation," *IEEE Trans. on Automatic Control*, vol. 27, pp. 463-464, April 1982.
- [77] T. Kailath, *Linear Systems*. Englewood Cliffs, New Jersey: Prentice-Hall Inc., 1980.
- [78] V. Strejc, *State Space Theory of Discrete Linear Control*. New York, New York: John Wiley & Sons, Inc., 1981.
- [79] B. R. Barmish, *New Tools for Robustness of Linear Systems*. New York, New York: Macmillan Publishing Company, 1994.
- [80] V. L. Kharitonov, "Asymptotic stability of an equilibrium position of a family of systems of linear differential equations," *Differentsial'nye Uravneniya*, vol. 14, pp. 2086-2088, 1978.

- [81] D. S. Bernstein and W. M. Haddad, "LQG control with an H_∞ performance bound: A Riccati equation approach," *IEEE Trans. on Automatic Control*, vol. AC-34, pp. 293-305, March 1989.
- [82] W. M. Haddad, D. S. Bernstein, and C. N. Nett, "Decentralized H_2/H_∞ controller design: the discrete-time case," in *Proc., IEEE Conf. on Decision and Control*, pp. 932-933, Dec. 1989.

APPENDICES

APPENDIX A
Identification Code

The following code is for all results presented in Chapter 2. The appendix is made up of four sets of code. The first set of code includes runcomp.m, id_plt.m, diff_model.m, diff_modes.m, diff_nn.m, diff_noise.m which allows the user to access either the *discrete Prony* or the *Moonen* code for identification comparison. The second set of code is comprised of pr1.m, pr1b.m, pr2.m, pr2b.m, pr3.m, prony_in.m, u_prony.m, get_prony_in.m, in_exmpl.m, rm_mode.m, xdisp_mode.m, make_obsoble.m, obsv.m, sys2obsv_can.m, and sys2obsv_can.m which is all of the *discrete Prony* identification code. The third set of code is for *Moonen* identification and is comprised of moon1.m, moon2.m, moon_in.m, u_moonen.m, and get_moon_in.m. The last set of code, comprised of u_modes.m, sysm6.m, discr.m, fftst.m, dir_stamp.m, file_stamp.m, and time.m, is common to both the *discrete Prony* and *Moonen* code.

The main code for conducting the comparison between *discrete Prony* identification and *Moonen* identification is runcomp.m. This code allows the user to select between either identification approach. All results, except for example 4 of Chapter 2 were arrived at by running runcomp.m. Example 4 code is given at the end of this appendix.

Each M-file has a preamble that provides enough information to understand and use the code.

runcomp.m

```
function []=runcomp(system,in_drive)
%+-----
% FILE:
%      runcomp.m      - This M-file runs a comparison between
%                      identification routines from Moonen and
%                      Hietpas(Prony).
%
% INPUT
%      system          - Prior to running the identification
%                      program the user is allowed to create an
%                      output for some system which is assumed
%                      to be in continuous state-space form, i.e.,
%                       $\dot{x}=ax+bu$ ,  $y=cx+du$ .
%                      Secondly, it is also assumed that the
%                      system has a second input where the noise
%                      can be injected. The first input is for
%                      identification probing and the second for
%                      noise.
%
%      in_drive        - The input control file is provided by the
%                      user at the start of the program or
%                      prompted by the program itself these items
%                      are not given at the onset of the program.
%
```

```

% EXAMPLES OF INPUT DRIVE CODE:
%
% As an example of an input file that generates the disturbance
% input to the system, consider the following "in_drive" file that
% generates a SINC type input:
%
%*** function uu = u_moonen(t,k,dt,m)
%*** %*****
%*** % INPUT:
%*** %
%*** %           t = time;
%*** %           k = time mult. factor.
%*** %           dt = time shift factor.
%*** %           m = a magnitude factor.
%*** % OUTPUT:
%*** %
%*** %           uu = the input control signal.
%*** %
%*** %           For the SINC input, k is the frequency adjustment
%*** %           knob.
%*** %           For k = 30 the bandwidth of u is 5 Hz, k=15 give a BW
%*** %           of 2.5 Hz.
%*** %           The center of the SINC input is located at dt seconds.
%*** %           If dt = 5, the peak of the SINC signal will occur at
%*** %           5 seconds. The peak magnitude of the input located at
%*** %           dt seconds is determined by the value m.
%*** %*****
%*** if (t-dt)==0
%***   uu=m;
%*** else
%***   uu=m*sin(k*(t-dt))/(k*(t-dt));
%*** end
%*** %*****
%
% This input file is for the Moonen algorithm.
% The discrete integration routine diff_nn.m will execute this
% input routine.
% Prior to executing the in_drive file, however, moon_in.m will
% prompt the user for k, dt and m.
%
% As a second example, which is necessary for the Prony method,
% consider the following input file, which is used for constructing
% a sum of damped and non-damped sine wave:
%
%*** function uu = u_prony(t,dd,mpi,cc,ss)

```

```

***% %*****
***% % INPUT:
***% %
***% %         t = time;
***% %         dd = vector of duration factors in seconds;
***% %         mpi = number of modes/interval;
***% %         cc = vector of amplitude factors;
***% %         ss = vector of mode factors.
***% %
***% % OUTPUT:
***% %
***% %         uu = the input control signal.
***% %
***% %         For a sine wave of 5 Hz, amplitude 1, for 2 sec.
***% %         then
***% %         dd=2; mpi=2; cc=[-i/2 i/2]; ss=[i*2*pi*5 -i*2*pi*5].
***% %         From Euler's identity:
***% %         sin(2*pi*5*t)=cc(1)*exp(ss(1)*t)-cc(2)*exp(ss(2)*t);
***% %
***% %*****
***% q=length(dd);
***% if t<=dd(q)
***%     ii=max(find(t>dd));
***%     if isempty(ii)
***%         m_i=0;
***%         ii=1;
***%     elseif ii<q
***%         m_i=0;
***%         for k=1:ii
***%             m_i=m_i+mpi(k);
***%         end
***%         ii=ii+1;
***%     else
***%     end
***%     if mpi(ii)~=0
***%         if ii==1
***%             ddd=0;
***%         else
***%             ddd=dd(ii-1);
***%         end
***%         uu=0.0;
***%         for k=m_i+1:m_i+mpi(ii)
***%             uu=uu+cc(k)*exp(ss(k)*(t-ddd));
***%         end
***%     else

```

```

***%      uu=0.0;
***%      end
***%      else
***%      uu=0.0;
***%      end
***%      %*****
%
% The discrete integration routine diff_nn.m will execute this
% input routine.
% Prior to executing the in_drive file, however, prony_in.m will
% prompt the user for dd, mpi, cc and ss.
%
% Once diff_nn.m is executed, diff_noise.m will use the probing
% input and generate the noisy signal for the second input and
% drive the system again.
%
% GLOBAL:
%      None.
%
% CALLS:
%      {system}.m
%      moon_in.m
%      prony_in.m
%      diff_nn.m
%      diff_noise.m
%      diff_model.m
%      id_plt.m
%      moon1.m
%      pr1.m
%
% DATA FILES WRITTEN:
%      {data_file}.mat - {data_file} is prompted to the user
%                       for data to be saved in.
%                       To see what is stored in this data
%                       file, just search on 'data_file'
%                       and 'eval(sv)'.
%      {data_file}_mod.mat - _mod is appended to {data_file} and
%                           the final fitted model is stored
%                           here.
%
% DATA FILES READ:
%      {data_file}.mat
%

```

```

% AUTHOR:
%       Steve Hietpas, January 1994
%
% SYNTAX:
% If your system file is labeled sys1.m and you desire Moonen
% identification, and the in_drive file is u_moonen.m, you would
% type at the Matlab prompt the following:
%
%       >> runcomp('sys1','u_moonen');
%
% If, however, you desired to do the Prony identification, and
% the input file is u_prony.m, you would type at the Matlab
% prompt the following:
%
%       >> runcomp('sys1','u_prony');
%
% The user is also given the opportunity to just enter the
% following at the
% the Matlab prompt.
%
%       >> runcomp;
%
% The user will then be prompted for the name of the two files,
% system and in_drive.
%+-----+
dia=0;
strg='Which type of identification, [1=Moonen] and [2=Prony] >> ';
id_type=input(strg)
if isempty(id_type)
    id_type=1;
end
ans=input('Do you want a diary? [n,N,<ret>=no] >> ','s');
if ~(isempty(ans) | ans=='n' | ans=='N')
    dia=1;
    strg='Enter diary name: [<ret>=id_compare.dia] >> ';
    diary_file=input(strg,'s');
    if isempty(diary_file)
        diary_file='id_compare.dia';
    end
    disp(['The diary file is: ',diary_file])
    if id_type~=2
        if exist(diary_file)
            strg='Append to the existing diary? [y,Y,<ret>=yes] >> ';

```

```

    ans=input(strg,'s');
    if ~(isempty(ans) | ans=='y' | ans=='Y')
        rmm=['!rm ',diary_file];eval(rmm)
    end
end
do_diary=['diary ',diary_file];eval(do_diary)
disp('*****')
disp('*          MOONEN IDENTIFICATION          *')
disp('*****')
else
    if exist(diary_file)
        strg='Append to the existing diary? [y,Y,<ret>=yes] >> ';
        ans=input(strg,'s');
        if ~(isempty(ans) | ans=='y' | ans=='Y')
            rmm=['!rm ',diary_file];eval(rmm)
        end
    end
    do_diary=['diary ',diary_file];eval(do_diary)
    disp('*****')
    disp('*          PRONY IDENTIFICATION          *')
    disp('*****')
    end
    [tim,dat]=time;
    dat=['The date is ',dat];disp(dat),disp(' ')
    tim=['The time is ',tim];disp(tim)
end
what
if nargin==0
    notdone=1;
    while notdone
        strg='Enter the m-file containing the system eq''s >> ';
        system=input(strg,'s');
        if exist(system)
            eval([system,',';'])
            notdone=0;
        else
            disp('That system m-file is nowhere to be found.')
            notdone=1;
        end
    end
end
notdone=1;
while notdone
    strg='Enter the m-file containing the input drive code >>.';

```

```

    in_drive=input(strg,'s');
    if exist(in_drive)
        notdone=0;
    else
        disp('That unput drive m-file is nowhere to be found.')
        notdone=1;
    end
end
elseif nargin==1
    strg='Enter the data file containing the input m-file >> ';
    in_drive=input(strg,'s');
end
notdone=1;
default_data='ex_uyt';
while notdone
    strg='Retrieve previous data, enter a data file name: [<ret>='];
    strg=[strg,default_data,'] >> ';
    data_file=input(strg,'s');
    if ~isempty(data_file)
        if exist([data_file,'.mat'])
            ld=['load ',data_file];eval(ld)
            notdone=0;
        else
            disp('That file does not exist.')
        end
    else
        data_file=default_data;
        if exist([data_file,'.mat'])
            ld=['load ',data_file];eval(ld)
        end
        notdone=0;
    end
end
end
sys=system;
in=in_drive;
disp(' '),disp('Start time will be set to 0. '),disp(' ')
if exist('FT')
    fprintf('The previous value of FT was: %g seconds \n',FT)
    fprintf('The previous value of DT was: %g seconds \n',DT)
    ans=input('Keep these? [y,Y,<ret>=yes] >> ','s');
    if ~(isempty(ans) | ans=='y' | ans=='Y')
        FT=input('Final Time = ');
        DT=input('Sampling Rate = ');
    end
end

```

```

end
else
    FT=input('Final Time = ');
    DT=input('Sampling Rate = ');
end
ld=['load ',sys];eval(ld)
eigA=eig(A);
[F,G]=discr(A,B,DT); %discretize the system.
eigF=eig(F);
var_svd_sys=' A B C D F G DT eigA eigF';
sv_dat=[var_svd_sys,' var_svd_sys'];
sv=['save ',sys,sv_dat];eval(sv);
if id_type~=2
    u_in='moon_in';
else
    u_in='prony_in';
end
uin=[u_in,('','','sys','','','in','','FT);'];eval(uin);
ld=['load ',u_in];eval(ld) %var_svd_in
%
diff1='diff_nn'; %difference equation solution w/o noise
dif=[diff1,('','','sys','','','u_in','','FT);'];eval(dif);
ld=['load ',diff1];eval(ld) %var_svd_nn
if id_type==2
    strg='Zero-value initial conditions? [n,N,<rtn>=no] >> ';
    ans=input('strg','s');
    if (isempty(ans) | ans=='n' | ans=='N')
        disp('The modes of the system are:')
        disp(eigA)
        if exist('ee')
            strg='The previous values for initial condition ';
            strg=[strg,'amplitudes were:'];
            disp(strg)
            disp(ee)
        end
        disp(' ')
        strg='New amplitudes of initial conditions ';
        strg=[strg,'[<rtn>=old values] >> '];
        ee_new=input(strg);
        if isempty(ee_new)
            ee_new=ee;
        end
        disp('Your choice for initial conditions is: ')
    end
end

```

```

    ee=ee_new
    len_eigA=length(eigA);
    Z=eigF;
    thet2=zeros(length(tsd),len_eigA);
    thet2(1,:)=ones(1,len_eigA);
    for p=2:length(tsd)
        z=Z.^p;z=z(:);z=z.';
        thet2(p,:)=z;
    end
    y_ini=thet2*ee(:);
    ysn=ysnn+y_ini;
else
    ee=[];
    y_ini=[];
end
else
    ee=[];
    y_ini=[];
end
diff2='diff_noise';
dif=[diff2,'('','',sys','', '',u_in','', '',diff1','',FT,y_ini);'];
eval(dif)
ld=['load ',diff2];eval(ld) %var_svd_noise
%
var_svd=[' sys in FT ee y_ini',var_svd_sys,var_svd_in]
var_svd=[var_svd,var_svd_nn,var_svd_noise];
sv_dat=[var_svd,' var_svd'];
what
strg=['Enter a {new} data file name [<ret>=',data_file,'] >> '];
data_file_new=input(strg,'s');
if isempty(data_file_new)
    data_file_new=data_file;
end
data_file=data_file_new;
disp(['The data file name for saving data to is: ',data_file])
sv=['save ',data_file,sv_dat];eval(sv)
ysd0=ysd2;
SNR=[];sgnro=[];
[rr,cc]=size(ysd2);
ans=input('Add noise to output signal? [n,N,<ret>=no] >> ','s');
if ~(isempty(ans) | ans=='n' | ans=='N')
    notdone=1;
    while notdone

```

```

strg='Retrieve noise from data file? [y,Y,<ret>=yes] >> ';
ans=input(strg,'s');
if (isempty(ans) | ans=='y' | ans=='Y')
    what
    fn=input('Enter a filename: >> ','s');
    ld=['load ',fn];eval(ld)
else
noise=input('Which noise type: (normal=1) or ([uniform=2]) >> ');
    nnmag_def=0.0001;
    strg='Enter the noise multiplication factor:';
    strg=[strg,'[def=',num2str(nnmag_def),'] >> '];
    nnmag=input(strg);
    if isempty(nnmag)
        nnmag=nnmag_def;
    end
    if noise~=1
        nn=rand(size(ysd0));
        nn=nn-0.5;
    else
        nn=randn(size(ysd0));
        [nnr,nncl]=size(nn);
        for k=1:nncl
            nnm=mean(nn(:,k));
            nn(:,k)=nn(:,k)-nnm;
        end
    end
    nn=nnmag*nn;
ans=input('Save this noise vector? [y,Y,<ret>=yes] >> ','s');
if (isempty(ans) | ans=='y' | ans=='Y')
    what
    fn=input('Enter a file name >> ','s');
    sv=['save ',fn,' nn'];eval(sv)
end
end
ysd2=ysd0+nn;
for kk=1:cc
    sgnro=20*log10(norm(ysnn)/norm(ysnn-ysd2));
    SNR=[' ',num2str(sgnro),','];
end
strg=['The SNR(s) is(are) ',SNR,' dB.'];disp(' '),disp(strg);
ans=input('Is the SNR suitable? [y,Y,<ret>=yes] >> ','s');
if (isempty(ans) | ans=='y' | ans=='Y')
    notdone=0;

```

```

        end
    end
else
    sgnro=1/0;
    SNR=[SNR,' ',num2str(sgnro),',','];
    strg=['The SNR(s) is(are) ',SNR,' dB.'];disp(' '),disp(strg)
end
var_svd=[var_svd,' SNR sgnro'];
sv_dat=[var_svd,' var_svd'];
sv=['save ',data_file,sv_dat];eval(sv)
if id_type~=2
    disp(' ')
    disp('The Hankel matrix size factor is typically >= to 5')
    strg='Hankel Matrix size factor (factor: #columns > #rows) >> ';
    g=input(strg);
    var_svd=[var_svd,' g'];
    sv_dat=[var_svd,' var_svd'];
    sv=['save ',data_file,sv_dat];eval(sv)
    moon1(data_file);
    disp('*****')
else
    pr1(data_file);
end
model_file=[data_file,'_mod'];
ld=['load ',model_file];eval(ld)
disp(' ')
disp(['The model data is stored in ',model_file]),disp(' ')
strg='The response of the model to the original input ';
strg=[strg,'now calculated:'];
disp(strg)
ysdm=diff_model(data_file);
var_svd_m=[var_svd_m,' ysdm'];
sv_dat=[var_svd_m,' var_svd_m'];
sv=['save ',model_file,sv_dat];eval(sv)
id_plt(data_file);
if dia==1
    diary off
end
disp('You have reached the end of runcomp.m')
%+++++
%
%           That is the end of runcomp.m
%+++++

```

id_plt.m

```

function []=id_plt(data_file)
%+++++
% FILE:
%     id_plt.m      - Primarily a plotting routine for Moonen or
%                   Prony identification.
%
% INPUT:
%     data_file     - Data file name, if left empty user will be
%                   queried for a file name.
%
% OUTPUT:
%     None.
%
% GLOBAL:
%     None.
%
% CALLS:
%     None.
%
% DATA FILES WRITTEN:
%     None.
%
% DATA FILES READ:
%     {data_file}.mat
%
% AUTHOR:
%     Steve Hietpas January, 1994
%
% REFERENCES:
%     None.
%
% SYNTAX:
%     id_plt('data_file');
%     or
%     id_plt
%+++++
disp('Running the function id_plt.m')
global fig_table
if nargin~=1
    what
    no_file_name=1;

```

```

while no_file_name
    data_file=input('Enter a data file name (uyt#) >> ','s');
    no_file_name=isempty(data_file);
end
end
model_file=[data_file,'_mod'];%
ld=['load ',data_file];eval(ld)
ld=['load ',model_file];eval(ld)
fftst(usd,tsd,1,'Y',data_file);
%
s1=['SNR of second input to probing input is ',num2str(sgnri),' dB'];
ss=str2mat(s1);
disp('Place 1 string of information on figure.')
tt=['Noise Input Disturbance to Generate Colored Noise on Output.'];
xx='TIME (seconds)';
yy='Magnitude u_second';
[tim,dat]=time;tim=['The time is ',tim];disp(tim)
figure
fig_table=[fig_table;gcf];
plot(tsd,usd2(:,2)),title(tt),xlabel(xx),ylabel(yy),
gtext(ss)
time_stamp,dir_stamp,file_stamp(data_file);
%
if ~exist('ysdm')
    ysdm=diff_model(data_file);
end
ordr=length(eigAm);
tt='Response of model and actual to input 1 (second input off)';
yy='y(t)';
s1=['SNR of second input to probing input is ',num2str(sgnri),' dB'];
s2a='MODEL OUTPUT .....';
s2b='ACTUAL OUTPUT ___';
s3=['Order of model is ',num2str(ordr)];
n2=norm(ysnn-ysdm)/(norm(ysnn));
snr5=20*log10(n2);
n2dif=['The normalized difference is: ',num2str(n2)];
n2dif=[n2dif,' (',num2str(snr5),' dB)'];
ss=str2mat(s2a,s2b,s1,s3,n2dif);
disp('Place 5 string of information on figure.')
[tim,dat]=time;tim=['The time is ',tim];disp(tim)
figure
fig_table=[fig_table;gcf];
plot(tsd,ysdm,': ',tsd,ysnn,'-'),title(tt),xlabel(xx),ylabel(yy),grid

```

```

gtext(ss)
time_stamp,dir_stamp,file_stamp(data_file);
%
sss=['load ',sys];eval(sss);
tt='Poles of system and model.';
s2=['Model(o) --- Actual(*)'];
xxx='Real axis';yyy='Imaginary axis';
[tim,dat]=time;tim=['The time is ',tim];disp(tim)
ss=str2mat(s2);
disp('Place 1 string of information on figure.')
[tim,dat]=time;tim=['The time is ',tim];disp(tim)
figure
fig_table=[fig_table;gcf];
plot(eigAm,'o'),hold on,plot(eigA,'*'),grid
title(tt),xlabel(xxx),ylabel(yyy),
gtext(ss)
time_stamp,dir_stamp,file_stamp(data_file);
hold off
%
strg='Drive model with variety of inputs? [n,N,<ret>=no] >> ';
ans=input(strg,'s');
if ~(isempty(ans) | ans=='n' | ans=='N')
%
    plt2=input('Do you want plotting? [y,Y,<ret>=yes] >> ','s');
    if (isempty(plt2) | ans=='y' | ans=='Y')
        plt2=1;
    else
        plt2=0;
    end
%
    ss1=['','','data_file','','',sys','','','u_modes',plt2'];
    ss2=['diff_modes(',ss1,')'];eval(ss2);
    load uuyytd
    p1=length(tsde);
    p1by2=p1/2;
    p12=(1:p1by2);
    p22=(p1by2+1:p1);
    tt='Response of Model and Actual to all modes of Model.';
    snr6a=y1toy2(tsde,y2,y1,tt,plt2,data_file);
    tsda=tsde(p12);y2a=y2(p12);y1a=y1(p12);
    snr6b=y1toy2(tsda,y2a,y1a,tt,plt2,data_file);
    tsda=tsde(p22);y2a=y2(p22);y1a=y1(p22);
    snr6c=y1toy2(tsda,y2a,y1a,tt,plt2,data_file);

```

```

%
md=[];
mdk=imag(eigA);
for jj=1:length(eigA)
    if mdk(jj)>eps
        ii=find(md==mdk(jj));
        if isempty(ii)
            md=[md;mdk(jj)];
        end
    end
end
mode=md;
len_mode=length(mode);
snr7a=zeros(len_mode,1);snr7b=snr7a;snr7c=snr7a;
for kj=1:len_mode
    md=mode(kj);
    ss1=['','','data_file','','','sys','','','u_modes',plt2,md'];
    ss2=['diff_modes('',ss1,'')'];eval(ss2);
    load uuyytd
    strg='Response of Model and Actual to mode: ';
    tt=[strg,num2str(md),' rad/sec..'];
    snr7a(kj)=y1toy2(tsde,y2,y1,tt,plt2,data_file);
    tsda=tsde(p12);y2a=y2(p12);y1a=y1(p12);
    snr7b(kj)=y1toy2(tsda,y2a,y1a,tt,plt2,data_file);
    tsda=tsde(p22);y2a=y2(p22);y1a=y1(p22);
    snr7c(kj)=y1toy2(tsda,y2a,y1a,tt,plt2,data_file);
    axis;
end
%
var_svd=[var_svd,' snr5 snr6a snr6b snr6c snr7a snr7b snr7c mode'];
sv_dat=[var_svd,' var_svd'];
sv=['save ',data_file,sv_dat];eval(sv)
disp('The error in dB for all the modes:')
disp(mode)
disp('for full time, first half, and second half are:')
disp([snr6a snr6b snr6c])
disp('The error in dB for individual modes:')
disp('Mode: full time, first half, and second half are:')
disp([mode snr7a snr7b snr7c])
ss=['load ',sys];
eval(ss)
[n_act,d_act]=ss2tf(A,B,C,D,1);
disp('The numerator coefficients of the actual system are:')

```

```

vv=length(n_act)-1:-1:0;
disp([vv(:),n_act(:)])
[n_mod,d_mod]=ss2tf(Am,Bm,Cm,Dm,1);
disp('The numerator coefficients of the model are:')
vv=length(n_mod)-1:-1:0;
disp([vv(:),n_mod(:)])
disp('The denominator coefficients of the actual system are:')
vv=length(d_act)-1:-1:0;
disp([vv(:),d_act(:)])
disp('The denominator coefficients of the model are:')
vv=length(d_mod)-1:-1:0;
disp([vv(:),d_mod(:)])
disp(' '),disp(' ')
end
%+++++
%
%           That is the end of id_plt.m
%+++++

```

diff_model.m

```

function [ysdm]=diff_model(data_file)
%+++++
% FILE:
%   diff_model.m   - This function evaluates the discrete
%                   difference equations for the following system:
%
%                    $x((k+1)T) = F*x(kT) + G*u(kT)$ 
%
%                    $y(kT) = C*x(kT) + D*u(kT)$ 
%
% INPUT:
%   data_file      - Data file name, if left empty user will be
%                   queried for a file name.
%
% OUTPUT:
%   None.
%
% GLOBAL:
%   None.
%
% CALLS:
%   None.

```

```

%
% DATA FILES WRITTEN:
%     None.
%
% DATA FILES READ:
%     {'data_file'}.mat
%
% AUTHOR:
%     Steve Hietpas October, 1992
%
% REFERENCES:
%     None.
%
% SYNTAX:
%     [ysdm]=diff_model('data_file');
%     or
%     [ysdm]=diff_model;
%+++++
disp('Running the discrete difference equation solver diff_model.m')
default_data='ex_uyt';
if nargin~=1
    what
    strg=['Enter a data file name [',default_data,'] >> '];
    data_file=input(strg,'s');
    if isempty(data_file)
        data_file=default_data;
    end
end
ld=['load ',data_file];eval(ld);
model_file=[data_file,'_mod'];
ld=['load ',model_file];eval(ld);
%pr3_file=[data_file,'_pr3'];
%ld=['load ',pr3_file];eval(ld);
[Fi,Fj]=size(Fm);
xk=zeros(1,Fj);xk=xk(:);
t=0;tt=length(tsd);
T=tsd(2)-tsd(1);
ysdm=[];
for k=1:tt
    uk = usd(k);
    yk=Cm*xk + Dm*uk;
    ysdm=[ysdm; yk'];
    xkk=Fm*xk+Gm*uk;

```

```

    xk=xkk;
end
if exist('EE')
    if ~isempty(EE)
        len_P=length(P);
        Z=exp(P*T);
        thet2=zeros(length(tsd),len_P);
        thet2(1,:)=ones(1,len_P);
        for p=2:length(tsd)
            z=Z.^p;z=z(:);z=z.';
            thet2(p,:)=z;
        end
        y_ini_m=thet2*EE(:);
        ysdm=ysdm+real(y_ini_m);
    end
end
return
%+++++
%               That is the end of diff_model.m
%+++++

```

diff_modes.m

```

function []=diff_modes(data_file,system,u_file,plt,md)
%+++++
% FILE:
%   diff_model.m   - This function evaluates the discrete
%                   difference equations for the following
%                   system:
%
%                   
$$x((k+1)T) = F*x(kT) + G*u(kT)$$

%
%                   
$$y(kT) = C*x(kT) + D*u(kT)$$

%
% INPUT:
%   data_file      - Data file name, if left empty user will be
%                   queried for a file name.
%
%   system         - Data file that contains original system
%   u_file         - Input probing file used.
%   plt            - if plt == 1 then do plotting, else not.
%   md             - a vector of modes that can be applied to

```

```

%                               to the modeled system.
%
% OUTPUT:
%     None.
%
% GLOBAL:
%     None.
%
% CALLS:
%     None.
%
% DATA FILES WRITTEN:
%     {uuyytd.mat} - see saving section.
%
% DATA FILES READ:
%     {'data_file'}.mat
%
% AUTHOR:
%     Steve Hietpas October, 1992
%
% REFERENCES:
%     None.
%
% SYNTAX:
%     diff_modes(data_file,system,u_file,plt,md);
%+-----+
disp('Running the discrete difference equation solver diff_modes.m')
model_file=[data_file,'_mod'];
ld=['load ',data_file];eval(ld)
ld=['load ',model_file];eval(ld)
ld=['load ',system];eval(ld);
if nargin<=4
    disp('    The continuous time ')
    disp('eigenvalues of the model are are:')
    disp('-----')
        disp(eigAm)
%
%generate input
strg='Do you want to apply all complex eigenvalues? ';
strg=[strg,'[y,Y,<ret>=yes] >> '];
ans=input(strg,'s');
if ~(isempty(ans) | ans=='y' | ans=='Y')
strg='Select a vector of modes to apply, in rad/sec. [...] >> ';

```

```

md=input(strg)
ma=input('Select a vector of amplitudes to apply [....] >> ');
else
md=[];
mdk=imag(eigA);
for jj=1:length(eigA)
    if mdk(jj)>eps
        ii=find(md==mdk(jj));
        if isempty(ii)
            md=[md;mdk(jj)];
        end
    end
end
disp(' ')
disp('The mode(s) to apply are:')
disp(md)
ma=input('Select a vector of amplitude(s) to apply [....] >> ');
end
else
disp(' ')
disp('The modes to apply are:')
disp(md)
ma=input('Select a vector amplitude(s) to apply [....] >> ');
end
disp(' '),
disp('***** The final selection is *****'),disp(' ')
disp('The mode(s) to apply are:')
disp(md)
disp('The amplitude(s) are:')
disp(ma)
tt=2*length(tsd);
ft=tt*DT;
fprintf('\n The total time of simulation will be %g \n',ft)
dt=input('How long to apply the signal (seconds) >> ');
notdone=1;
while notdone
    t=0;tsde=[];usde=[];
    for k=1:tt
        u=feval(u_file,t,md,ma,dt);
        usde=[usde;u];tsde=[tsde;t];
        t=t+DT;
    end
    if plt

```

```

fftst(usde,tsde,plt,'Y',data_file);
ans=input('Is this acceptable? [y,Y,<ret>=yes] >> ','s');
if (isempty(ans) | ans=='y' | ans=='Y')
    notdone=0;
else
    disp('The previous values of modes are:')
    disp(md)
    disp('The previous value of mode amplitudes are:')
    disp(ma)
    md=input('What value for frequencies? ');
    ma=input('What value for amplitudes? ');
end
end
end
%Now apply the input driving signal
subplot(111)
[Fmj,Fmj]=size(Fm);
[Fj,Fj]=size(F);
t=0;
y1=[];y2=[];
xk1=zeros(1,Fmj);xk1=xk1(:);
xk2=zeros(1,Fj);xk2=xk2(:);
for k=1:tt
    uk = usde(k);
    yk1=Cm*xk1 + Dm*uk;
    yk2=C*xk2 + D1*uk;
    y1=[y1; yk1'];y2=[y2; yk2'];
    xkk1=Fm*xk1+Gm*uk;
    xkk2=F*xk2+G1*uk;
    xk1=xkk1;
    xk2=xkk2;
end
y1=real(y1);
y2=real(y2);
save uyyttd usde tsde y1 y2
return
%+++++
%
%           That is the end of diff_modes.m
%+++++

```

diff_nn.m

```

function ysnn=diff_nn(system,u_in,FT)
%+++++
% FILE:
%     diff_nn.m   - This function evaluates the discrete
%                  difference equations for the following
%                  system:
%
%                   $x((k+1)T) = F*x(kT) + G*u(kT)$ 
%
%                   $y(kT) = C*x(kT) + D*u(kT)$ 
%
% INPUT:
%     system      - Data file that contains original system
%     u_in        - Input probing file used.
%     FT          - Final Time.
% OUTPUT:
%     ysnn        - The output of the system with out any
%                  noise added.
%
% GLOBAL:
%     None.
%
% CALLS:
%     None.
%
% DATA FILES WRITTEN:
%     {diff_nn.mat} - see saving section.
%
% DATA FILES READ:
%     {'system'}.mat
%     {'u_in'}.mat
%
% AUTHOR:
%     Steve Hietpas October, 1992
%
% REFERENCES:
%     None.
%
% SYNTAX:
%     diff_modes(data_file,system,u_file,plt,md);

```

```

%+++++
disp(' '),disp(' ')
disp('Running the function diff_nn.m')
ld=['load ',system];eval(ld);
ld=['load ',u_in];eval(ld);
[Fi,Fj]=size(F);
xk=zeros(1,Fj);xk=xk(:);
xk0=xk;
%Now compute y with no extra noise at second input, call it ysnn
%(no-noise)
ysnn=[];
D1=D(:,1);G1=G(:,1);
for k=1:length(tsd)
    uk=usd(k);
    yk=C*xk+D1*uk;
    ysnn=[ysnn;yk];
    xkk=F*xk+G1*uk;
    xk=xkk;
end
var_svd_nn=' ysnn xk0 D1 G1';
sv_dat=[var_svd_nn,' var_svd_nn'];
sv=['save diff_nn',sv_dat];eval(sv)
return
%+++++
%               That is the end of diff_nn.m
%+++++

```

diff_noise.m

```

function [uout,tout]=diff_noise(system,u_in,diff_in,FT,y_ini)
%+++++
% FILE:
%       diff_noise.m - This function evaluates the discrete
%                       difference equations for the following
%                       system:
%
%                       
$$x((k+1)T) = F*x(kT) + G*u(kT)$$

%
%                       
$$y(kT) = C*x(kT) + D*u(kT)$$

%
% INPUT:
%       system      - Data file that contains original system.

```

```

%          u_in      - Input probing file used.
%          diff_in   - The difference equation solution without
%                      noise.
%          FT        - Final Time.
%          y_ini     - Initial value of yout.
% OUTPUT:
%          uout      - The input signal.
%          tout      - The time signal.
%
% GLOBAL:
%          None.
%
% CALLS:
%          None.
%
% DATA FILES WRITTEN:
%          {diff_noise.mat} - see saving section.
%
% DATA FILES READ:
%          {'system'}.mat
%          {'u_in'}.mat
%          {'diff_in'}.mat
%
% AUTHOR:
%          Steve Hietpas October, 1992
%
% REFERENCES:
%          None.
%
% SYNTAX:
%          diff_modes(data_file,system,u_file,plt,md);
%+-----+
% This function evaluates the discrete difference equations for
% the following system:
%
%      
$$x((k+1)T) = F*x(kT) + G*u(kT)$$

%
%      
$$y(kT) = C*x(kT) + D*u(kT)$$

%
% Written by Steve Hietpas: January, 1994
%
% disp(' '),disp(' ')

```

```

disp('Running the discrete difference equation solver diff_noise.m')
ld=['load ',system];eval(ld);
ld=['load ',u_in];eval(ld)
ld=['load ',diff_in];eval(ld)
[Fi,Fj]=size(F);
t=0;tt=fix(FT/DT);
%
[Di,Dj]=size(D);
[usdi,usdj]=size(usd);
sgnri=[];
notdone = 1;
while notdone
    strg='Use an existing data file for noise input ';
    strg=[strg,'[n,N,<ret>=no] >> '];
    ans=input(strg,'s');
    if ~(isempty(ans) | ans=='n' | ans=='N')
        what
        nn=input('Select a noise data file: >> ','s');
        if exist([nn,'.mat'])
            ld=['load ',nn];eval(ld) %should give back a noise vector nn.
        end
    else
        noise=input('Which noise type: [(normal=1)] or (uniform=2) >> ');
        if isempty(noise)
            noise=1;
        end
        nnmf_def=0.0001;
        strg='Enter the noise multiplication factor: ';
        strg=[strg,'[def=',num2str(nnmf_def),'] >> '];
        nnmf=input(strg);
        if isempty(nnmf)
            nnmf=nnmf_def;
        end
        if ~(noise==1)
            nn=rand(size(usd));
            nn=nnmf*nn; %as nnmf is larger the SNR goes down.
            nn=nn-0.5; %uniform distribution, mean=0
        else %normal distribution, 0 mean, unit variance
            nn=randn(size(usd));
            nn=nnmf*nn; %as nnmf is larger the SNR goes down.
            nnm=mean(nn); %force mean to be as close to 0 as possible
            nn=nn-nnm;
        end
    end
end

```

```

end
sgnri=20*log10(norm(usd)/norm(nn));
SNR=['The SNR of probing to noise input is ',num2str(sgnri)];
disp(SNR)
usd2=[usd nn];
ysd2=[];
xk=xk0;
for k=1:tt
    uk=usd2(k,:);
    if ~isempty(y_ini)
        yk=C*xk + D*uk + y_ini(k);
    else
        yk=C*xk + D*uk;
    end
    ysd2=[ysd2; yk'];
    xkk=F*xk+G*uk;
    xk=xkk;
end
sgnr2=20*log10(norm(ysnn)/norm(ysnn-ysd2));
fprintf('The SNR of ysnr to ysnr-ysd2 is %g dB \n',sgnr2)
ans=input('Is this acceptable? [y,Y,<ret>=yes] >> ','s');
if (isempty(ans) | ans=='y' | ans=='Y')
    notdone=0;
    strg='Save the noise vector, nn, to a file? [n,N,<ret>=no] >> ';
    ans=input(strg,'s');
    if ~(isempty(ans) | ans=='n' | ans=='N')
        what
        fn=input('Enter a file name to save nn to: ','s');
        sv=['save ',fn,' nn sgnr2'];eval(sv)
    end
end
end
end
%
var_svd_noise=' usd2 ysd2 sgnri sgnr2';
sv_dat=[var_svd_noise,' var_svd_noise'];
sv=['save diff_noise',sv_dat];eval(sv)
return
%+++++
%
%               That is the end of diff_noise.m
%+++++

```

pr1.m

```

function []=pr1(data_file)
%+++++
% FILE:
%      pr1.m      - Forward prediction using backward difference
%                  equations.
%
% INPUT:
%      data_file  - Data file name, if left empty user will be
%                  queried for a file name.
%                  See 'runcomp.m' for all variables saved.
%                  If you are not running pr1.m after having
%                  run 'runcomp.m', and your data is generated
%                  elsewhere than you will need to make
%                  available the following data variables:
%                  DT      - delta T
%                  FT      - final time
%                  cc      - magnitude of input modes} see
%                  dd      - delay intervals of input} u_prony.m
%                  ss      - modes of input}
%                  mpi     - number of modes/interval}
%                  ee      - magnitude of initial condition modes
%                  tsd     - time data vector
%                  ysd2    - output data vector
%                  usd     - input data vector from u_prony.m
%                          possibly.
%                  var_svd - variables saved, i.e.,
%                  var_svd='DT FT cc dd ss ee mpi tsd ysd usd var_svd'
%
% OUTPUT:
%      None.
%
% GLOBAL:
%      None.
%
% CALLS:
%      pr1b      - A subprogram of pr1 that allows for
%                  preliminary reduction of the modes found in
%                  this subroutine.
%
% DATA FILES WRITTEN:
%      pr1.mat   - See saving section.

```

```

%
% DATA FILES READ:
%     None.
%
% AUTHOR:
%     Steve Hietpas August 1993
%
% REFERENCES:
%     None.
%
% SYNTAX:
%     pr1('data_file');
%     or
%     pr1
%+-----+
%+-----+
%      $y(n) = \sum_{k=1:M} A_k z^n$  where  $z = \exp(s)$ , for  $n=0,1,\dots,N$ 
%     and where  $s = -a_k + j*2\pi*f_k$ , for  $k=1,2,\dots,M$ ,  $a_k$  are
%     positive.
%     Thus  $a_k$  are the damping coefficients,  $f_k$  are the pole
%     frequencies
%     and  $A_k$  are the amplitudes for  $M$  damped exponential/sinusoids.
%
%     The input signal to the system,  $u$ , may or may not be of this
%     form, but its signal decomposition can be expressed in the same
%     way.
%     There are basically four methods for modal decomposition:
%     I. Separate (each interval) modal decomposition:
%         For each interval of sufficient length,
%         (to be defined below)
%         solve for a set of modes that includes both system and
%         input modes.
%         Remove the input modes for each set, find an
%         intersection of the output modes.
%         This may be fairly difficult to do, since for numerical and
%         practical reasons the intersection may be the null set.
%         Therefore, each mode will have an assigned uncertainty
%         profile; if a modes from each set fall within these

```

% uncertainty
 % regions a system mode is thus determined and a corresponding
 % uncertainty profile is defined. This approach is difficult,
 % however, an uncertainty model is automatically built into
 % the process and can be quite useful for robust control
 % design.
 % For sufficient length consider the following example.
 % Consider any interval (denoted by i) with an input drive
 % signal
 % with k_i modes associated with it.
 % Assume L modes for the system. An interval of sufficient
 % length needs to have more than $2xL + k_i$ samples of data.

II. All modes per interval decomposition:
 % Consider all intervals of sufficient length
 % (to be defined below) and form a large (overdetermined) set
 % of equations and solve for the coefficients of systems
 % characteristic equation.
 % Sufficient length is determined by the sum of all number of
 % modes used for each interval and the number of modes one
 % chooses to estimate for the system. Say there are three
 % intervals (denoted by i) of differing input drive signals
 % and each interval has k_i modes associated with it.
 % Assume L modes for the system. An interval of
 % sufficient length must necessarily have more than $L +$
 % $\sum(k_j)$ samples, where the modes k_j are the union of
 % modes k_i .
 % Additionally, there needs to be enough intervals and
 % data to form more than $L + \sum(k_j)$ equations.

III. System modes over all intervals where input is off:
 % Consider all intervals of sufficient length
 % (to be defined below)
 % for which the input probing signal is off and form a large
 % (overdetermined) set of equations and solve for the
 % coefficients of the system's characteristic equation.
 % Sufficient length is determined by the number of modes
 % one chooses
 % to estimate for the system. Assume L modes for the system
 % then
 % there needs to be enough intervals and data to form more
 % than L equations.

IV. System mode decomposition:
 % Consider the known characteristic equation of the input and
 % form auxiliary equations involving equation and the output

```

%      data.
%      These auxilliary equations can than be used for each
%      interval to form a linear set of equations to solve for the
%      coefficients of the characteristic equation for the system
%      directly.
%      An interval of data can be used to form equations if it is
%      of sufficient length (to be defined below).
%      The set of equations can be used all together to determine
%      one set of system modes similar to step II or an approach
%      similar to I above can be employed to determine separate
%      set of modes from which an intersection can be determined.
%      The method of all interval equations combined as in II is
%      preferred from the standpoint of ease in numerical example.
%      Sufficient length in this case, using the example from II
%      above is an interval with at least L+1 samples of data.
%      Additionally, there must exist enough data such that more
%      than L equations can be formed.
%      If noise is an issue, this may not be a good method.
%
%      Method II and III are the approaches implemented in this
%      routine.
%
%+++++
disp('Now running pr1.m')
default_data='ex_uyt';
if nargin~=1
    what
    strg=['Enter a data file name [',default_data,'] >> '];
    data_file=input(strg,'s');
    if isempty(data_file)
        data_file=default_data;
    end
end
ld=['load ',data_file];eval(ld)
T=DT;yy=ysd2;
N=length(yy);
%+++++
%
m_title='Choose method of decomposition';
I='1 >> Not available at this time, sorry';
II='2 >> All modes over each interval (ok approach)';
III='3 >> System modes over all intervals s.t. input is off (good)';
IV='4 >> System modes only (best in no noise case)';

```

```

meth=menu(m_title,I,II,III,IV);
%disp(m_title)
%disp(I),disp(II),disp(III),disp(IV)
%meth=input('Enter your choice [<ret>=3] >. ');
%if isempty(meth)
% meth=3;
%end
L=input('What order system do you predict? ');
disp(' ')
disp(' ----- CRUNCHING ----- ')
disp(' ')
%+++++
q=length(dd);
Phi=[];v=[];
%Start of loop for constructing characteristic eq. of input signal
% for each interval.
M=dd/T;
if meth==2
    %Determine how many distinct input modes for all intervals
    %combined.
    d_ss=ss(1);
    for kk=2:length(ss)
        flag=1;
        for kkk=1:length(d_ss)
            if ss(kk)==d_ss(kkk)
                flag=0;
            end
        end
        if flag==1
            d_ss=[d_ss;ss(kk)];
        end
    end
    Ld_ss=length(d_ss);
    L=L+Ld_ss;
end
if meth~=1
    for k=1:q
        m_k=mpi(k); %modes per interval.
        if k==1
            M_l=0; %M_{k-1}
        else
            M_l=M(k-1); %M_{k-1}
        end
    end
end

```

```

if meth==2
    %Decide if this interval will be of sufficient length.
    if (M(k)-1)>(M_l+L)
        flag2=1;
        p_k=yy(M_l+1:M(k));
    else
        flag2=0;
    end
elseif meth==3
    %Determine if input is off.
    if mpi(k)==0
        %Decide if this interval will be of sufficient length.
        if (M(k)-1)>(M_l+L)
            flag2=1;
            p_k=yy(M_l+1:M(k));
        else
            flag2=0;
        end
    else
        flag2=2; %Input is on, can't use data during this interval.
    end
else %meth=4
    %Decide if this interval will be of sufficient length.
    if (M(k)-1)>(M_l+m_k+L)
        flag2=1;
        m_tot=sum(mpi(1:k))-m_k;
        if m_k~=0
            %Construct input characteristic poly. a_k for interval k.
            a_k=1;
            for j=1:m_k
                a_tmp=[1 -exp(ss(m_tot+j)*T)];
                a_k=conv(a_k,a_tmp);
            end
            a_k=real(a_k);
            %Find vector p_k by operating a_k on measured output y.
            p1=M_l+m_k+1;p2=M(k);
            col=yy(p1:p2);
            row=yy(M_l+1:p1);row=flipud(row);
            Y_k=toeplitz(col,row);
            a_k=a_k(:);
            p_k=Y_k*a_k;
        else
            p_k=yy(M_l+1:M(k));
        end
    else
        flag2=0;
    end
end

```

```

        end
    else
        flag2=0;
    end
end
%Construct Phi_k which is used to solve for the characteristic
%polynomial coefficients of the system.
if flag2==1
    Lp_k=length(p_k);
    col=p_k(L:Lp_k-1);
    row=p_k(1:L);row=flipud(row);
    Phi_k=toeplitz(col,row);
    v_k=-p_k(L+1:Lp_k);
    Phi=[Phi;Phi_k];
    v=[v;v_k];
elseif flag2==0
    intv1=num2str(M_l*T);intv2=num2str(M(k)*T);
    s=['Output data from time ',intv1,' sec. to ',intv2];
    s=[s,' sec. could not be used; interval was too short.'];
    disp(s)
    disp('                Hit <rtm> to continue')
    pause
else %flag2=2
    intv1=num2str(M_l*T);intv2=num2str(M(k)*T);
    s=['Output data from time ',intv1,' sec. to ',intv2];
    s=[s,' sec. could not be used; probing input is on.'];
    disp(s)
    disp('                Hit <rtm> to continue')
    pause
end
end
if M(q)<N
    if (N-1)>(M(q)+L)
        col=yy(M(q)+L-1:N-1);
        row=yy(M(q):M(q)+L-1);row=flipud(row);
        Phi_k=toeplitz(col,row);
        v_k=-yy(M(q)+L:N);
        Phi=[Phi;Phi_k];
        v=[v;v_k];
        bb=pinv(Phi)*v;
        B=[1 rot90(bb)];
        Z_sys=conj(roots(B));
        %Remove all modes in 2nd and 3rd quadrants of complex Z-plane.
    end
end

```

,

pr1b.m

```

function []=pr1b(data_file)
%+++++
% FILE:
%     pr1.m      - This code follows pr1.m.  All continuous time
%                  modes are displayed.  Removal of modes is made
%                  possible by this routine.
%
% INPUT:
%     data_file  - File name that contains all data necessary for
%                  analysis.  If not provided, user will be
%                  queried.
%
% OUTPUT:
%     None.
%
% GLOBAL:
%     None.
%
% CALLS:
%     pr2        - This routine fits residues to the selected
%                  values of system modes.
%
% DATA FILES WRITTEN:
%     data_file  - See saving section.
%
% DATA FILES READ:
%     data_file  - See pr1.m.
%
% AUTHOR:
%     Steve Hietpas August 1993
%
% REFERENCES:
%     None.
%
% SYNTAX:
%     pr1b('data_file_name');
%     qr
%     pr1b;
%+++++
%+++++

```

```

% Get data.
%+++++
disp('Now running pr1b.m')
default_data='ex_uyt';
if nargin~=1
    what
    strg=['Enter a data file name [',default_data,'] >> '];
    data_file=input(strg,'s');
    if isempty(data_file)
        data_file=default_data;
    end
end
ld=['load ',data_file];eval(ld)
%
%Consider the output modes, Z_sys.
%Zero out real and/or imaginary parts of modes if they are of
%insignificant size.
L=length(Z_sys);
for k=1:L
    Sre_sys(k)=log(abs(Z_sys(k)))/T; %Convert to continuous time
                                   %eigenvalues
    Sim_sys(k)=angle(Z_sys(k))/T; %Convert to continuous time
                                   %eigenvalues

    if abs(Sre_sys(k))<1e-5
        Sre_sys(k)=0.0;
    end
    if abs(Sim_sys(k))<1e-5
        Sim_sys(k)=0.0;
    end
end
Sre_sys=Sre_sys(:);Sim_sys=Sim_sys(:);
S_sys=Sre_sys+i*Sim_sys;
%+++++
% Make sure list of complex-valued modes have a conjugate present.
%+++++
S_sys2=[];
k=1;
while k<=L
    if Sim_sys(k)~=0
        if k+1<=L
            check=norm(S_sys(k)-conj(S_sys(k+1)));
            if check<=eps % Keep S_sys(k) and S_sys(k+1).
                S_sys2=[S_sys2;S_sys(k);S_sys(k+1)];
            end
        end
    end
    k=k+1;
end

```

```

        k=k+2;
    else % Don't keep S_sys(k) and S_sys(k+1).
        k=k+1;
    end
    else % Don't keep S_sys(k) and S_sys(k+1).
        k=k+1;
    end
    else % Keep S_sys(k).
        S_sys2=[S_sys2;S_sys(k)];
        k=k+1;
    end
end
S_sys=S_sys2;
Sim_sys=imag(S_sys);Sre_sys=real(S_sys);
%+++++
% Display and remove section.
%+++++
Nq=2/(T);
Nqr=2*pi*Nq;
notdone = 1;
while notdone
    xdisp_mode(S_sys,'system');
    ans1=input('Do you wish to remove modes? [n,N,<rtn>=no] >> ','s');
    if ~(isempty(ans1) | ans1=='n' | ans1=='N')
        strg='The Nyquist frequency in Hz is: %g (in rad/sec is: %g)';
        sprintf(strg,Nq,Nqr)
        S_sys2=rm_mode(S_sys);
        xdisp_mode(S_sys2,'system');
        strg='Are you satisfied with this set? [y,Y,<rtn>=yes] >> ';
        ans2=input(strg,'s');
        if (isempty(ans2) | ans2=='y' | ans2=='Y')
            S_sys=S_sys2;
            notdone=0;
        else
            clear S_sys2 ans1
        end
    else
        notdone=0;
    end
end
clear S_sys2
%+++++
% Save section.

```

```

%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
new_data=' S_sys';
if isempty(findstr(new_data,var_svd))
    var_svd=[var_svd,new_data];
end
sv_dat=[var_svd,' var_svd'];
sv=['save ',data_file,sv_dat];eval(sv);
disp(['The following variables were saved in ',data_file,'.mat']);
disp(' ')
disp(sv_dat)
disp('----- hit <rtn> to continue -----')
pause
disp('This is the end of pr1b.m')
ss='pr2.m will use results of pr1b.m and in a least-squares sense'
disp(ss)
disp('fit residues to these modes.')
ans=input('Do you want to run pr2.m? [y,Y,<rtn>=yes] >> ','s');
if (isempty(ans) | ans=='y' | ans=='Y')
    pr=['pr2('','',data_file,'','',')'];eval(pr)
    return
end
return
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
%                               That's the end of pr1b.m
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

```

pr2.m

```

function []=pr2(data_file)
%++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
% FILE:
%      pr2.m      - This is the residue fitting stage of digital
%                  Prony. This code follows pr1b. All modes of
%                  the system are displayed for the continuous
%                  time model.
%                  This routine will take each individual
%                  system mode and form a discretized system for
%                  it.
%                  Each system is then driven by the original
%                  input producing a new set of data from which
%                  associated residues can be estimated.
%
% INPUT:

```

```

%      data_file - File name that contains all data necessary for
%                  analysis.
%                  If not provided, user will be queried.
%
% OUTPUT:
%      None.
%
% GLOBAL:
%      None.
%
% CALLS:
%      pr2b      - A subprogram of pr2 that allows for ordering
%                  of mode/residue data.
%
% DATA FILES WRITTEN:
%      data_file - See saving section.
%
% DATA FILES READ:
%      data_file - See pr1b.m.
%
% AUTHOR:
%      Steve Hietpas August 1993
%
% REFERENCES:
%      None.
%
% SYNTAX:
%      pr2({data_file});
%      or
%      pr2;
%
% ++++++
%
% ++++++
% The following set of equations are formed:
%
% 
$$x_k((p+1)T) = F_k(T) * x_k(pT) + H_k(T) * u(pT); \quad k=0,1,\dots,L \text{ and } p=0,1,\dots,N.$$

% 
$$y_k(pT) = R_k * x_k(pT);$$

%
% where  $H_k(pT) = \int\limits_0^T \exp(S_{\text{sys}}(k)T)$  and
%  $F_k(T) = \exp(S_{\text{sys}}(k)T)$ ,
% Then  $z_k = g_k(T)$  and the following result:
%

```

```
% [Theta1 Theta2]*[R(:) E(:)]' = yy(:) + error
```

```
%
```

```
% where
```

```
%
```

```
%
```

```
%      -
%      | u(0)  x1(0)      ... xL(0) |
% Theta1 = | u(1)  x1(1)      ... xL(1) |
%          | :      :      :      |
%          | u(N)  x1(N)      ... xL(N) |
%      -
```

```
%
```

```
%
```

```
%      -
%      | 1      1      ... 1      |
%      | z1     z2     ... zL     |
% Theta2 = | z1^2  z2^2  ... zL^2  |
%          | :      :      :      |
%          | z1^N  z2^N  ... zL^N  |
%      -
```

```
%
```

```
%
```

```
%      - -
%      | R0 |
% R(:) = | R1 |
%      | :  |
%      | RL |
%      - -
```

```
%
```

```
%
```

```
%      - -
%      | E1 |
% E(:) = | E2 |
%      | :  |
%      | EL |
%      - -
```

```
%
```

```
%
```

```
%
```

```
%      - -
%      | yy(0) |
% yy(:) = | yy(1) |
%      | :      |
%      | yy(N) |
%      - -
```

```
%
```

```
%
```

```
%+++++
```

```
%+++++
```

```
% Get data.
```

```
%+++++
```

```

disp('Now running pr2.m')
default_data='ex_uyt';
if nargin~=1
    what
    strg=['Enter a data file name [',default_data,'] >> '];
    data_file=input(strg,'s');
    if isempty(data_file)
        data_file=default_data;
    end
end
ld=['load ',data_file];eval(ld)
T=DT;
yy=ysd2;N=length(yy);
uu=usd;
L=length(S_sys);

%+++++
%
% This section evaluates the discrete difference equations for
% n 1st order continuous systems, assuming the input u(t) is
% zero-order sample-and-held driving function, uu(t):
%
% 
$$d/dt(x_k(t)) = A(k)*x_k(t) + B(k)*uu(t)$$

%
% Discretized for T
%
% 
$$x_k((p+1)T) = F(T)*x_k(pT) + H(T)*uu(pT)$$

%
% 
$$y_k(pT) = C_k*x_k(pT)$$

%
%+++++
theta1=[];
FF=[];HH=[];
for k=1:L
    f=exp(S_sys(k)*T);
    if f==1
        h=T;
    else
        h=(f-1)/S_sys(k);
    end
    FF=[FF;f];HH=[HH;h];
end
xk=zeros(size(FF));xk=xk(:);

```

```

%
%For these 1st order systems, Ck = 1 for k=1,2,...,L
%(we will solve for R last)
%Thus
%
%          T
%   C = [1 1 ... 1 ]
%
%and yk=C.*xk=xk
%
for p=1:N
    theta1=[theta1;xk.'];
    uk=uu(p);
    xkk=FF.*xk+HH*uk;
    xk=xkk;
end
%+++++
%If the initial conditions were not assumed to be zero, then set up
%theta2 for initial condition residues to be found.
strg='Was system at zero-value initial conditions? ';
strg=[strg,'[y,Y,<ret>=yes] >> '];
ini_cond_0=input(strg,'s');
if ~(isempty(ini_cond_0) | ini_cond_0=='y' | ini_cond_0=='Y')
    ini_cond_0='N';
    % construct theta2.
    Z=exp(S_sys*T);
    theta2=zeros(N,L);
    theta2(1,:)=ones(1,L);
    for p=2:N
        z=Z.^p;z=z(:);z=z.';
        theta2(p,:)=z;
    end
else
    theta2=[];
end
%+++++
%Now set up the matrix to solve for Rk and Ek where k=1,2,...,L.
%
strg='Do you have feedthru in your system? [n,N,<ret>=no] >> ';
feedthru=input(strg,'s');
if ~(isempty(feedthru) | feedthru=='n' | feedthru=='N')
    feedthru='Y';
    %Then feedthru is present.
    theta1=[uu theta1];

```

```

if ini_cond_0=='N'
    fit=1;%Data is being fit to system with feedthru and non-zero
        %value initial conditions.
    theta = [theta1 theta2];
    RE = pinv(theta)*yy;
    R = RE(1:L+1);
    E = RE(L+2:2*L+1);
else
    fit=2;
    %Data is being fit to system with feedthru and zero value
    %initial conditions.
    theta = theta1;
    R = pinv(theta)*yy;
    E = [];
end
else
    if ini_cond_0=='N'
        fit=3;
        %Data is being fit to system with no feedthru and non-zero
        %value initial conditions.
        theta = [theta1 theta2];
        RE = pinv(theta)*yy;
        R = RE(1:L);
        E = RE(L+1:2*L);
    else
        fit=4;
        %Data is being fit to system with no feedthru and zero value
        %initial conditions.
        theta = theta1;
        R = pinv(theta)*yy;
        E = [];
    end
end
end
%+++++
tol=eps*100;
%zero out very small values of real and imaginary parts of R
if ((fit==1) | (fit==2))
    Lt=L+1;
else
    Lt=L;
end
for k=1:Lt
    Rre=real(R(k));Rim=imag(R(k));

```

```

if abs(Rre)<tol
    Rre=0;
end
if abs(Rim)<tol
    Rim=0;
end
if ((fit==1) | (fit==2))
    if k > 1
        if imag(S_sys(k-1))==0
            Rim=0;
        end
    else
        Rim=0;
    end
else
    if imag(S_sys(k))==0
        Rim=0;
    end
end
R(k)=Rre+i*Rim;
end
if ((fit==1) | (fit==3))
    %zero out very small values of real and imaginary parts of E
    for k=1:L
        Ere=real(E(k));Eim=imag(E(k));
        if abs(Ere)<tol
            Ere=0;
        end
        if abs(Eim)<tol
            Eim=0;
        end
        if imag(S_sys(k))==0
            Eim=0;
        end
        E(k)=Ere+i*Eim ;
    end
end
end
%+++++
% Save section.
%+++++
new_data=' theta1 theta2 R E fit';
if isempty(findstr(new_data,var_svd))
    var_svd=[var_svd,new_data];

```

```

end
sv_dat=[var_svd,' var_svd'];
sv=['save ',data_file,sv_dat];eval(sv);
disp(['The following variables were saved in ',data_file,'.mat']);
disp(' ')
disp(sv_dat)
disp('----- hit <rtm> to continue -----')
pause
disp('This is the end of pr2.m')
disp('pr2b.m will apply sorting algorithms to the data from pr2.m')
ans=input('Do you want to run pr2b.m? [y,Y,<ret>=yes] >> ','s');
if (isempty(ans) | ans=='y' | ans=='Y')
    pr=['pr2b('','',data_file,'','','')'];eval(pr)
    return
end
return
%+++++
%
%               That's the end of pr2.m
%+++++

```

pr2b.m

```

function []=pr2b(data_file)
%+++++
%  FILE:
%      pr2b.m      - This code follows pr2.  The function of
%                  this code is to sort the set of systems
%                  modes and residues according to some
%                  ordering criterion.
%                  Two methods of ordering are available,
%                  Sequential and by Residue.
%
%  INPUT:
%      data_file    - File name that contains all data necessary
%                  for analysis.  If not provided, user will
%                  be queried.
%
%  OUTPUT:
%      None.
%
%  GLOBAL:
%      None.
%

```

```

% CALLS:
%     pr3           - This routine will display and plot model
%                   verses original data subjected to the
%                   probing input uu.
%
% DATA FILES WRITTEN:
%     data_file     - See saving section.
%
% DATA FILES READ:
%     data_file     - See pr2.m.
%
% AUTHOR:
%     Steve Hietpas August 1993
%
% REFERENCES:
%     None.
%
% SYNTAX:
%     pr2b({data_file});
%     or
%     pr2b;
%+-----+

%+-----+
% Get data.
%+-----+
disp('Now running pr2b.m')
default_data='ex_uyt';
if nargin~=1
    what
    strg=['Enter a data file name [',default_data,'] >> '];
    data_file=input(strg,'s');
    if isempty(data_file)
        data_file=default_data;
    end
end
ld=['load ',data_file];eval(ld)
T=DT;
yy=ysd2;N=length(yy);
L=length(S_sys);
%
%     fit=1: %Data is fit to system with feedthru and non-zero value
%           %initial conditions.

```

```

%   fit=2: %Data is fit to system with feedthru and zero value
%           %initial conditions.
%   fit=3: %Data is fit to system with no feedthru and non-zero
%           %value initial conditions.
%   fit=4: %Data is fit to system with no feedthru and zero value
%           %initial conditions.
thetaR1=theta1; %theta1 has to do with system residues, R.
thetaE=theta2; %theta2 has to do with initial condition residues,
               %E.

R1=R;
if ((fit==1) | (fit==2)) %Feedthru term exists.
    R0=R1(1); R1=R1(2:L);
    thetaR0=thetaR1(:,1); thetaR1=thetaR1(:,2:L);
else
    R0=[]; thetaR0=[];
end
%+++++
%This section will compute yhat for each mode and determine a root
%mean square error for the difference between yhat and yy, the
%measured output data.
%From this data the modes will be ordered in such a way that those
%modes contributing the least amount of error for the given error
%criteria will be placed first in the ordering list.
ytemp=zeros(N,1);
R1len=length(R1);
thetar1=thetaR1;
thetae=thetaE;
r1=R1;
e=E;
thetaR1=[];
thetaE=[];
ord=[];
%+++++
disp('      This phase of ordering is SEQUENTIAL')
%+++++
k=0;
if ((fit==1) | (fit==2)) %Feedthru term exists.
    ytemp=real(thetaR0*R0);
end
while R1len>0
    yherr=zeros(R1len,1);
    k=1;
    while k<=R1len

```

```

if ((fit==1) | (fit==3)) %non-zero initial conditions exist.
    if imag(r1(k))==0.0
        yh=ytemp+real(thetar1(:,k)*r1(k)+thetae(:,k)*e(k));
        yherr(k)=20*log10(norm(yh-yy)/norm(yy));
        k=k+1;
    else
        yh=ytemp+real(thetar1(:,k:k+1)*r1(k:k+1)+thetae(:,k:k+1)*e(k:k+1));
        yherr(k)=20*log10(norm(yh-yy)/norm(yy));
        yherr(k+1)=yherr(k);
        k=k+2;
    end
else %Zero-value initial conditions.
    if imag(r1(k))==0.0
        yh=ytemp+real(thetar1(:,k)*r1(k));
        yherr(k)=20*log10(norm(yh-yy)/norm(yy));
        k=k+1;
    else
        yh=ytemp+real(thetar1(:,k:k+1)*r1(k:k+1));
        yherr(k)=20*log10(norm(yh-yy)/norm(yy));
        yherr(k+1)=yherr(k);
        k=k+2;
    end
end
end
[yher,index]=sort(yherr);
r1ord=[];
eord=[];
thetar1ord=[];
thetaeord=[];
modeord=[];
for kk=1:Rilen
    ii=index(kk);
    r1ord=[r1ord; r1(ii) yher(kk)];
    thetar1ord=[thetar1ord thetar1(:,ii)];
    if ((fit==1) | (fit==3))
        eord=[eord; e(ii)];
        thetaeord=[thetaeord thetae(:,ii)];
    end
    modeord=[modeord; S_sys(ii)];
end
clear yher index
if ((fit==1) | (fit==3)) %Non-zero initial conditions.
    if(imag(r1ord(1,1)))==0.0

```

```

        ytemp=ytemp+real(thetarlord(:,1)*rlord(1,1)+...
            thetaeord(:,1)*eord(1,1));
        r1=rlord(2:Rilen,1);
        e=eord(2:Rilen,1);
        S_sys=modeord(2:Rilen);
        ord=[ord; modeord(1) rlord(1,:) eord(1)];
        thetaR1=[thetaR1 thetarlord(:,1)];
        thetar1=thetarlord(:,2:Rilen);
        thetaE=[thetaE thetaeord(:,1)];
        thetae=thetaeord(:,2:Rilen);
        Rilen=Rilen-1;
    else
        ytemp=ytemp+real(thetarlord(:,1:2)*rlord(1:2,1) + ...
            thetaeord(:,1:2)*eord(1:2,1));
        r1=rlord(3:Rilen,1);
        e=eord(3:Rilen,1);
        S_sys=modeord(3:Rilen);
        ord=[ord; modeord(1:2) rlord(1:2,:) eord(1:2,:)];
        thetaR1=[thetaR1 thetarlord(:,1:2)];
        thetar1=thetarlord(:,3:Rilen);
        thetaE=[thetaE thetaeord(:,1:2)];
        thetae=thetaeord(:,3:Rilen);
        Rilen=Rilen-2;
    end
else %Zero-value initial conditions.
    if(imag(rlord(1,1)))==0.0
        ytemp=ytemp+real(thetarlord(:,1)*rlord(1,1));
        r1=rlord(2:Rilen,1);
        S_sys=modeord(2:Rilen);
        ord=[ord; modeord(1) rlord(1,:)];
        thetaR1=[thetaR1 thetarlord(:,1)];
        thetar1=thetarlord(:,2:Rilen);
        Rilen=Rilen-1;
    else
        ytemp=ytemp+real(thetarlord(:,1:2)*rlord(1:2,1));
        r1=rlord(3:Rilen,1);
        S_sys=modeord(3:Rilen);
        ord=[ord; modeord(1:2) rlord(1:2,:)];
        thetaR1=[thetaR1 thetarlord(:,1:2)];
        thetar1=thetarlord(:,3:Rilen);
        Rilen=Rilen-2;
    end
end
end

```

```

clear r1ord eord thetar1ord thetaeord modeord
end
S_sys=ord(:,1);
R1=ord(:,2);
ord_err=ord(:,3);
if ((fit==1) | (fit==3))
    E=ord(:,4);
end
%+++++
% Save section 1.
%+++++
S_sys_seq=S_sys;
R0_seq=R0;
thetaR0_seq=thetaR0;
R1_seq=R1;
E_seq=E;
thetaR1_seq=thetaR1;
thetaE_seq=thetaE;
ord_err_seq=ord_err;
data_file2=[data_file,'_ord'];
new_data=' S_sys_seq R0_seq thetaR0_seq R1_seq E_seq thetaR1_seq
new_data=[new_data,' thetaE_seq ord_err_seq'];
if exist('var_svd2')
    if isempty(findstr(new_data,var_svd2))
        var_svd2=[var_svd2,new_data];
    end
else
    var_svd2=new_data;
end
sv_dat=[var_svd2,' var_svd2'];
sv=['save ',data_file2,sv_dat];eval(sv);
disp(['The following variables were saved in ',data_file2,'.mat']);
disp(' ')
disp(sv_dat)
disp('----- hit <rtm> to continue -----')
pause
%This completes the first approach to ordering.
%+++++
% This starts the second approach to ordering which is done by
% residue size, the largest magnitudes to the smallest.
%+++++
ytemp=zeros(N,1);
ord_err=[];

```

```

R1len=length(R1);
r1=R1;
e=E;
thetar1=thetaR1;
thetae=thetaE;
%+++++
disp('      This phase of ordering is RESIDUE')
%+++++
% Sort the residues and save the ordering by index.
[rr1,index]=sort(r1);
r1=flipud(rr1);
index=flipud(index);
% Now use index to allign the modes, etc.
s_sys=S_sys(index);
thetar1=thetar1(:,index');
if ((fit==1) | (fit==3)) %Non-zero initial conditions.
    thetae=thetaE(:,index');
    e=E(index);
end
% Save these values in original variables.
R1=r1;
thetaR1=thetar1;
S_sys=s_sys;
if ((fit==1) | (fit==3)) %Non-zero initial conditions.
    thetaE=thetae;
    E=e;
end
Rl=R1len;
ytemp=zeros(N,1);
if (fit==1) | (fit==2) %Feedthru term exists.
    ytemp=real(thetaR0*R0);
end
rr1=r1;
re=e;
if ((fit==1) | (fit==3)) %Non-zero initial conditions.
    while Rl>0
        if(imag(rr1(1)))==0.0
            ytemp=ytemp+real(thetar1(:,1)*rr1(1) + thetae(:,1)*re(1));
            yherr=20*log10(norm(ytemp-yy)/norm(yy));
            ord_err=[ord_err; yherr];
            rr1=rr1(2:Rl);
            re=re(2:Rl);
            s_sys=s_sys(2:Rl);

```

```

    thetar1=thetar1(:,2:Rl);
    thetae=thetae(:,2:Rl);
    Rl=Rl-1;
else
    ytemp=ytemp+real(thetar1(:,1:2)*rr1(1:2)+...
        thetae(:,1:2)*re(1:2));
    yherr=20*log10(norm(ytemp-yy)/norm(yy));
    ord_err=[ord_err; yherr; yherr];
    rr1=rr1(3:Rl);
    re=re(3:Rl);
    s_sys=s_sys(3:Rl);
    thetar1=thetar1(:,3:Rl);
    thetae=thetae(:,3:Rl);
    Rl=Rl-2;
end
end
else %Zero-value initial conditions.
    while Rl>0
        if(imag(rr1(1)))==0.0
            ytemp=ytemp+real(thetar1(:,1)*rr1(1));
            yherr=20*log10(norm(ytemp-yy)/norm(yy));
            ord_err=[ord_err; yherr];
            rr1=rr1(2:Rl);
            s_sys=s_sys(2:Rl);
            thetar1=thetar1(:,2:Rl);
            Rl=Rl-1;
        else
            ytemp=ytemp+real(thetar1(:,1:2)*rr1(1:2));
            yherr=20*log10(norm(ytemp-yy)/norm(yy));
            ord_err=[ord_err; yherr; yherr];
            rr1=rr1(3:Rl);
            s_sys=s_sys(3:Rl);
            thetar1=thetar1(:,3:Rl);
            Rl=Rl-2;
        end
    end
end
end
if ((fit==1) | (fit==3))
    E=re;
end
%+++++
% Save section 2.
%+++++

```

```

S_sys_res=S_sys;
R0_res=R0;
thetaR0_res=thetaR0;
R1_res=R1;
E_res=E;
thetaR1_res=thetaR1;
thetaE_res=thetaE;
ord_err_res=ord_err;
new_data=' S_sys_res R0_res thetaR0_res R1_res E_res thetaR1_res';
new_data=[new_data,' thetaE_res ord_err_res'];
findstr(new_data,var_svd2);
if isempty(findstr(new_data,var_svd2))
    var_svd2=[var_svd2,new_data];
end
sv_dat=[var_svd2,' var_svd2'];
sv=['save ',data_file2,sv_dat];eval(sv);
disp(['The following variables were saved in ',data_file2,'.mat']);
disp(' ')
disp(sv_dat)
disp('----- hit <rtm> to continue -----')
pause
%This completes the second approach to ordering.
%+++++
disp('This is the end of pr2b.m')
disp('pr3.m will take data from pr2b.m and display the ordering')
disp('and the associated errors. From this the user can make')
disp('further reduction and obtain a final model.')
ans=input('Do you want to run pr3.m? [y,Y,<ret>=yes] >> ','s');
if (isempty(ans) | ans=='y' | ans=='Y')
    pr=['pr3(','''',data_file,'''',')'];eval(pr)
    return
end
return
%+++++
%               That's the end of pr2b.m
%+++++

```

pr3.m

```

function []=pr3(data_file)
%+++++
% FILE:
%       pr3.m       - This routine uses data from {data_file}.mat

```

```

%               for SEQUENTIAL and RESIDUE ordering.
%
% INPUT:
%   data_file   - File name that contains all data necessary
%                 for analysis.
%                 If not provided, user will be queried.
%
% OUTPUT:
%   None.
%
% GLOBAL:
%   None.
%
% CALLS:
%   sys2obsv_can - A routine to take any system and transform
%                 it into canonical observable form.
%   discr.m      - A routine that discretizes the continuous
%                 model.
%   rm_fig.m     - A routine that allows for the removal of
%                 figures.
%
% DATA FILES WRITTEN:
%   data_file.mat - A final model of the system data.
%
% DATA FILES READ:
%   data_file.mat - SEQUENTIAL and RESIDUE ordering of model
%                 data.
%
% AUTHOR:
%   Steve Hietpas August 1993
%
% REFERENCES:
%   None.
%
% SYNTAX:
%   pr3({data_file});
%   or
%   pr3;
%+++++
format short e
%+++++
% Get data.
%+++++

```

```

disp('Now running pr3.m')
default_data='ex_uyt';
if nargin~=1
    what
    strg=['Enter a data file name [',default_data,'] >> '];
    data_file=input(strg,'s');
    if isempty(data_file)
        data_file=default_data;
    end
end
ld=['load ',data_file];eval(ld)
data_file2=[data_file,'_ord'];
ld=['load ',data_file2];eval(ld);
data_file3=[data_file,'_pr3'];
if exist([data_file3,'.mat'])
    ld=['load ',data_file3];eval(ld)
end
T=DT;
tt=tsd;
yy=ysd2;N=length(yy);
L=length(S_sys);

%+++++
notdone=1;
flag=0;
while notdone
    ans1=menu('Menu for ordering choice',...
        'SEQUENTIAL',...           %Selection 1
        'RESIDUE',...             %Selection 2
        'Individual Mode Selection',... %Selection 3
        'Construct Final Model',... %Selection 4
        'Quit');                  %selection 5
    if (ans1==1 | ans1==2)
        flag=1;
        if ans1==1
            disp('You have selected SEQUENTIAL')
            S_sys=S_sys_seq;
            R0= R0_seq;
            thetaR0=thetaR0_seq;
            R1=R1_seq;
            thetaR1=thetaR1_seq;
            thetaE=thetaE_seq;
            ord_err=ord_err_seq;

```

```

E=E_seq;
ss='The following data is for SEQUENTIAL ordering';
else
    disp('You have selected RESIDUE')
    S_sys=S_sys_res;
    R0=R0_res;
    thetaR0=thetaR0_res;
    R1=R1_res;
    thetaR1=thetaR1_res;
    thetaE=thetaE_res;
    ord_err=ord_err_res;
    E=E_res;
    ss='The following data is for RESIDUE ordering';
end
if ((fit==1) |(fit==3)) %Non-zero initial conditions exist
    ss3='Initial condition residues are not shown.';
end
R1len=length(R1);
num=[1:1:R1len]';
ord=[num S_sys ord_err];
ss1='    Mode #                Modes                ';
ss1=[ss1,'Error(dB)'];
ss2='-----';
ss2=[ss2,'-----'];
disp(ss),disp(ss1),disp(ss2)
disp(ord)
if ((fit==1) |(fit==3)) %Non-zero initial conditions exist.
ss3='          Modes                Residues                ';
ss3=[ss3,'    Init. Cond.  Res.'];
ss4='-----';
ss4=[ss4,'-----'];
    disp(ss3)
    disp(ss4)
    disp([S_sys R1 E])
end
%tl='Digital Prony fit';
figure
plot(ord_err,'*'),xlabel('MODE #'),
ylabel('ERROR (dB)'),grid%,title(tl)
time_stamp,dir_stamp,file_stamp(data_file,data_file2);
strg='How many terms do you wish to keep? [<ret> = all]'
strg=[strg,' 0=quit >> '];
jk=input(strg);

```

```

disp(' ')
if jk~=0
    if isempty(jk)
        disp('You have chosen to keep all modes.')
        jk=R1len;
    end
    RR1=R1(1:jk);
    P=S_sys(1:jk);
    if ((fit==1) | (fit==3)) %Non-zero initial conditions exists.
        EE=E(1:jk);
    ss1='          Modes          Residues          ';
    ss1=[ss1,' Init. Cond. Res.'];
    ss2='-----';
    ss2=[ss2,'-----'];
    disp(ss1)
    disp(ss2)
    disp([P RR1 EE])
    else
        EE=[];
        ss1='          Modes          Residues';
        ss2='-----';
        disp(ss1)
        disp(ss2)
        disp([P RR1])
    end
    %+++++
    %Plotting section
    xl='TIME (seconds)';
    yl='y(t)';
    Lyy=length(yy);
    if ((fit==1) | (fit==2)) %Feedthru term exists.
        ytemp=real(thetaR0*R0);
    else
        ytemp=zeros(Lyy,1);
    end
    if ((fit==1) | (fit==3)) %Non-zero initial conditions exists.
        yhat=ytemp+...
            real(thetaR1(:,1:jk)*R1(1:jk)+thetaE(:,1:jk)*E(1:jk));
    else
        yhat=ytemp+real(thetaR1(:,1:jk)*R1(1:jk));
    end
    disp('You have 4 strings of information to place, use mouse.')
    n2=10^(ord_err(jk)/20);

```

```

n2db=20*log10(n2);
tc='ACTUAL OUTPUT ___';
tc2='MODEL OUTPUT -.-.-';
tc3=['Number of Eigenvalues are: ',int2str(jk)];
n2dif=['The difference is: ',num2str(n2),' (',num2str(n2db),' dB)'];
s_m=str2mat(tc2,tc,tc3,n2dif);
xmax=max(tt);xmin=min(tt);
ymax2=max([yy;yhat]);ymin2=min([yy;yhat]);
%tl='Digital Prony fit';
figure
axis([xmin xmax ymin2 ymax2]);
plot(tt,yy,'-',tt,yhat,'-.'),xlabel(xl),ylabel(yl),grid,
%,title(tl)
gtext(s_m),
time_stamp,dir_stamp,file_stamp(data_file,data_file2);
else
    flag=0;
end
elseif ans1==3
    disp('You have selected Individual Mode')
    flag=1;
    %+++++
    %Data display section for selecting specific eigenvalues.
    S_sys=S_sys_res;
    R0= R0_res;
    thetaR0=thetaR0_res;
    R1=R1_res;
    E=E_res;
    thetaR1=thetaR1_res;
    thetaE=thetaE_res;
    ord_err=ord_err_res;
    selection=[];
    R1len=length(R1);
    num=[1:1:R1len]';
    ord=[num S_sys R1];
    ss='The following data is from RESIDUE ordering';
    ss1=' Mode Number      Eigenvalues      System Residues ';
    ss2=' -----';
    if ((fit==1) |(fit==3)) %Non-zero initial conditions exist.
ss3='Initial condition residues are not shown, but are included.';
        disp(ss3),disp(' '),disp(ss),disp(ss1),disp(ss2)
    else
        disp(ss),disp(ss1),disp(ss2)
    end
end

```

```

end
disp(ord)
se=input('Which eigenvalue/residue pairs? i.e. [1,2,6,7,9] > ');
if isempty(se)
    se=1:Rilen;
end
selen=length(se);
%+++++
% Plotting section
xl='TIME (seconds)';
yl='y(t)';
Lyy=length(yy);
if ((fit==1) | (fit==2)) %Feedthru term exists.
    yhat=real(thetaR0*R0);
else
    yhat=zeros(Lyy,1);
end
sse='';RR1=[];EE=[];P=[];
if ((fit==1) | (fit==3)) %Non-zero initial conditions exists.
    for kk=1:selen
        ii=se(kk);
        yhat=yhat+real(thetaR1(:,ii)*R1(ii) + thetaE(:,ii)*E(ii));
        sse=[sse,int2str(ii),' '];
        RR1=[RR1;R1(ii)];EE=[EE;E(ii)];P=[P;S_sys(ii)];
    end
else
    for kk=1:selen
        ii=se(kk);
        yhat=yhat+real(thetaR1(:,ii)*R1(ii));
        sse=[sse,int2str(ii),' '];
        RR1=[RR1;R1(ii)];P=[P;S_sys(ii)];
    end
    EE=[];
end
disp('You have 5 strings of information to place, use mouse.')
sse=['[ ',sse,']'];
n2=norm(yhat-yy)/norm(yy);
n2db=20*log10(n2);
tc='ACTUAL OUTPUT ___';
tc2='MODEL OUTPUT -.-.-';
tc3=['Eigenvalues/Residue used from RESIDUE ordering are:'];
n2dif=['The difference is: ',num2str(n2),' (',num2str(n2db),' dB)'];
s_m=str2mat(tc2,tc,tc3,n2dif,sse);

```

```

xmax=max(tt);xmin=min(tt);
ymax2=max([yy;yhat]);ymin2=min([yy;yhat]);
%tl='Digital Prony fit';
figure
axis([xmin xmax ymin2 ymax2]);
plot(tt,yy,'-',tt,yhat,'-.'),xlabel(xl),ylabel(y1),grid,
%title(tl),
gtext(s_m),
time_stamp,dir_stamp,file_stamp(data_file,data_file2);
elseif ans1==4
disp('You have selected Construct Final Model')
if flag
    if ((fit==1) | (fit==2)) %Feedthru term exists.
        feed=R0;
    else
        feed=0.0;
    end
    [num,den]=residue(RR1,P,feed);
    num=real(num);den=real(den);
    [Am,Bm,Cmc,Dmc]=tf2ss(num,den);
    [Fm,Gm]=discr(Am,Bm,T); % Discretize the model.
    [Fm,Gm,Cm,Dm,flag1]=sys2obsv_can(Fm,Gm,Cmc,Dmc);
    eigFm=eig(Fm);
    [Am,Bm]=discr(Fm,Gm,T);
    eigAm=eig(Am);
    L=length(eigFm);
    disp('The modes of the model are:')
    disp(eigAm)
    new_data=' Fm Gm Cm Dm';
    if flag1==1
        disp('The system has been reduced due to unobservability!')
    disp('You should choose individual modes via the menu selection.')
    end
    new_data=[new_data,' num den R0 RR1 EE P yhat'];
    if ~exist('var_svd3')
        var_svd3=new_data;
    end
    if isempty(findstr(new_data,var_svd3))
        var_svd3=[var_svd3,new_data];
    end
    sv_dat=[var_svd3,' var_svd3'];
    sv=['save ',data_file3,sv_dat];eval(sv);
disp(['The following variables were saved in ',data_file3,'.mat']);

```

```

disp(' ')
disp(sv_dat)
disp('----- hit <rtn> to continue -----')
pause
else
disp('You have not selected any mode/residue pairs yet.')
notdone=1;
end
else
disp('You have selected Quit')
notdone=0;
end
end
disp(' ')
model_file=[data_file,'_mod'];
var_svd_m=' Am Bm Cmc Dmc Fm Gm Cm Dm eigAm eigFm sys EE P yhat';
sv_dat=[var_svd_m,' var_svd_m'];
sv=['save ',model_file,sv_dat];eval(sv)
disp('This is the end of pr3.m')
return
%+++++
%               That's the end of pr3.m
%+++++

```

prony_in.m

```

function [usd,tsd]=prony_in(system,u_file,FT)
%+++++
% File:
%      prony_in      - This function generates the input drive
%                      signal for the following system:
%
%                       $x((k+1)T) = F*x(kT) + G*u(kT)$ 
%
%                       $y(kT) = C*x(kT) + D*u(kT)$ 
%
% INPUT:
%      system      - The data file that contains the system
%                   matrices and DT.
%      u_file      - The name of the input probing file.
%      FT          - The final time for evaluation of system;
%
% OUTPUT:

```

```

%      usd          - Vector of input signal;
%      tsd          - Vector of time signal;
%
% GLOBAL:
%      None.
%
% CALLS:
%      get_prony_in - A routine that retrieves information for
%                    u_file.
%
% DATA FILES WRITTEN:
%      prony_in
%
% DATA FILES READ:
%      None.
%
% AUTHOR:
%      Steve Hietpas August 1993
%
% REFERENCES:
%      None.
%
% SYNTAX:
%      [usd,tsd]=prony_in(system,u_file,FT)
%+-----+
global fig_table
disp(' '),disp(' ')
disp('Running the discrete difference equation solver prony_in.m')
ld=['load ',system];eval(ld);
if exist('prony_in.mat')
    load prony_in
else
    dd=[];mpi=[];cc=[];ss=[];
end
disp(' ')
[dd,mpi,cc,ss]=get_prony_in(dd,mpi,cc,ss);
usd=[];tsd=[];
ee=[]; %zero-value initial conditions are always assumed for these
      %comparisons.
t=0;tt=fix(FT/DT);
for k=1:tt
    u=feval(u_file,t,dd,mpi,cc,ss);
    usd=[usd;u];

```

```

    tsd=[tsd;t];
    t=t+DT;
end
var_svd_in=' usd tsd dd mpi cc ss DT';
sv_dat=[var_svd_in,' var_svd_in'];
sv=['save prony_in',sv_dat];eval(sv)
return
%+++++
%               That's the end of prony_in.m
%+++++

```

u_prony.m

```

function uu = u_prony(t,dd,mpi,cc,ss)
%+++++
%  FILE:
%      u_prony    - This routine provide a piecewise
%                  discontinuous input probing signal for
%                  Prony identification.
%
%  INPUT:
%      t - time;
%      dd - vector of duration factors in seconds;
%      mpi - number of modes/interval;
%      cc - vector of amplitude factors;
%      ss - vector of mode factors.
%
%      e.g. a sine wave of 5 Hz, amplitude 1,
%      for 0 to 2 seconds would be:
%
%      dd=2; mpi=2; cc=[-i/2 i/2]; ss=[i*2*pi*5 -i*2*pi*5]..
%
%      One can see this from Euler's identity, just
%      plug in appropriate values:
%
%      
$$\sin(2\pi*5*t) = cc(1)*\exp(ss(1)*t) + cc(2)*\exp(ss(2)*t)$$

%      
$$= [\exp(i*2\pi*5)t - \exp(-i*2\pi*5)t]/(2i)$$

%
%  OUTPUT:
%      uu - input control signal.
%
%  GLOBAL:
%      None.

```

```

%
% CALLS:
%     None.
%
% DATA FILES WRITTEN:
%     None.
%
% DATA FILES READ:
%     None.
%
% AUTHOR:
%     Steve Hietpas October 1992
%
% REFERENCES:
%     None.
%
% SYNTAX:
%     uu = u_prony(t,dd,mpi,cc,ss);
%+++++
q=length(dd);
if t<=dd(q)
    ii=max(find(t>dd));
    if isempty(ii)
        m_i=0;
        ii=1;
    elseif ii<q
        m_i=0;
        for k=1:ii
            m_i=m_i+mpi(k);
        end
        ii=ii+1;
    else
        end
    if mpi(ii)~=0
        if ii==1
            ddd=0;
        else
            ddd=dd(ii-1);
        end
        uu=0.0;
        for k=m_i+1:m_i+mpi(ii)
            uu=uu+cc(k)*exp(ss(k)*(t-ddd));
        end
    end
end

```

```

else
    uu=0.0;
end
else
    uu=0.0;
end
return
%+++++
%               That's the end of u_prony.m
%+++++

```

get_prony_in.m

```

function [dd,mpi,cc,ss]=get_prony_in(dd,mpi,cc,ss)
%+++++
% FILE:
%     get_prony_in - Routine for getting input probing information.
%
% INPUT:
%     dd           - delay intervals for input signal;
%     mpi          - number of modes per interval;
%     cc           - vector of amplitude factors;
%     ss           - vector of mode factors.
% OUTPUT:
%     dd,mpi,cc,ss
%
% CALLS:
%     None.
%
% GLOBAL:
%     u_file       - Input file name for probing.
%
% DATA FILES WRITTEN:
%     The output data is save in a file named by u_file.
%
% DATA FILES READ:
%     None.
%
% AUTHORS:
%     Steve Hietpas      November 1993
%
% REFERENCES:
%     None.

```

```

%
% SYNTAX:
% [dd,mpi,cc,ss]=get_prony_in(dd,mpi,cc,ss);
%+++++
global u_file
strg='Do you want an example for how to enter data?';
strg=[strg,' [n,N,<rtn>=no] >> '];
ans1=input(strg,'s');
if ~(isempty(ans1) | ans1=='n' | ans1=='N')
    disp(' '),disp('+++++ EXAMPLE +++++'),disp(' ')
    in_exmpl;
    disp('+++++ END EXAMPLE +++++'),disp(' ')
end
if ~isempty(dd)
    disp('The previous values for delays, dd, were:')
    disp(dd)
    disp('The previous values for modes/interval, mpi, were:')
    disp(mpi)
    disp('The previous values for modes, ss, were:')
    disp(ss)
    disp('The previous values for mode amplitudes, cc, were:')
    disp(cc)
    disp(' ')
    ans=input('Do you want to keep these? [y,Y,<ret>=yes] >> ','s');
    if ~(isempty(ans) | ans=='y' | ans=='Y')
        disp('The previous values for delays, dd, were:')
        disp(dd)
        disp('What values for dd_new [...] (<rtn> for old values)? ')
        dd_new=input('dd_new: ');
        if isempty(dd_new)
            dd_new=dd;
        end
        disp('The previous values for modes/interval, mpi, were:')
        disp(mpi)
        disp('What values for mpi_new [...] (<rtn> for old values)? ')
        mpi_new=input('mpi_new: ');
        if isempty(mpi_new)
            mpi_new=mpi;
        end
        disp('The previous values for modes, ss, were:')
        disp(ss)
        disp('What values for ss_new [...] (<rtn> for old values)? ')
        ss_new=input('ss_new: ');
    end
end

```

```

    if isempty(ss_new)
        ss_new=ss;
    end
    disp('The previous values for amplitudes, cc, were:')
    disp(cc)
    disp('What values for cc_new [...]? (<rtn> for old values) ')
    cc_new=input('cc_new: ');
    if isempty(cc_new)
        cc_new=cc;
    end
    dd=dd_new;
    mpi=mpi_new;
    ss=ss_new;
    cc=cc_new;
end
else
    disp('What values for dd [...]? ')
    dd=input('dd: ');
    disp(' ')
    disp('What values for mpi [...]? ')
    mpi=input('mpi: ');
    disp(' ')
    disp('What values for ss [...]? ')
    ss=input('ss: ');
    disp(' ')
    disp('What values for cc [...]? ')
    cc=input('cc: ');
    disp(' ')
end
disp('The final choices are:')
disp(' ')
disp('dd: ')
disp(dd)
disp(' ')
disp('mpi: ')
disp(mpi)
disp(' ')
disp('ss: ')
disp(ss)
disp(' ')
disp('cc: ')
disp(cc)
disp(' ')

```



```

strg='0 1T 2T 3T 4T 5T 6T 7T 8T 9T 10T 11T';
disp([strg,' 12T 13T'])
disp(' ')
disp('Interval 1, 2 modes')
disp('Let the input be c1*exp(s1*t)+c2*exp(s2*t) for 0-2T seconds,')
disp(' ')
disp('Interval 2, 0 modes')
disp('Let input be off for 2T-4T seconds.')
disp(' ')
disp('Interval 3, 4 modes')
strg='Let the input be c3*exp(s3*t)+c4*exp(s4*t)+c5*exp(s5*t)';
disp([strg,'+c6*exp(s6*t)'])
disp(' ')
disp('for 4T-8T seconds, and then off for remaining time.')
disp(' ')
disp('For this example set T = 1 (second).')
disp(' ')
strg='You will be prompted for the times that the input';
disp([strg,' intervals end.'])
disp('To this prompting you would enter: [1 4 8].')
disp(' ')
strg='You will next be prompted for the number of modes for';
disp([strg,' each interval.']);
disp('To this prompting you would enter: [2 0 4].')
disp(' ')
disp('You will next be prompted for amplitude values.')
disp('To this prompting you would enter: [c1 c2 c3 c4 c5 c6].')
disp(' ')
disp('You will next be prompted for mode values.')
disp('To this prompting you would enter: [s1 s2 s3 s4 s5 s6].')
disp(' ')
%+++++
%               That's the end of in_exmpl.m
%+++++

```

rm_mode.m

```

function [Modes]=rm_mode(modes)
%+++++
% FILE:
%       in_exmpl - Removes modes produced by prib.m and other
%                   m-files.
%

```

```

% INPUT:
%         modes - vector of modes.
%
% OUTPUT:
%         Modes - new vector of modes.
%
% CALLS:
%         None.
%
% GLOBAL:
%         None.
%
% DATA FILES WRITTEN:
%         None.
%
% DATA FILES READ:
%         None.
%
% AUTHORS:
%         Steve Hietpas April 1993
%
% REFERENCES:
%         None.
%
% SYNTAX:
%         [Modes]=rm_mode(modes);
%+++++
mode_len=length(modes);
for ii=1:mode_len
    fprintf('\n\n')
    disp(modes(ii))
    ans=input('Remove this above mode? [n,N,<rtm>=no] >> ','s');
    if (isempty(ans) | ans=='n' | ans=='N')
        Modes=[Modes;modes(ii)];
    end
end
return
%+++++
%         That's the end of rm_mode.m
%+++++

```

xdisp_mode.m

```

function []=xdisp_mode(mode,title)
%+++++
% FILE:
%       xdisp_mode - Display section for modes produced by pr1.m
%                   and other m-files for X-windows environment.
%
% INPUT:
%       modes      - modes to display
%       title      - name of signal to be used in header of
%                   displayed modes.
%
% OUTPUT:
%       None.
%
% CALLS:
%       None.
%
% GLOBAL:
%       None.
%
% DATA FILES WRITTEN:
%       None.
%
% DATA FILES READ:
%       None.
%
% AUTHORS:
%       Steve Hietpas April 1993
%
% REFERENCES:
%       None.
%
% SYNTAX:
%       []=xdisp_mode(mode,title);
%+++++
mode_len=length(mode(:,1));
mode=[real(mode) imag(mode)];
ss1=[' The ',title,' modes are:'];
ss2='      Real      Imag';
ss3=' -----';
disp(ss1)

```

```

disp(ss2)
disp(ss3)
disp(mode)
return
%+++++
%               That's the end of xdisp_mode.m
%+++++

```

make_obsvble.m

```

function [num,den,C,D,flag] = make_obsvble(num,den,C,D)
%+++++
% FILE:
%     make_obsvble   - Take any arbitrary system and make it
%                     observable.
%
% INPUT:
%     if nargin is 2
%         num         - The numerator polynomial.
%         den         - The denominator polynomial.
%     elseif nargin is 4
%         num         - State-space transition matrix, A
%         den         - State-space input matrix, B
%         C           - State-space output matrix,
%         D           - State-space direct matrix.
%
% OUTPUT:
%     if nargout is 3
%         num         - The numerator polynomial.
%         den         - The denominator polynomial.
%         flag        - flag==0 if system was not reduced.
%                     - flag==1 if system was reduced.
%     elseif nargout is 5
%         num         - State-space transition matrix, A
%         den         - State-space input matrix, B
%         C           - State-space output matrix,
%         D           - State-space direct matrix.
%         flag        - flag==0 if system was not reduced.
%                     - flag==1 if system was reduced.
%
% GLOBAL:
%     None.
%

```

```

% CALLS:
%     ss2tf      - State-space to transfer function (Signal toolbox)
%     tf2ss      - Transfer function to state-space (Signal toolbox)
%
% DATA FILES WRITTEN:
%     None.
%
% DATA FILES READ:
%     None.
%
% AUTHOR:
%     Steve Hietpas December 1993
%
% REFERENCES:
%     None.
%
% SYNTAX:
%     [num,den,flag] = make_obsvble(num,den)
%     or
%     [A,B,C,D,flag] = make_obsvble(A,B,C,D)
%
%+-----+
tol=input('Enter tolerance. [<ret>=0.01] >> ');
if (isempty(tol))
    tol=0.01;
end
if nargin==2
    [A,B,C,D]=tf2ss(num,den);
else
    A=num;
    B=den;
    [num,den]=ss2tf(A,B,C,D,1);
end
num_old=num;
den_old=den;
A_old=A;
B_old=B;
C_old=C;
D_old=D;
[n,n] = size(A);
obs_M=obsv(A,C); %The observability matrix.
if rank(obs_M)<n
    notdone=1;

```

```

while notdone
    % Try to make system observable.
    zer=roots(num);
    pol=roots(den);
    lz=length(zer);
    lp=length(pol);
    zer_keep=[1:1:lz]';
    pol_keep=[1:1:lp]';
    k_rid=[];
    kk_rid=[];
    for k=1:lp
        for kk=1:lz
            diff=min(abs(pol(k)-zer(kk)),abs(pol(k)-conj(zer(kk))));
            if diff<=tol %mark these endices for modes to remove.
                if isempty(find(k==k_rid))
                    k_rid=[k_rid; k];
                end
                if isempty(find(kk==kk_rid))
                    kk_rid=[kk_rid; kk];
                end
            end
        end
    end
    l_krid=length(k_rid);
    if ~isempty(l_krid)
        for k=1:l_krid
            zer_keep=zer_keep(find(kk_rid(k)~=zer_keep));
            pol_keep=pol_keep(find(k_rid(k)~=pol_keep));
        end
    end
    zer=zer(zer_keep);
    pol=pol(pol_keep);
    num=poly(zer);
    den=poly(pol);
    [A,B,C,D]=tf2ss(num,den);
    A=real(A);
    B=real(B);
    C=real(C);
    D=real(D);
    [n,n]=size(A);
    obs_M=obsv(A,C); %The observability matrix.
    if rank(obs_M)<n
        disp('An attempt was made to make the system observable.')
    end
end

```

```

disp('You may increase the value for tol')
disp('to make it observable.')
ans1=input('Do you want to try again? [y,Y,<ret>=yes] >> ','s');
if (isempty(ans1) | ans1=='y' | ans1=='Y')
    tol=input('Enter a new tolerance value: >> ');
    notdone=1;
    num_old=num;
    den_old=den;
    A=A_old;
    B=B_old;
    C=C_old;
    D=D_old;
    [n,n] = size(A);
else
    notdone=0;
    if nargout==3
        num=num_old;
        den=den_old;
        flag=0;
        C=flag;
    elseif nargout==5
        num=A_old;
        den=B_old;
        C=C_old;
        D=D_old;
        flag=0;
    else
        disp('No output requested.')
    end
end
else
    notdone=0;
    disp('The system was unobservable but now is.')
    if nargout==3
        flag=1;
        C=flag;
    elseif nargout==5
        num=A;
        den=B;
        flag=1;
    else
        disp('No output requested.')
    end
end

```

```

    end
end
else
    disp('The system is observable.')
    if nargout==3
        num=num_old;
        den=den_old;
        flag=0;
        C=flag;
    elseif nargout==5
        num=A_old;
        den=B_old;
        C=C_old;
        D=D_old;
        flag=0;
    else
        disp('No output requested.')
    end
end
return
%+++++
%               That's the end of make_obsvble.m
%+++++

```

obsv.m

```

function ob = obsv(a,c)
%+++++
% FILE:
%     obsv - Formulates the observability matrix.
%           Borrowed from the control toolbox, no copyright.
%     ob = [a; ca; ca^2 ...]
%
% INPUT:
%     a    - State-space matrix.
%     c    - State-space output matrix.
% OUTPUT:
%     ob   - observability matrix.
%
% GLOBAL:
%     None.
%
% CALLS:

```

```

%      None.
%
% DATA FILES WRITTEN:
%      None.
%
% DATA FILES READ:
%      None.
%
% AUTHOR:
%      Unknown.
%
% REFERENCES:
%      None.
%
% SYNTAX:
%      ob = obsv(a,c);
%+++++
[m,n] = size(a);
ob = c;
for i=1:n-1
    ob = [c; ob*a];
end
return
%+++++
%                        That's the end of obsv.m
%+++++

```

sys2obsv_can.m

```

function [num,den,C,D,flag] = sys2obsv_can(num,den,C,D)
%+++++
% FILE:
%      sys2obsv_can    - Take any system and place in observable
%                      canonical form.
%
% INPUT:
%      if nargin is 2
%          num          - The numerator polynomial.
%          den          - The denominator polynomial.
%      elseif nargin is 4
%          num          - State-space transition matrix, A
%          den          - State-space input matrix, B
%          C            - State-space output matrix,

```

```

%           D           - State-space direct matrix.
%
% OUTPUT:
%   if nargout is 3 %This is an unlikely request.
%       num           - The numerator polynomial.
%       den           - The denominator polynomial.
%       flag           - flag==0 if system was not reduced.
%                       flag==1 if system was reduced.
%   elseif nargout is 5
%       num           - State-space transition matrix, A
%       den           - State-space input matrix, B
%       C             - State-space output matrix,
%       D             - State-space direct matrix.
%       flag           - flag==0 if system was not reduced.
%                       flag==1 if system was reduced.
%
% GLOBAL:
%   None.
%
% CALLS:
%   ss2tf           - State-space to transfer function (Signal toolbox)
%   tf2ss           - Transfer function to state-space (Signal toolbox)
%
% DATA FILES WRITTEN:
%   None.
%
% DATA FILES READ:
%   None.
%
% AUTHOR:
%   Steve Hietpas December 1993
%
% REFERENCES:
%   [1] For transformation approach.
%       G. H. Hostetter, "Digital Control System Design,"
%       pp. 193-195, Holt, Rinehart and Winston, Inc.
%
%   [2] For transformation approach.
%       K. Ogata, "Discrete-Time Control Systems,"
%       pp. 491-493, Printice-Hall, Inc.
%
% SYNTAX:
%   [num,den,flag]={sys2obsv_can(num,den) or sys2obsv_can(A,B,C,D)}

```

```

%      or
%      [A,B,C,D,flag]={sys2obsv_can(num,den) or sys2obsv_can(A,B,C,D)}
%+++++
flag=0;
if nargin==2
    [A,B,C,D]=tf2ss(num,den);
else
    A=num;
    B=den;
    [num,den]=ss2tf(A,B,C,D,1);
end
num_old=num;
den_old=den;
A_old=A;
B_old=B;
C_old=C;
D_old=D;
[n,n] = size(A);
obs_M=obsv(A,C); %The observability matrix.
if rank(obs_M)<n
    disp('The system is not observable.')
    disp('Do you want to make the system observable')
    disp('by removing pole-zero matches?')
    rem=input('Remove matches? [n,N,<ret>=No] >> ','s');
    if ~(isempty(rem) | rem=='n' | rem=='N')
        [A,B,C,D,flag]=make_obsvble(A,B,C,D);
        [num,den]=ss2tf(A,B,C,D,1);
        [n,n] = size(A);
    end
end
do_canon=1;
if do_canon
    char_eq_A=poly(A);
    Q=C;
    for k=1:n-1
        Q(k+1,:)=Q(k,:)*A+char_eq_A(k+1)*C;
    end
    cq=cond(Q);
    ans1='n';
    if ~(isempty(ans1) | ans1=='n' | ans1=='N') % See reference [1]
        iQ=inv(Q);
        A=real(Q*A*iQ);
        B=real(Q*B);
    end
end

```

```

C=real(C*iQ);
tol=input('Enter tolerance. [<ret>=1e-4] >> ');
if (isempty(tol))
    tol=1e-4;
end
for k=1:n
    I=find(abs(A(:,k))<tol);
    for kk=1:length(I);
        A(I(kk),k)=0;
    end
end
I=find(abs(B)<tol);
for kk=1:length(I);
    B(I(kk))=0;
end
I=find(abs(C)<tol);
for kk=1:length(I);
    C(I(kk))=0;
end
else % See reference [2]
    A=[zeros(n,1) [eye(n-1,n-1);zeros(1,n-1)]];
    A(:,1)=-[den(2:n+1)]';
    B=[num(2:n+1)-den(2:n+1)*num(1)]';
    C=[1 zeros(1,n-1)];
end
[n,n] = size(A);
obs_M=obsv(A,C); %The observability matrix.
if rank(obs_M)<n
    disp('The system is not observable.')
    disp('Do you want to make the system observable')
    disp('by removing pole-zero matches?')
    rem=input('Remove matches? [n,N,<ret>=No] >> ','s');
    if ~(isempty(rem) | rem=='n' | rem=='N')
        [A,B,C,D,flag]=make_observble(A,B,C,D);
        [num,den]=ss2tf(A,B,C,D,1);
        [n,n] = size(A);
    else
        do_canon=0;
        disp('Warning: The system is unobservable!')
    end
else
    disp('The transformed system is observable.')
    do_canon=0;

```

```

    end
else
    A=A_old;
    B=B_old;
    C=C_old;
    D=D_old;
    flag=0;
end
if nargout==3 %This is an unlikely request.
    [num,den]=ss2tf(A,B,C,D,1);
    C=flag;
elseif nargout==5
    num=A;
    den=B;
else
    disp('No output requested.')
end
return
%+++++
%
%               That's the end of sys2obsv_can.m
%+++++

```

moon1.m

```

function []=moon1(data_file)
%+++++
% FILE:
%
%       moon1.m   - This routine uses the data from
%
%                  "data_file".mat. The vectors or matrices
%                  of data, usd and ysd2, are used to construct
%                  the Hankel matrices which in turn is used for
%                  the estimation of the discrete
%                  state-space model, (y=ysd2, u=usd):
%
%                  
$$x((k+1)T) = A*x(kT) + B*u(kT)$$

%
%                  
$$y(kT) = C*x(kT) + D*u(kT)$$

%
%                  This algorithm assumes u(kT) sufficiently
%                  excites the system modes. For more on
%                  'sufficient' excitation see notes (i,ii,iii)
%                  on pp. 222 and 223 of Moonen, et al.
%
%

```

```

% INPUT:
%     data_file     - Data file name, if left empty user will be
%                   queried for a file name.
%
% OUTPUT:
%     None.
%
% GLOBAL:
%     None.
%
% CALLS:
%     moon2.m       - see moon2.m
%
% DATA FILES WRITTEN:
%     moon1.mat     - see saving section.
%
% DATA FILES READ:
%     {'data_file'}.mat
%
% AUTHOR:
%     Steve Hietpas April, 1993
%
% REFERENCES:
%
%     [1] M. Moonen, B. De Moor, L. Vandenberghe, J. Vandewalle,
%     "On- and off-line identification of linear state-space models,"
%     Int. J. Contr., 1989, vol. 49, no. 1, pp. 219-232.
%     The specific algorithm used is taken from Algorithm 1,
%     pp. 228-229. of (1).
%
% SYNTAX:
%     moon1(data_file);
%     or
%     moon1;
%+-----+
disp('Running the function moon1.m')
default_data='ex_uyt';
if nargin~=1
    what
    strg=['Enter a data file name [<ret>=',default_data,'] >> '];
    data_file=input(strg,'s');
    if isempty(data_file);
        data_file=default_data;

```

```

end
end
ld=['load ',data_file];eval(ld);
%The following code allows the user to specify at what time
%data is to be used for system identification.
tsdi=length(tsd);
[ysdi,ysdj]=size(ysd2); [usdi,usdj]=size(usd);
m=usdj; %m is the number of inputs.
l=ysdj; %l is the number of outputs.
u=flipud(rot90(usd)); y=flipud(rot90(ysd2));
%
%Building the Hankel Matrix of equation (6), pg. 223 of (1).
%First determine the number of rows of H,
% the concatenation of H1 and H2.
notdone=1;
while notdone
    notdone=0;
    notdone1=1;
    while notdone1
        notdone1=0;
        %To maximize the use of the data determine from the user supplied
        %g what the dimension of H should be.
        Hi=2*fix((ysdi+1)/(g+2));%Total number of rows for H will
                                %be (m+1)*Hi.
        Hj=fix(g*Hi/2); used=Hi+Hj-1; U1i=m*Hi/2;Y1i=l*Hi/2;
        fprintf('\n\n There are %g data points of which %g will be used.',...
            ysdj,used);
        fprintf('\n\n The dimension of U1 and Y1 are %g and %g by %g,...
            respectively.',U1i,Y1i,Hj);
        strg='Do you want to try another multiplication factor,';
        strg=[strg,' [n,N,<ret>=no]? >> '];
        ans=input(strg,'s');
        if (isempty(ans) | ans=='n' | ans=='N')
            notdone1=0;
        else
            notdone1=1;
            g=input('Enter another multiplication factor: ');
        end
    end
end
disp('Constructing Hankel Matrix')
U=[];Y=[];H=[];
for jj=1:Hi
    jjfin=(jj-1)+Hj;

```

```

    ytmp=y(:,jj:jjfin); utmp=u(:,jj:jjfin);
    H=[H;ytmp;utmp];
    Y=[Y;ytmp]; U=[U;utmp];
end
Y1=Y(1:l*Hi/2,:); Y2=Y(l*Hi/2+1:l*Hi,:);
U1=U(1:m*Hi/2,:); % U2=U(m*Hi/2+1:m*Hi,:);
H1=[Y1;U1]; % H2=[Y2;U2];
clear Y U
%This part is from Equation (5) and proof on pg 224 of (1).
rH1=rank(H1); rU1=rank(U1);
if rU1~=m*Hi/2
    fprintf('The rank of U1 is %g \n',rU1)
    fprintf('\n U1 has %g rows \n',m*Hi/2)
    disp('U1 is not full row rank!')
    ans=input('Redo this run? [y,Y,<ret>=yes] ','s')
    if ~(isempty(ans) | ans=='y' | ans=='Y')
        notdone=0;
        n=rH1-rU1;
        ii=U1i/m;
        %Now that H is constructed we can proceed with
        %Step 1 pg 228 of (1).
        disp('Computing SVD')
        [U,S,V]=svd(H); % H=U*S*V'
        var_svd=' U S V H rH1 rU1 ii n m l';
        sv_dat=[var_svd,' var_svd'];
        sv=['save moon1 ',sv_dat];eval(sv)
    else
        notdone=1;
    end
else
    notdone=0;
    n=rH1-rU1;
    ii=U1i/m;
    %Now that H is constructed we can proceed with
    %Step 1 pg 228 of (1).
    disp(' '),disp(' '),disp('Computing SVD'),disp(' '),disp(' ')
    [U,S,V]=svd(H); % H=U*S*V'
    var_svd=' U S V H rH1 rU1 ii n m l';
    sv_dat=[var_svd,' var_svd'];
    sv=['save moon1 ',sv_dat];eval(sv)
end
end
end
%+++++

```

```
%Now run the second part of the system identification program.
disp('Moon1 is done'),pause(2)
moon2(data_file)
disp(' '),disp(' '),disp('Moon2 is done. '),pause(2)
%+++++
return
%+++++
%               That's the end of moon1.m
%+++++
```

moon2.m

```
function []=moon2(data_file)
%+++++
% FILE:
%       moon2.m   - This section is the second stage of the
%                   identification procedure for Moonen et al.
%                   Moon2 should always be executed after having
%                   executed Moon1.m
%
% INPUT:
%       None.
%
% OUTPUT:
%       None.
%
% GLOBAL:
%       fig_table - Table of existing figure numbers.
%
% CALLS:
%       discr.m   - A routine that discretizes the continuous model.
%
% DATA FILES WRITTEN:
%       see saving section.
%
% DATA FILES READ:
%       {'data_file'.mat} - see moon1.m
%
% AUTHOR:
%       Steve Hietpas September 1992
%
% REFERENCES:
%       [1] M. Moonen, B. De Moor, L. Vandenberghe, J. Vandewalle,
```

```

% "On- and off-line identification of linear state-space models,"
% Int. J. Contr., 1989, vol. 49, no. 1, pp. 219-232.
% The specific algorithm used is taken from Algorithm 1,
% pp. 228-229. of (1).
%
% SYNTAX:
%     moon2(data_file);
%     or
%     moon2;
%+++++
disp('Running the function moon2.m')
global fig_table
if nargin~=1
    what
    strg='Enter a data file name [<ret>=orig_dat] >> ';
    data_file=input(strg,'s');
    if isempty(data_file);
        data_file='orig_dat';
    end
end
ld=['load ',data_file];eval(ld);
load moon1
s1=['SNR of colored noise induced on output is, '];
s1=[s1,num2str(sgnr2),' dB'];
s2=['SNR of additive output noise is ',num2str(sgnro),' dB'];
ss=str2mat(s1,s2);
disp('Place 2 strings of information on figure.')
tt='Singular Values of H1';
yy='Singular Value Mag.';xx='Singular Value Order #';
[tim,dat]=time;tim=['The time is ',tim];disp(tim)
figure
fig_table=[fig_table;gcf];
plot(diag(S(1:n,1:n)),'*'),title(tt),xlabel(xx),ylabel(yy),grid
gtext(ss)
time_stamp,dir_stamp,file_stamp(data_file);
notdone=1;
while notdone
    order=input('What is the final choice of system order? ');
    if isempty(order) order=n; end, n=order;
    %*****
    %Form HQ, which is from Theorem 4, page 225:
    %    HQ = U12'*U11*S11*V',
    %        where dim(S11) is taken to be (2*m*ii+n)X(2*m*ii+n)

```

```

%           dim(U11) is taken to be (m*ii+l*ii)X(2*m*ii+n)
%           dim(U12) is taken to be (m*ii+l*i)X(2*l*ii-n)
%Next perform SVD on Hq:
%
%           -           -
%   HQ = [Uq Uq(perp)] | Sq  0 | [Vq Vq(perp)]'
%           | 0    0 |
%           -           -
%           where dim(Uq) is chosen to be (2*l*ii-n)X(n)
% *****
U11=U(1:(m+l)*ii,1:2*m*ii+n);
U12=U(1:(m+l)*ii,2*m*ii+n+1:2*(m+l)*ii);
S1=S(1:2*m*ii+n,:);
HQ=U12'*U11*S1*V';
[UQ,SQ,VQ]=svd(HQ);
%dim(Uq) is taken to be UQ(:,1:n)
Uq=UQ(:,1:n);
%*****
%Form the left and right matrice of Theorem 5, page 227-228:
%
%           -           -
%   USl=| Uq'*U12'*U(m+l+1:(m+l)*(ii+1),:)*S |
%         | U((m+l)*ii+1:(m+l)*ii+m,:)*S      |
%         -           -
%
%           -           -
%   USr=| Uq'*U12'*U(1:(m+l)*ii,:)*S          |
%         | U((m+l)*ii+m+1:(m+l)*(ii+1),:)*S |
%         -           -
%
%   -   -
%   | Fm Gm |=USl*pinv(USr)
%   | Cm Dm |
%   -   -
%
% *****
USl1=[Uq'*U12'*U(m+l+1:(m+l)*(ii+1),:)*S];
USl2=[U((m+l)*ii+1:(m+l)*ii+m,:)*S];
USl=[USl1;USl2];
USr1=[Uq'*U12'*U(1:(m+l)*ii,:)*S];
USr2=[U((m+l)*ii+m+1:(m+l)*(ii+1),:)*S];
USr=[USr1;USr2];
FtoD=USl*pinv(USr);
[ll,mm]=size(FtoD);
Fm=FtoD(1:n,1:n);
Gm=FtoD(1:n,n+1:mm);

```

```

Cm=FtoD(n+1:ll,1:n);
Dm=FtoD(n+1:ll,n+1:mm);
eigFm=eig(Fm);
eigAm=log(eigFm)/DT;
disp('Complete Set of Eigenvalues (z-domain)')
disp('-----')
disp(eigFm)
disp('If some of the real parts of the eigenvalues are')
disp('negative, you should try choosing a larger order.')
ans=input('Try larger order? [n,N,<ret>=no] >> ','s');
if ~(isempty(ans) | ans=='n' | ans=='N')
    notdone=1;
else
    notdone=0;
end
end
[Am,Bm]=discr(Fm,Gm,DT);
[n_mod,d_mod]=ss2tf(Am,Bm,Cm,Dm,1);
n_mod=real(n_mod);d_mod=real(d_mod);
[Am,Bm,Cmc,Dmc]=tf2ss(n_mod,d_mod);
model_file=[data_file,'_mod'];
var_svd_m=' Am Bm Cmc Dmc Fm Gm Cm Dm eigAm eigFm sys';
sv_dat=[var_svd_m,' var_svd_m'];
sv=['save ',model_file,sv_dat];eval(sv)
return
%+++++
%               That's the end of moon2.m
%+++++

```

moon_in.m

```

function [usd,tsd]=moon_in(system,u_file,FT)
%+++++
% File:
%       moon_in      - This function generates the input drive
%                      signal for the following system:
%
%                       $x((k+1)T) = F*x(kT) + G*u(kT)$ 
%
%                       $y(kT) = C*x(kT) + D*u(kT)$ 
%
% INPUT:
%       system      - The data file that contains the system

```

```

%          matrices and DT.
%          u_file      - The name of the input probing file.
%          FT          - The final time for evaluation of system;
%
% OUTPUT:
%          usd          - Vector of input signal;
%          tsd          - Vector of time signal;
%
% GLOBAL:
%          None.
%
% CALLS:
%          get_moon_in  - A routine that retrieves information for
%                        u_file.
%
% DATA FILES WRITTEN:
%          None.
%
% DATA FILES READ:
%          None.
%
% AUTHOR:
%          Steve Hietpas January 1994
%
% REFERENCES:
%          None.
%
% SYNTAX:
%          [usd,tsd]=moon_in(system,u_file,FT)
%+++++
disp(' '),disp(' ')
disp('Running the function moon_in.m')
ld=['load ',system];eval(ld);
if exist('moon_in.mat')
    load moon_in
else
    kk=[];dt=[];m=[];
end
tsd=[];usd=[];
t=0;tt=fix(FT/DT);
[kk,dt,m]=get_moon_in(kk,dt,m); %Get information for disturbance input
for k=1:tt
    u = feval(u_file,t,kk,dt,m);

```

```

    usd=[usd;u];
    tsd=[tsd;t];
    t=t+DT;
end
var_svd_in=' usd tsd kk dt m';
sv_dat=[var_svd_in,' var_svd_in'];
sv=['save moon_in',sv_dat];eval(sv)
return
%+++++
%               That's the end of moon_in.m
%+++++

```

u_moonen.m

```

function u = u_moonen(t,k,dt,m)
%+++++
%  FILE:
%      u_moonen.m   - This code provides a SINC type input for
%                    Moonen, et al. type identification.
%
%  INPUT:
%      t - time;
%      k - time mult. factor.
%      dt - time shift factor.
%      m - a magnitude factor.
%
%  OUTPUT:
%      u - matrix of input control signals.
%
%  GLOBAL:
%      None.
%
%  CALLS:
%      None.
%
%  DATA FILES WRITTEN:
%      None.
%
%  DATA FILES READ:
%      None.
%
%  AUTHOR:
%      Steve Hietpas October 1992
%

```

```

% REFERENCES:
%     None.
%
% SYNTAX:
%     u = u_moonen(t,k,dt,m);
%+++++
%This is the SINC function.
if (t-dt)==0
    u=m;
else
    u=m*sin(k*(t-dt))/(k*(t-dt));
    u=u+(rand(1)-.5)/(100*m);
end
return
%+++++
%           That's the end of u_moonen.m
%+++++

get_moon_in.m

function [k,dt,m]=get_moon_in(k,dt,m)
%+++++
% FILE:
%     get_moon_in - Routine for getting input probing information.
%
% INPUT:
%     k           - time mult. factor.
%     dt          - time shift factor.
%     m           - a magnitude factor.
%
% OUTPUT:
%     k,dt,m      - see above.
%
% CALLS:
%     None.
%
% GLOBAL:
%     None.
%
% DATA FILES WRITTEN:
%     None.
%
% DATA FILES READ:

```

```

%      None.
%
%  AUTHORS:
%      Steve Hietpas January 1994
%
%  REFERENCES:
%      None.
%
%  SYNTAX:
%      [k,dt,m]=get_moon_in(k,dt,m);
%+++++
if exist('k')
    disp('The previous value for the time mult. factor, k, was:')
    disp(k)
    disp('The previous value for time shift factor, dt, was:')
    disp(dt)
    disp('The previous value for amplitude factor, m, was:')
    disp(m)
    disp(' ')
    ans1=input('Do you want to keep these? [y,Y,<rtn>=yes]: ','s')
    if ~(isempty(ans1) | ans1=='y' | ans1=='Y')
        disp('The previous value for k was:')
        disp(k)
        disp('What values for k_new [...] (<rtn> for old values)? ')
        k_new=input('k_new: ');
        if isempty(k_new)
            k_new=k;
        end
        disp('The previous value for m was:')
        disp(m)
        disp('What values for m_new [...] (<rtn> for old values) ')
        m_new=input('m_new: ');
        if isempty(m_new)
            m_new=m;
        end
        disp('The previous values for dt was:')
        disp(dt)
        disp('What values for dt_new [...] (<rtn> for old values) ')
        dt_new=input('dt_new: ');
        if isempty(dt_new)
            dt_new=dt;
        end
        k=k_new;
    end
end

```

```

    dt=dt_new;
    m=m_new;
end
else
    disp('What values for k [...]? ')
    k=input('k: ');
    disp(' ')
    disp('What values for dt [...]? ')
    dt=input('dt: ');
    disp(' ')
    disp('What values for m [...]? ')
    m=input('m: ');
    disp(' ')
end
disp('The final choices are:')
disp(' ')
disp('k: ')
disp(k)
disp(' ')
disp('dt: ')
disp(dt)
disp(' ')
disp('m: ')
disp(m)
disp(' ')
return
%+++++
%                               That's the end of get_moon_in.m
%+++++

```

u_modes.m

```

function u = u_modes(t,k,m,dt)
%+++++
%  FILE:
%      u_modes.m      - This codes provides sinusoidal inputs
%                      for any identification routine.
%
%  INPUT:
%      t - time;
%      k - Vector of modes in rad/sec.
%      m - The amplitude vector for each mode.
%      dt - How long you want the drive signal on.

```

```

% OUTPUT:
%           u - input control signals to system.
%
% GLOBAL:
%           None.
%
% CALLS:
%           None.
%
% DATA FILES WRITTEN:
%           None.
%
% DATA FILES READ:
%           None.
%
% AUTHOR:
%           Steve Hietpas October 1992
%
% REFERENCES:
%           None.
%
% SYNTAX:
%           u = u_modes(t,k,m,dt);
%+++++
u=0;
if t<=dt
    lk=length(k);
    for jj=1:lk
        u=u+m(jj)*sin(k(jj)*t);
    end
end
return
%+++++
%           That's the end of u_modes.m
%+++++

sysm6.m
%+++++
% FILE:
%           sysm6.m - This is an example of a 4th order mechanical
%                   system with a second order low-pass/wash-out
%                   filter.

```

```

%                               We want to control velocity of M1 := v1.
%
%
%                               ++++++++          k3          | /
%                               +   +-----/\ /\ /\ /\-----| /
%                               uin +   +                                     | /
%                               --> + M1 + k1 ++++++++ k2          | /
%                               +   +---/\ /\ /\---+ M2 +---/\ /\ /\---| /
%                               +   +       din +   +           | /
%                               ++++++++          --> ++++++++    | /
%                               #####b1          #####b2
%                               =====
%                               | p1          | p2
%                               -->          -->
%
% ##### indicates friction on the mass with
% b1 and b2 the damping coefficients.
% Let x1=p1, d/dt(x1)=x2, x3=p2, d/dt(x3)=x4
% thus the velocity of M1 is x2 and of M2 is x4.
% The output v1=x2 is passed through a
% low-pass/wash-out filter given by:
%
%   - - - - -
%   | x1f | | -31.4 -31.4 | | x1f | | 31.4 |
%   | .   | =|           | |   | + |       | x2
%   | x2f | | 0    -0.25 | | x2f | | 0.25 |
%   - - - - -
%
% y = x1f = filtered value of x2 := v1f.
% The addition of this filter can be seen below
% in the construction of A and of C, producing
% a 6th order system. Thus the states are:
%
%   - - - - -
%   | x1 | | p1 |
%   | x2 | | v1 |
%   | x3 | =| p2 |
%   | x4 | | v2 |
%   | x1f | | v1f |
%   | x2f | | x2f |
%   - - - - -
%
% INPUT:
%       None.
%
% OUTPUT:

```

```

%      None.
%
% GLOBAL:
%      None.
%
% CALLS:
%      None.
%
% DATA FILES WRITTEN:
%      None.
%
% DATA FILES READ:
%      None.
%
% AUTHOR:
%      Steve Hietpas October 1992
%
% REFERENCES:
%      None.
%
% SYNTAX:
%      sysm6;
%+++++
din=1;
%
%The A Matrix
%
k1=10;%spring constant
k2=10;%spring constant
k3=15;%spring constant
b1=.1;%damping coef. for M1
b2=.1;%damping coef. for M2
M1=1; %Mass 1
M2=1; %Mass 2
a11=0;          a12=1;          a13=0;          a14=0;          a15=0;
                                     a16=0;
a21=-(k1+k3)/M1;a22=-b1/M1;a23=k1/M1; a24=0;          a25=0;
                                     a26=0;
a31=0;          a32=0;          a33=0;          a34=1;          a35=0;
                                     a36=0;
a41=k1/M2;      a42=0;          a43=-(k1+k2)/M2;a44=-b2/M2; a45=0;
                                     a46=0;
a51=0;          a52=31.4; a53=0;          a54=0;          a55=-31.4;

```

```

a61=0;          a62=.25;  a63=0;    a64=0;          a56=-31.4;
a65=0;          a66=-.25;

%The entire system with filter.
A =[a11 a12 a13 a14 a15 a16
    a21 a22 a23 a24 a25 a26
    a31 a32 a33 a34 a35 a36
    a41 a42 a43 a44 a45 a46
    a51 a52 a53 a54 a55 a56
    a61 a62 a63 a64 a65 a66];

%
%The input matrix for the entire system and filter.
%
B=[0 0
   1 0
   0 0
   0 1
   0 0
   0 0];

%The output matrix for the entire system with filter.
C=[0 0 0 0 1 0];

%
%The D matrix
D=[0 0];

%
if din==0
    B=B(:,1);
    D=D(:,1);
end

%+++++
%               That's the end of sysm6.m
%+++++

```

discr.m

```

function [F,G]=discr(A,B,T)
%+++++
% File:
%       discr      - Compute z-plane state matrices from
%                   continuous state eqn's:
%                   x_dot=Ax+Bu
%                   y=Cx+Du
%                   Discrete form from ZOH with sample period T:

```

```

%       $x(k+1)=Fx(k)+Gu(k)$ 
%       $y(k)=Cx(k)+Du(k)$ .
%      Note matrix C and D remains the same.
%
%
% INPUT:
%      A      - The state matrix.
%      B      - The input matrix.
%      T      - The sampling time.
%
% OUTPUT:
%      F      - The discretized state matrix.
%      H      - The discretized input matrix.
%
% GLOBAL:
%      None.
%
% CALLS:
%      None.
%
% DATA FILES WRITTEN:
%      None.
%
% DATA FILES READ:
%      None.
%
% AUTHOR:
%      Tia Sharpe, May 1992
%
% REFERENCES:
%      Franklin, Powel, and Workman,
%      "Digital Control of Dynamic Systems,"
%      pg. 53.
%
% SYNTAX:
%      [F,G]=discr(A,B,T)
%      ++++++
s=size(A);
I=eye(s(1));
psi=I;
%To increase accuracy increase the variable 'acc'.
acc=20;
for m=acc:-1:2

```

```

    psi=I+(A*T)/m*psi;
end
G=psi*T*B;
F=I+A*T*psi;
return
%+++++
%                               That's the end of discr.m
%+++++

```

fftst.m

```

function [sigf,f,energy]=fftst(sigt,t,plt,N,data_file)
%+++++
%  FILE:
%      fftst.m - This routine performs a Fast Fourier Transform
%                  on the vector of data, sigt.
%
%  INPUT:
%      sigt      - The signal vector of data over time
%                  (real or complex).
%      t         - Either a vector of time or the step
%                  (sampling) time.
%      plt       - Plotting on for plt=1, else no plotting.
%      N         - The size of the FFT to be performed. (optional)
%                  If N is not given, you will be asked if you want
%                  radix-2 fft to be performed.
%                  If, however, N='Y', then a radix-2 fft will be
%                  performed.
%                  If, however, N=0, then N will be set to length
%                  of sigt.
%      data_file - The name of the data file from which the data
%                  originates.
%
%  OUTPUT:
%      sigf      - The resulting FFT of sigt over frequency (complex).
%      f         - The vector of frequency (real).
%
%  GLOBAL:
%      None.
%
%  CALLS:
%      None.
%
%  DATA FILES WRITTEN:

```

```

%      None.
%
% DATA FILES READ:
%      None.
%
% AUTHOR:
%      Steve Hietpas November 1993
%
% REFERENCES:
%      None.
%
% SYNTAX:
%      [sigf,f,energy]=fftst(sigt,t,plt,N,data_file);
%+++++
if ~nargin>=2
    disp('Must enter at least 2 variables')
    return
end
if nargin==2
    plt=0;
end
tlen=length(t);sigtlen=length(sigt);
if tlen~=sigtlen
    T=t;
    t=0:T:(sigtlen-1)*T;
else
    T=t(2)-t(1);
end
if nargin<4
    strg='Do you want radix-2 fft to be performed? ';
    strg=[strg,'[y,Y,<ret>=yes] >> '];
    rad2=input(strg,'s');
    if (isempty(rad2) | rad2=='y' | rad2=='Y')
        N=2^(ceil(log10(sigtlen)/log10(2)));%This step insures radix 2 FFT.
    else
        N=sigtlen;
    end
else
    if N==0
        N=sigtlen;
    elseif N=='Y'
        N=2^(ceil(log10(sigtlen)/log10(2)));%This step insures radix 2 FFT.
    end
end

```

```

end
sigf=fft(sigt,N);
onebyNT=1/(N*T);
Nby2=N/2;
f=onebyNT*[0:Nby2-1];
Smag=(abs(sigf)/N)';
Sby2=Smag(1:Nby2);
energy=0;
for k=1:Nby2
    energy=energy + Sby2(k)^2;
end
energy=sqrt(energy)
if plt
    figure
    tt1='Signal vs Time';yy1='Signal';xx1='Time (seconds)';
    tt='FFT of data';yy='Magnitude';xx='Frequency (Hz)';
    axis([min(t) max(t) min(sigt) max(sigt)]);
    subplot(211),plot(t,sigt,'r'),title(tt1),xlabel(xx1),ylabel(yy1),
    axis;subplot(212),plot(f(1:Nby2),Smag(1:Nby2),'g'),
    title(tt),xlabel(xx),ylabel(yy)
    time_stamp,dir_stamp,
    if nargin==5
        file_stamp(data_file);
    end
    subplot(111)
end
return
%+++++
%
%               That's the end of fftst.m
%+++++

```

dir_stamp.m

```

%+++++
% FILE:
%     dir_stamp.m - Provides the ability to directory stamp your
%                   plots.
%
% INPUT:
%     None.
%
% OUTPUT:
%     None.

```

```

%
% GLOBAL:
%     None.
%
% CALLS:
%     None.
%
% DATA FILES WRITTEN:
%     None.
%
% DATA FILES READ:
%     None.
%
% AUTHOR:
%     Steve Hietpas January 1994
%
% REFERENCES:
%     None.
%
% SYNTAX:
%     dir_stamp;
%
%+++++
curr_dir=cd;
disp('Place current directory listing on plot using mouse')
gtext(curr_dir)
return
%+++++
%           That's the end of dir_stamp.m
%+++++

```

file_stamp.m

```

function []=file_name(df1,df2,df3,df4)
%+++++
% FILE:
%     file_stamp.m - Provides the ability to file stamp your
%                   plots.
%
% INPUT:
%     df#           - Four possible file names can be entered.
%
% OUTPUT:
%     None.

```

```

%
% GLOBAL:
%     None.
%
% CALLS:
%     None.
%
% DATA FILES WRITTEN:
%     None.
%
% DATA FILES READ:
%     None.
%
% AUTHOR:
%     Steve Hietpas January 1994
%
% REFERENCES:
%     None.
%
% SYNTAX:
%     file_name(df1,df2,df3,df4);
%+++++
data_files=[];
no_files=nargin;
if no_files==1 %Could be a matrix of string data.
    [no_files,c]=size(df1);
    for k=1:no_files
        k1=findstr(' ',df1(k,:));
        if ~isempty(k1)
            fl_nm=df1(k,1:min(k1)-1);
        else
            fl_nm=df1(k,:);
        end
        data_files=[data_files,fl_nm,'.mat, '];
    end
else
    for k=1:no_files
        fl_nm=eval(['df',int2str(k)]);
        data_files=[data_files,fl_nm,'.mat, '];
    end
end
data_files=data_files(1:length(data_files)-2);
disp('Place file name on plot using mouse.')

```

```
d_f=['Data from file(s): ',data_files];
gtext(d_f)
return
```

```
%+++++
%               That's the end of file_stamp.m
%+++++
```

time.m

```
function [tim,dat]=time
```

```
%+++++
```

```
% FILE:
```

```
%   time.m      - Retrieves the time and date and converts it to
%                  something readable.
```

```
%
```

```
% INPUT:
```

```
%   None.
```

```
%
```

```
% OUTPUT:
```

```
%   tim      - The time.
```

```
%   dat      - The date.
```

```
%
```

```
% GLOBAL:
```

```
%   None.
```

```
%
```

```
% CALLS:
```

```
%   None.
```

```
%
```

```
% DATA FILES WRITTEN:
```

```
%   None.
```

```
%
```

```
% DATA FILES READ:
```

```
%   None.
```

```
%
```

```
% AUTHOR:
```

```
%   Steve Hietpas January 1994
```

```
%
```

```
% REFERENCES:
```

```
%   None.
```

```
%
```

```
% SYNTAX:
```

```
%   time;
```

```
%+++++
```

```

fix(clock);
cc=clock;
hr=cc(4);mn=cc(5);
if mn<10
    mn=['0',num2str(mn)];
else
    mn=num2str(mn);
end
if hr>=12
    part=' p.m.';
    if hr>12
        hr=hr-12;
    end
else
    part=' a.m.';
end
tim=[num2str(hr),':',mn,part];
dat=date;
return
%+++++
%               That's the end of time.m
%+++++

```

Chapter 2, Example 4: Equations and Code

Following is a detailed description of the necessary code for producing the simulation results of example 4 in chapter 2. Figure 14 is the classical model diagram for the one-machine infinite-bus power system. The data for machine M_1 and the transmission line is as follows:

Although there exist programs that compute the steady-state load flow of power systems, the simplicity of this model allows for manual calculation and is therefore presented below. As stated in Example 4 of Chapter 2, the value of the bus and transmission line voltages and currents are given by

- $\vec{V}_t = 1 \angle \beta$
- $\vec{I}_t = I_t \angle \phi$
- $\vec{V}_\infty = 1 \angle 0^\circ$
- $\vec{I}_\infty = -\vec{I}_t$

At steady-state, the electrical power output of the machine, P_e , is approximately equal to the mechanical input power,

$$P_m \approx P_e = \frac{|\vec{V}_t| |\vec{V}_\infty| \sin \beta}{|\vec{x}_t|} \quad (\text{A.1})$$

	Description	Symbol	Value	Units
M	d-axis synchronous reactance	$\vec{x}_d$	j1.0	p.u.
A	d-axis transient reactance	$\vec{x}'_d$	j0.18	p.u.
C	d-axis transient time-constant	T'_{do}	3.0	seconds
H	q-axis synchronous reactance	$\vec{x}_q$	j0.6	p.u.
I	q-axis transient reactance	$\vec{x}'_q$	j0.15	p.u.
N	q-axis transient time-constant	T'_{qo}	1.6	seconds
E	Inertia constant	H	7	MWatt-seconds/MVA
	Mechanical power input	P_m	1.5	p.u.
	Transmission line reactance	$\vec{x}_t$	j0.2	p.u.

Table 8. Machine data for Example 4 of Chapter 2.

From (A.1) we compute $\beta = 17.458^\circ$. The injected current at the machine bus is given by

$$\vec{I}_t = I_t \angle \phi = \frac{\vec{V}_t - \vec{V}_\infty}{\vec{x}_t},$$

and using values from Table 8, we find that $I_t = 1.518$ and $\phi = 8.729^\circ$. Next, we compute the steady-state value of the machine shaft angle denoted by δ (this angle is equivalent to the angle between the quadrature-axis and the infinite bus voltage). Refer to Figure 25, which describes the relationship between the bus voltages and the machine's internal quadrature-axis voltages.

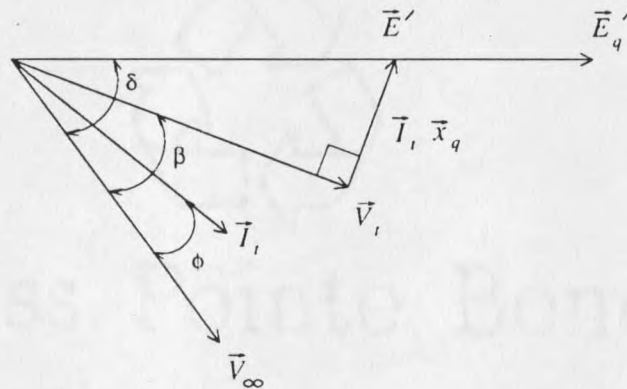


Figure 25. Steady-state vector diagram of bus voltages.

With reference to the infinite bus voltage, $\vec{V}_\infty$, we obtain the following equations:

$$\vec{E}' = E' \angle \delta = \vec{V}_t + \vec{I}_t \vec{x}_q$$

Hence, $E' = 1.451$ and $\delta = 55.792^\circ$. As an algebraic check the following equations may be used:

- $|\vec{I}_d| = I_d = |\vec{I}_t| \sin(\delta - \phi) = 1.1110$
- $|\vec{I}_q| = I_q = |\vec{I}_t| \cos(\delta - \phi) = 1.0338$
- $|\vec{V}_d| = V_d = |\vec{V}_t| \sin(\delta - \beta) = 0.6202$
- $|\vec{V}_q| = V_q = |\vec{V}_t| \cos(\delta - \beta) = 0.7843$
- $Pe = |\vec{V}_d||\vec{I}_d| + |\vec{V}_q||\vec{I}_q| = V_d I_d + V_q I_q = 1.5$
- $|\vec{E}'_q| = E'_q = |\vec{V}_q| + |\vec{x}'_d||\vec{I}_d| = 0.9843$

The state-equations of the machine may be written now that the steady-state values of the system have been determined. Let the states of the system be defined as follows:

- $x_1 \triangleq \delta$: the quadrature-axis voltage angle which is equivalent to the shaft angle
- $x_2 \triangleq \dot{x}_1 \triangleq \omega$: the shaft speed deviation
- $x_3 \triangleq |\vec{E}'_q| = E'_q$: the quadrature-axis synchronous voltage

The final set of state-equations are listed as follows:

$$\dot{x}_2 = \dot{\omega} = \frac{\pi f}{H} (P_m - P_e)$$

and

$$\dot{x}_3 = \dot{E}'_q = \frac{E_f - (x_d - x'_d)I_d - E'_q}{T'_{do}} \quad (\text{A.2})$$

where, $f = 60$, is the steady-state line frequency and E_f is the field voltage. At steady-state $\dot{E}'_q = 0$ which allows the use of (A.2) to compute the initial (steady-state) value of $E_{f0} \triangleq E_f = 1.8954$. The machine has an exciter whose purpose is to adjust the field voltage E_f for voltage regulation. The exciter introduces a 4th state, $x_4 \triangleq E_f$; thus the last differential equation is

$$\dot{x}_4 = \frac{k_e(V_{ref} - V_t) - x_4}{t_e}$$

where k_e is the exciter gain, t_e is the exciter time constant, and V_{ref} is the voltage reference.

With this 4th state defined as above, we see that (A.2) becomes

$$\dot{x}_3 = \frac{x_4 - (x_d - x'_d)I_d - x_3}{T'_{do}}$$

The algebraic side equations are listed below:

- $V_q = \frac{x_3 x_t + x_d' V_\infty \cos(x_1)}{x_t + x_d'}$
- $V_d = \frac{V_\infty \sin(x_1) x_q}{x_t + x_q}$
- $V_t = \sqrt{V_q^2 + V_d^2}$
- $I_q = \frac{V_d}{x_q}$
- $I_d = \frac{-V_q + x_3}{x_d'}$
- $P_e = V_q I_q + V_d I_d$
- $I_t = \frac{P_e}{V_t}$

The code used for Example 4, which includes the above equations, is found in `sys_infty.m` and `rk.m` (both are listed below).

sys_infty.m

```

%+++++
% FILE:
%   sys_infty.m - This routine simulates a one machine
%                 infinite bus power system
%                 dynamics problem using 'rk.m'.
%                 The routine rk.m performs the Runge-Kutta
%                 integration.
%                 The state derivative, xdot, and the counter,
%                 k, for controlling which stage of integration
%                 the R-K routine should be evaluating are
%                 passed between each routine.
%
% INPUT:
%   None.
%
% OUTPUT:
%   None.
%
% GLOBAL:
%   None.
%
% CALLS:
%   rk.m - a Runge-Kutta (4th order) integration routine.
%
% DATA FILES WRITTEN:
%   uyt.mat - all output data
%
% DATA FILES READ:
%   lin.mat - state-space matrix values for augmented linear
%             control (PSS unit) to exciter unit.
%
% AUTHOR(s):
%   Steve Hietpas, March 1992
%   Cannabalized by Jim Smith January 1993
%   Completed by Fereshteh Fatehi
%   Majorly modified by Steve Hietpas, June 1994
%
% REFERENCES:
%   None.
%
% SYNTAX:

```

```

%      sys_infty;
%+++++
global K Th0 Th0_int gam0 R Q x0 f0 h0 Fa n m F H
t0=0.00;      %initial starting time
deltat=0.01; %time step for performing RK integration.
oens=5;       % output every n steps ( every n deltat steps)
DT=oens*deltat; % sampling interval for digital analysis and control
t=t0; % t is the time variable
k=0; % k is the control variable for Runge-Kutta;
j=0; % j is the the control variable for displaying the time
      % of simulation and also for data output control;
xt0=.2 ; % line (transmission) impedance
xt=xt0;
dxt=.15; % perturbation on line impedance
vinf=1.0;% magnitude of infinite bus voltage
vt=1.0; % magnitude of terminal bus voltage
H=7 ; % inertia constant
pm=1.5; % mechanical input power
pe=1.5; % electrical power output at steady state
ke=10; % exciter gain constant
te=0.4; % exciter time constant
dd=55.7903*pi/180; % angle delta of V_infty (infinite bus)
      % with respect to Eq' (Eqp)
Ef0=1.8954; % initial value of field voltage
Eqp0=0.98439; % magnitude of initial value of quadrature
      % voltage of machine
xdp=0.18; % magnitude of d-axis impedance of machine
xq=0.6; % magnitude of q-axis impedance of machine
% The states of the system with exciter are:
% x(1) = angle (delta) of V_infty (infinite bus) with respect
%      to Eq' (Eqp)
% x(2) = change in angle (omega = delta dot = d/dt of delta)
% x(3) = quadrature voltage of machine (Eqp)
% x(4) = field voltage (Ef)
x=[dd;0.0;Eqp0;Ef0];
xdot=[0;0;0;0]; % initial state derivatives of machine
vref0=(Ef0/ke)+vt; % initial reference voltage value
vsc0=0.0; % initial control voltage
vsc=vsc0; % initialize control voltage
vref02=vref0;
it=1.5175; % initial magnitude of line current
nout=1500; % specifies # of steps of deltat in the simulation
tfinal=nout*deltat;

```

```

del_ref=input('Disturb vref?          [<ret>,n,N,=no] >> ','s');
del_xt =input('Disturb xt?           [<ret>,n,N,=no] >> ','s');
do_lin =input('Control with linear PSS? [<ret>,n,N,=no] >> ','s');
do_lqr =input('Control with LQR PSS?   [<ret>,n,N,=no] >> ','s');
do_mm  =input('Control with minimax LQR? [<ret>,n,N,=no] >> ','s');
do_pr  =input('Run Prony id?           [<ret>,n,N,=no] >> ','s');
do_ove =input('Run OVE id?            [<ret>,n,N,=no] >> ','s');
kk=1; % this variable is used by either Prony ID or LQR control
if ~(isempty(do_pr) | do_pr=='n' | do_pr=='N')
    % make sure no disturbances,
    % control or other identification are present
    del_ref='n';
    del_xt='n';
    do_lin='n';
    do_lqr='n';
    do_mm='n';
    do_ove='n';
    disp('All disturbances are neglected and control unit is off')
    disp(['The initial reference voltage is: ',num2str(vref0)]);
    [usd,tsd]=prony_in('prony_in','u_prony',tfinal);
    load prony_in
    vsc=usd(kk);
    kk_len=length(usd);
elseif ~(isempty(do_ove) | do_ove=='n' | do_ove=='N')
    % make sure no disturbances,
    % control or other identification are present
    del_ref='n';
    del_xt='n';
    do_lin='n';
    do_lqr='n';
    do_mm='n';
    do_pr='n';
    disp('All disturbances are neglected and control unit is off')
    n=4;
    m=n;
    N=nout; % see above
    usd=rand(N+1,1);
    usd=usd-mean(usd);
    max_usd=max(usd);
    strg='Enter maximum value for input probing signal [<ret>=1] >> ';
    usd_max=input(strg);
    if isempty(usd_max)
        usd_max=1;

```

```

end
usd=usd*usd_max/max_usd;
vsc=usd(kk);
usd0=usd;
kk_len=length(usd);
elseif ~(isempty(do_lin) | do_lin=='n' | do_lin=='N')
    % make sure no other control or identifaciton are present
    do_lqr='n';
    do_mm='n';
    do_pr='n';
    do_ove='n';
    xc=[0;0;0;0;0];    % states of the linear controller
    x=[x;xc];          % combine states of entire system
    xdotc=[0;0;0;0;0]; % state derivatives of linear controller
    xdot=[xdot;xdotc]; % combine state derivatives of entire system
    load lin
elseif ~(isempty(do_lqr) | do_lqr=='n' | do_lqr=='N')
    % make sure no other control or identifaciton are present
    do_lin='n';
    do_mm='n';
    do_pr='n';
    do_ove='n';
% load sys_infty_pr; % this file is initially created by sys_infty,
                    % then by pr1.m, pr1b.m, pr2.m, pr2b.m, and pr3.m
load sys_infty_pr_mod % this file is created by Prony ID (pr3.m)
n=length(eigFm);
m=n;
%LQR gain setup
Q=Cm'*Cm;
not_done=1;
while not_done
    strg='Enter scalar value R for LQR cost [<ret>=1] >> ';
    R=input(strg);
    if isempty(R)
        R=1;
    end
    [K,Plqr]=dlqr(Fm,Gm,Q,R);
    cls_pls=eig(Fm-K*Gm);
    disp('The closed-loop poles of the discrete system are:')
    disp(cls_pls)
    strg='Are the closed-loop poles acceptable?';
    strg=[strg,' [<ret>,y,Y=yes] >> '];
    ans=input(strg,'s');

```

```

    if (isempty(ans) | ans=='y' | ans=='Y')
        not_done=0;
    end
end
%Observer, Kalman filter setup
E=eye(n);
x_est=zeros(n,1);
P=zeros(n,n);
KKe=[];
Y_est=0;
Ke=x_est;
%w=randn(10000,1);
%w=w-mean(w);
%w=w/10; %standard deviation is approximately 0.1
%w=zeros(10000,1);
Qe=0.1^2*100;
const_e=Gm*Qe*Gm';
%v=randn(10000,1);
%v=v-mean(v);
%v=v/10; %standard deviation is approximately 0.05
Re=1/(10^2);
elseif ~(isempty(do_mm) | do_mm=='n' | do_mm=='N')
    % make sure no other control or identification are present
    do_lin='n';
    do_lqr='n';
    do_pr='n';
    do_ove='n';
    load sys_infty_ove % this file is created by sys_infty when
                      % ove id is selected.

    n=length(H0);
    m=n;
    %LQR setup
    % use the center of the ellipsoid for the nominal values.
    Fm=F0;
    Gm=H0;
    Cm=C0;
    Q=Cm'*Cm;
    not_done=1;
    while not_done
        strg='Enter scalar value R for LQR cost [<ret>=1] >> ';
        R=input(strg);
        if isempty(R)
            R=1;

```

```

end
% run the minimax algorithm to find K.
mmJd;
cls_pls=eig(F-K*H);
disp('The closed-loop poles of the discrete system are:')
disp(cls_pls)
strg='Are the closed-loop poles acceptable?';
strg=[strg,' [<ret>,y,Y=yes] >> '];
ans=input(strg,'s');
if (isempty(ans) | ans=='y' | ans=='Y')
    not_done=0;
end
end
%Observer, Kalman filter setup
E=eye(n);
x_est=zeros(n,1);
P=zeros(n,n);
KKe=[];
Y_est=0;
Ke=x_est;
%w=randn(10000,1);
%w=w-mean(w);
%w=w/10; %standard deviation is approximately 0.1
%w=zeros(10000,1);
Qe=0.1^2*100;
const_e=Gm*Qe*Gm';
%v=randn(10000,1);
%v=v-mean(v);
%v=v/10; %standard deviation is approximately 0.05
Re=1/(10^2);
end
xi=x; % initial state vector used by rk.m
xd=x; % intermediate state vector used by rk.m
% Initial setup of output variables
tout=[]; uout=[]; yout=[]; xout=[]; pout1=[]; vscc=[];
vtout=[]; itout=[];
var_svd=[];
%+++++
%                               BEGIN MAIN PROGRAM LOOP
%+++++
while t <= tfinal
    %Data output section
    if rem(j,oens) == 0.0

```

```

if k==0
    y = [x(1) x(2)]; % specify output y to be some of the states
    tout = [tout; t];
    yout = [yout; y];
    pout1=[pout1;pe];
    vscc=[vscc; vsc];
    vtout=[vtout;vt];
    itout=[itout;it];
end
end
% Prony identification section
if ~(isempty(do_pr) | do_pr=='n' | do_pr=='N')
    %probe reference input for Prony identification
    if rem(j,oens) == 0.0
        if k==0
            kk=kk+1;
            if kk<=kk_len;
                vsc=usd(kk);
            end
        end
    end
end
% OVE algorithm for finding bounding ellipsoid.
if ~(isempty(do_ove) | do_ove=='n' | do_ove=='N')
    if rem(j,oens) == 0.0
        if k==0
            kk=kk+1;
            if kk<=kk_len;
                vsc=usd(kk);
            end
            if (t>1 & t<=(1+DT))
                x0=x;
            end
        end
    end
end
% disturbance section.
if ~(isempty(del_ref) | del_ref=='n' | del_ref=='N')
    %Change reference input
    if t>0.0
        vref02=1.05*vref0;
    end
else

```

```

    vref02=vref0;
end
if ~(isempty(del_xt) | del_xt=='n' | del_xt=='N')
    %Change line impedance
    if (t>1 & t<2)
        xt=xt0+dxt;
    else
        xt=xt0;
    end
end
end
% Algebraic side equations.
vq=(x(3)*xt+xdp*vinf*cos(x(1)))/(xt+xdp);
vd=vinf*sin(x(1))*xq/(xt+xq);
vt=sqrt(vq^2+vd^2);
iq=vd/xq;
id=(-vq+x(3))/xdp;
pe=vq*iq+vd*id;
it=abs(pe/vt);
vref=vref02+vsc;
%Runge-Kutta control section and state derivative update.
% xdot(1:4) are for the main system
xdot(1)=x(2);
xdot(2)=(pi*60/H)*(pm-pe);
xdot(3)=(x(4)-.82*id-x(3))/3;
xdot(4)=(ke*(vref-vt)/te)-(x(4)/te);
% xdot(5:9) are for the linear augmenting PSS control
if ~(isempty(do_lin) | do_lin=='n' | do_lin=='N')
    xdot(5:9)=a_lin*x(5:9)+b_lin*x(2);
    vsc=c_lin*x(5:9)+d_lin*x(2);
end
if k==0
    ti=t;
    xi=x;
end
% xd are the intermediate state values from the
% Runge-Kutta routine.
[x,t,xd]=feval('rk',ti,deltat,xi,xdot,xd,k);
k=k+1;
if k==4
    j=j+1;
    k=0;
end
end
if ~(isempty(do_lqr) | do_lqr=='n' | do_lqr=='N')

```

```

if rem(j,oens) == 0.0
    if k==0
        kk=kk+1;
        if kk>(n+1) %wait until the estimates of the states
            %is approximately stable.
            vsc=-K*x_est; % K is the lqr gain as designed above.
        else
            vsc=0;
        end
        %Observer equations.
        x_estb=Fm*x_est+Gm*(vsc);
        y_est=Cm*x_estb;
        x_est=x_estb+Ke*(x(2)-y_est);
        Pb=Fm*P*Fm'+const_e;
        Ke=Pb*Cm'*inv(Cm*Pb*Cm'+Re);
        KKe=[KKe Ke];
        Y_est=[Y_est;y_est];
        P=(E-Ke*Cm)*Pb;
    end
end
if ~(isempty(do_mm) | do_mm=='n' | do_mm=='N')
    if rem(j,oens) == 0.0
        if k==0
            kk=kk+1;
            if kk>(n+1) %wait until the estimates of the states
                %is approximately stable.
                vsc=-K*x_est; % K is the lqr gain as designed above.
            else
                vsc=0;
            end
            %Observer equations.
            x_estb=Fm*x_est+Gm*(vsc);
            y_est=Cm*x_estb;
            x_est=x_estb+Ke*(x(2)-y_est);
            Pb=Fm*P*Fm'+const_e;
            Ke=Pb*Cm'*inv(Cm*Pb*Cm'+Re);
            KKe=[KKe Ke];
            Y_est=[Y_est;y_est];
            P=(E-Ke*Cm)*Pb;
        end
    end
end
end

```

```

end
%+++++
% This section saves variables, etc.
%+++++
tsd=tout;
ysd2=yout(:,2); %pr1.m needs variable named ysd2 to run properly.
                %Here ysd2 is taken as speed deviation.

usd=vsc;
% Ellipsoid variables
F0=[];
H0=[];
gam0=[];
if ~(isempty(do_ove) | do_ove=='n' | do_ove=='N')
    %Determine bounding ellipsoid:
    nb=input('Enter the noise bound: >> ');
    [gam0_inv,Th0_int]=ove(0,usd,ysd2,nb);
    %Th0, the center of the ellipsoid
    %is returned as a global variable.
    gam0=inv(gam0_inv);
    Fa=[eye(n-1);zeros(1,n-1)];%For observable form.
    F0=[-Th0(1:n) Fa];H0=Th0(n+1:n+m);C0=[1 zeros(1,n-1)];D0=0;
end
ee=[];FT=tfinal;
if ~(isempty(do_pr) | do_pr=='n' | do_pr=='N')
    data_file='sys_infty_pr';
    var_svd=' DT FT cc dd ss mpi ee tsd ysd2 usd yout';
elseif ~(isempty(do_ove) | do_ove=='n' | do_ove=='N')
    data_file='sys_infty_ove';
    var_svd=' DT FT tsd ysd2 usd usd0 yout F0 H0 C0 D0';
    var_svd=[var_svd,' gam0 Th0 Th0_int x0'];
else
    var_svd=' DT FT tsd ysd2 usd usd0 yout K Ke Q R Qe Re';
    data_file='sys_infty';
end
sv_dat=[var_svd,' var_svd'];
sv=['save ',data_file,sv_dat];
eval(sv)
return
%+++++
%                That is the end of sys_infty.m
%+++++

```

rk.m

```

function [x_new,t_new,xd] = rk(ti,dt,xi,xdot,xd,k)
%+++++
% FILE:
%      rk.m      - This routine is called by rkode.m found
%                  in the same directory. This routine
%                  integrates a system of ordinary differential
%                  equations using a 4th-order Runge-Kutta
%                  formula.
%
% INPUT:
%      ti      - initial starting time for Runge-Kutta;
%      dt      - delta step time;
%      xi      - initial state values;
%      xdot    - vector of state derivatives;
%      xd      - vector of Runge-Kutta state derivatives.
%      k      - index for Runge-Kutta control.
%
% OUTPUT:
%      x_new   - vector of new state values;
%      t_new   - new time value.
%      xd      - vector of Runge-Kutta state derivatives.
%
% GLOBAL:
%      None.
%
% CALLS:
%      None.
%
% DATA FILES WRITTEN:
%      None.
%
% DATA FILES READ:
%      None.
%
% AUTHOR(s):
%      Steve Hietpas, January 1991
%
% REFERENCES:
%      None.
%
% SYNTAX:

```

```

%      [x_new,t_new,xd] = rk(ti,dt,xi,xdot,xd,k);
%+++++
if k==0
    xd(:,1) = xdot(:);
    x_new = xi+0.5*dt*xd(:,1);
    t_new = ti+0.5*dt;
    return
end
if k==1
    xd(:,2) = xdot(:);
    x_new = xi+0.5*dt*xd(:,2);
    t_new = ti+0.5*dt;
    return
end
if k==2
    xd(:,3) = xdot(:);
    x_new = xi+dt*xd(:,3);
    t_new = ti+dt;
    return
end
if k==3
    xd(:,4) = xdot(:);
    x_new = xi + (dt/6.0)*(xd(:,1)+2*xd(:,2)+2*xd(:,3)+xd(:,4));
    t_new = ti+dt;
    return
end
%+++++
%      That is the end of rk.m
%+++++

```

APPENDIX B
Robust Control Code

The following code is for all results presented in Chapter 4. The appendix is made up of primarily two sets of code. The first set of code is for all continuous-time minimax solutions, while the second set is for the discrete-time set-membership control example.

The code for the continuous-time minmax problems are comprised of mmJ.m, minJ1.m, maxJ1.m, gradJ1.m, minJ2.m, maxJ2.m, gradJ2.m; and ricc.m. The main code is mmJ.m which prompts the user to either use the code for **Algorithm 1** or for **Algorithm 2**. After this selection is made, mmJ.m will use the appropriate minimization and maximization code.

The second set of code is comprised of servo.m, maxJd.m, mmJd.m, and ove.m which are for the discrete-time set-membership control example at the end of Chapter 4. The main code for this example is servo.m which makes calls to maxJd.m, mmJd.m, and ove.m.

mmJ.m

```
function [x,k1]=mmJ(x0,a10,b10,c1,r1,gam,k1,tol,max_iter,...
                    max_gain,in_a_row)
%+++++
%  FILE:
%      mmJ.m - This program solves a minmax problem where the
%               parameters of the characteristic polynomial
%               vary within an ellipsoidal domain, and the
%               domain is bounded. This program is written
%               for an 1st order plant.
%               Simple modifications can be made to extend
%               this code to nth order single-input single-output
%               systems.
%  INPUT:
%      x0      - Initial value of the system state.
%      a10     - The nominal value of the state matrix
%                (denominator) coefficient.
%      b10     - The nominal value of the input matrix (numerator)
%                coefficient.
%      c1      - Constant value for the output matrix.
%      r1      - The cost function input control weighting matrix,
%                real-valued, symmetric and positive definite;
%                Since we are only dealing with SISO system, r
%                is a scalar;
%      gam     - The ellipsoidal bound matrix for parameters of A,
%                real-valued, symmetric and positive definite;
%      k1      - Initial value of gain;
%      tol     - The tolerance for the iterative scheme; the check
%                for the norm-value of theta(k)-theta(k-1), where
```

```

%          theta is the vector containing the parameters of
%          the system;
%  max_iter - Maximum iteration count.
%  max_gain - Maximum gain allowed for k1.
%  in_a_row - Convergence criteria must be met this many
%            times in a row.
%
%  OUTPUT:
%      x      - Final (optimal) value of numerator and denominator
%              coefficient.
%      k1     - Optimal gain vector at termination of algorithm;
%
%  CALLS:
%  constr.m - Constrained optimization program from the
%              optimization toolbox.
%  minJ1.m  - The functions to be minimized for alg. 1 and 2.
%  maxJ1.m  - The functions to be maximized for alg. 1.
%  gradJ1.m - The gradient functions used by constr.m for
%              algorithm 1.
%  minJ2.m  - The functions to be minimized for algorithm 2.
%  gradJ2.m - The gradient functions used by constr.m for
%              algorithm 2.
%
%  GLOBAL:
%      None.
%
%  DATA FILES WRITTEN:
%      None.
%
%  DATA FILES READ:
%      None.
%
%  AUTHORS:
%      Steve Hietpas  March 1994
%
%  REFERENCES:
%      None.
%
%  SYNTAX:
%      [x,k1]=mm1(x0,a10,b10,c1,r1,gam,k1,tol,max_iter,...
%              max_gain,in_a_row);
%
%  ++++++
%  global x0 a10 b10 c1 r1 gam k1
%  alg=menu('Which algorithm','Alg. 1 (Riccati)','Alg. 2 (Lyapunov)');

```

```

if alg==1
    Jmin='minJ1';
    Jmax='maxJ1';
    Jgrad='gradJ1';
else
    Jmin='minJ1';
    Jmax='maxJ2';
    Jgrad='gradJ2';
end
if nargin==0
    x0=1;
    % a10=2;
    % b10=.2;
    a10=0.1;
    b10=2;
    % a10=0;
    % b10=.2;
    c1=1;
    r1=1;
    gam=eye(2);
    k1=0;
    tol=.0001;
    max_gain=10000
    max_iter=1000;
    in_a_row=2;
end
kinarow=int2str(in_a_row);
notdone=1;
%options(1)= % Display Control, [def=0], 1,-1
%options(2)= % Terminate for x, [def=1e-4]
%options(3)= % Terminate for f, [def=1e-4]
%options(4)= % Terminate for g, [def=1e-7]
%options(5)= % Main algortihm, [def=0]
%options(6)= % SD algorithm, [def=0]
%options(7)= % Search algorithm, [def=0]
%options(8)= % Function, value at the last iteration -
%options(9)= % Gradient check, [def=0]
%options(10)= % Function count, -
%options(11)= % Gradient count, -
%options(12)= % Constraint Count, -
options(13)=1; % Equality Constraints, # of, [def=0]
%options(14)= % Max iterations,
               % [def=100n] where n is number of parameters

```

```

%options(15)= % Objective used, [def=0]
%options(16)= % Min perturbations, [def=1e-8]
%options(17)= % Max perturbations, [def=0.1]
%options(18)= % Step-size, -
%options=[];%leave empty if you want to take default values.
vub=[a10+1,b10+1,max_gain];
vlb=[a10-1,b10-1,eps];
x1=rand(1);
x2=randn(1);
x=[x1+a10;x2+b10;0];
[k1,p1]=feval(Jmin,x);
disp('-----')
disp(' ----- At initialization -----')
th=[x(1)-a10;x(2)-b10];
ellip_val=th'*gam*th;
if ellip_val<1
    fprintf('Interior:          measure= %e\n',ellip_val)
elseif ellip_val==1
    fprintf('On Boundary:      measure= %e\n',ellip_val)
else
    fprintf('Outside Boundary: measure= %e\n',ellip_val)
end
disp('-----')
disp(' ')
disp('-----')
x_new=constr(Jmax,x,options);
x=x_new;
kk=0;
k=1;
k_st=0;
flag=0;
while notdone
    k=k+1;
    [k1,p1]=feval(Jmin,x);
    x_new=constr(Jmax,x,options,vlb,vub,Jgrad);
% x_new=constr(Jmax,x,options,vlb,vub);
% x_new=constr(Jmax,x,options);
    if rem(k,10)==0
        fprintf('          Iteration %i\n',k)
        th=[x_new(1)-a10;x_new(2)-b10];
        ellip_val=th'*gam*th;
        if ellip_val<1
            fprintf('Interior:          measure= %e\n',ellip_val)

```

```

elseif ellip_val==1
    fprintf('On Boundary:      measure= %e\n',ellip_val)
else
    fprintf('Outside Boundary: measure= %e\n',ellip_val)
end
end
err=norm(x_new(1:2)-x(1:2));
x=x_new;
if err<tol
    fprintf('Error Below Tolerance: error= %e\n\n',err)
    if flag==1
        if k-k_last==1
            k_last=k;
            kk=kk+1;
            if kk==in_a_row
                notdone=0;
                strg=['Error criterion met at least '];
                strg=[strg,kinarow,' times in a row.'];
                disp(strg)
            end
        else
            flag=0;
            kk=1;
        end
    else
        flag=1;
        k_st=k;
        k_last=k_st;
    end
elseif err<10*tol
    fprintf('Error within 10*Tolerance: error= %e\n\n',err)
end
if (k==max_iter)
    fprintf('\n\n Maximum iteration count exceeded!')
    notdone=0;
end
end
[k1,p1]=feval(Jmin,x);
J=x0^2*p1;
fprintf('\n\n The final number of iterations was %i \n',k)
fprintf('The final value for den. a1 is %e \n',x(1))
fprintf('The final value for num. b1 is %e \n',x(2))
fprintf('The final value for Riccati p1 is %e \n',p1)

```

[illegible]

minJ1.m

```
function [k1,p1]=minJ1(x)
%+++++
% FILE:
%      minJ1.m - This program is called by mmJ.m to solve the
%                minimization portion of the minimax algorithm.
% INPUT:
%      x      - Optimal value of numerator and denominator
%                coefficient from mmJ.m
% OUTPUT:
%      k1      - Optimal gain vector at termination of algorithm;
%      p1      - The value of the Riccati matrix.
% CALLS:
%      ricc.m   - A matrix Riccati equation solver.
% GLOBAL:
%      None.
% DATA FILES WRITTEN:
%      None.
% DATA FILES READ:
%      None.
```

```

%
%  AUTHORS:
%      Steve Hietpas   March 1994
%
%  REFERENCES:
%      None.
%
%  SYNTAX:
%      [k1,p1]=minJ1(x)
%+++++
global c1 k1 r1
a1=-x(1);
b1=x(2);
p1 = ricc(a1,b1^2/r1,c1^2,-a1');
k1 = b1*p1/r1;
%check by looking at the quadratic equation.
%p1a= (a1+sqrt(a1^2+b1^2))/b1^2
%p1b= (a1-sqrt(a1^2+b1^2))/b1^2
return
%+++++
%      That's the end of minJ1.m
%+++++

```

maxJ1.m

```

function [f,g]=maxJ1(x)
%+++++
%  FILE:
%      maxJ1.m - This program is called by mmJ.m to solve the
%                maximization portion of the minimax algorithm.
%
%  INPUT:
%      x      - Optimal value of numerator and denominator
%                coefficient from mmJ.m
%
%  OUTPUT:
%
%      f      - Value of the function being maximized.
%      g      - Value of the constraint equations.
%
%  CALLS:
%      None.
%

```

```

% GLOBAL:
%     None.
%
% DATA FILES WRITTEN:
%     None.
%
% DATA FILES READ:
%     None.
%
% AUTHORS:
%     Steve Hietpas  March 1994
%
% REFERENCES:
%     None.
%
% SYNTAX:
%     [f,g]=maxJ1(x);
%+++++
global x0 a10 b10 c1 k1 r1 gam
ga=gam(1,1);
gb=gam(1,2);
gc=gam(2,2);
x1=x(1);
x2=x(2);
x3=x(3);
zk=r1*k1^2+c1^2;
f=-x3*x0^2;% note the negative sign -- optimization toolbox
           %does min. only.
e1=x1-a10;
e2=x2-b10;
g(1)=-2*(x1+x2*k1)*x3+zk; %equality constraint
g(2)=(ga*e1+gb*e2)*e1+(gb*e1+gc*e2)*e2-1; %inequality constraint
return
%+++++
%           That's the end of maxJ1.m
%+++++

```

gradJ1.m

```

function [df,dg]=gradJ2(x)
%+++++
% FILE:
%     gradJ2.m - This program is called by mmJ.m to solve the

```

```

%               maximization portion of the minimax algorithm.
%
% INPUT:
%     x         - Optimal value of numerator and denominator
%               coefficient from mmJ.m
%
% OUTPUT:
%
%     df        - Value of the gradient of the function being
%               maximized.
%     dg        - Value of the gradient of the constraint
%               equations.
%
% CALLS:
%     None.
%
% GLOBAL:
%     None.
%
% DATA FILES WRITTEN:
%     None.
%
% DATA FILES READ:
%     None.
%
% AUTHORS:
%     Steve Hietpas  March 1994
%
% REFERENCES:
%     None.
%
% SYNTAX:
%     [df,dg]=gradJ2(x);
%+++++
global x0 a10 b10 c1 k1 r1 gam
ga=gam(1,1);
gb=gam(1,2);
gc=gam(2,2);
x1=x(1);
x2=x(2);
x3=x(3);
zk=r1*k1^2+c1^2;
df=[0;0;-zk];

```

```

e1=x1-a10;
e2=x2-b10;
dg1=[-2*x3;-2*k1*x3;-2*(x1+x2*k1)];
dg2=[2*(ga*e1+gb*e2);2*(gb*e1+gc*e2);0];
dg=[dg1,dg2];
return
%+++++
%
%               That's the end of gradJ2.m
%+++++

```

maxJ2.m

```

function [f,g]=maxJ2(x)
%+++++
%  FILE:
%      maxJ2.m - This program is called by mmJ.m to solve the
%                maximization portion of the minimax algorithm.
%
%  INPUT:
%      x        - Optimal value of numerator and denominator
%                coefficient from mmJ.m
%
%  OUTPUT:
%
%      f        - Value of the function being maximized.
%      g        - Value of the constraint equations.
%
%  CALLS:
%      None.
%
%  GLOBAL:
%      None.
%
%  DATA FILES WRITTEN:
%      None.
%
%  DATA FILES READ:
%      None.
%
%  AUTHORS:
%      Steve Hietpas  March 1994
%
%  REFERENCES:

```

MONTANA STATE UNIVERSITY LIBRARIES



3 1762 10273292 0

