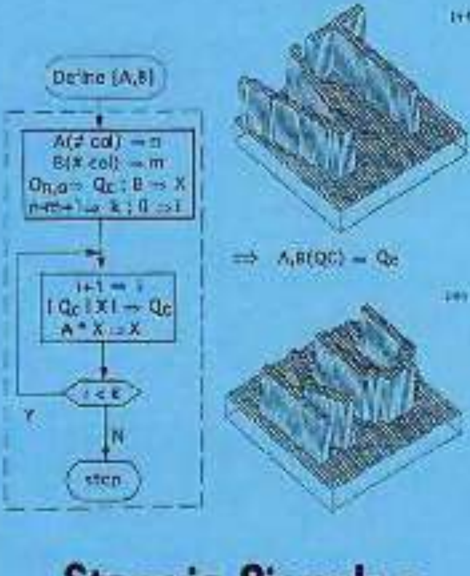
Algorithms for Computer-Aided Design of Multivariable Control Systems



Stanoje Bingulac
Hugh F. VanLandingham

Algorithms for Computer-Aided Design of Multivariable Control Systems

Algorithms for Computer-Aided Design of Multivariable Control Systems

Stanoje Bingulac

Kuwait University
Safat, Kuwait

Hugh F. VanLandingham

Virginia Polytechnic Institute and State University
Blacksburg, Virginia

Marcel Dekker, Inc. New York • Basel • Hong Kong

Library of Congress Cataloging-in-Publication Data

Bingulac, Stanoje
Algorithms for computer-aided design of multivariable control systems / Stanoje Bingulac, Hugh F. VanLandingham.
p. cm. — (Electrical engineering and electronics ; 84)
Includes bibliographical references and index.
ISBN 0-8247-8913-X
1. Control theory—Data processing. 2. Automatic control—Data processing. 3. Algebras, Linear—Data processing. 4. Computer-aided design. I. VanLandingham, Hugh F. II. Title. III. Series.
QA402.3.B525 1993 93-10191
003'.74—dc20 CIP

Comments, suggestions, or bug reports conserning the *L-A-S* software are welcome and should be sent to:

Professor S. Bingulac
Electrical and Computer Engineering
Kuwait University
P.O. Box 5969
13060 Safat, KUWAIT

or

Professor H.F. VanLandingham
Electrical Engineering Department
Virginia Polytechnic Institute and State University
Blacksburg, VA 24061 - 0111

Current printing (last digit):
10 9 8 7 6 5 4 3 2 1
PRINTED IN THE UNITED STATES OF AMERICA

To our families:

Svetlana, Slavko and Slavica

and

Patricia, Peter, Mark and Lisa

without whose support and patience
this book would not have been possible.

Preface

Practicing professionals increasingly find themselves in a position of modeling complex systems for understanding and/or control and require a more comprehensive knowledge of multivariable systems. This book focuses on the computer-aided approach as the most effective way of introducing the advanced topics of multivariable systems. Emphasis is placed on computer-aided modeling and analysis techniques to help both professionals and advanced students to extend their understanding well beyond a first course in automatic control systems. This book is also appropriate as a text for a senior or first-year graduate course in engineering. It is realistically possible to cover all essential contents of the book in one semester, and, with some selection, in one academic quarter. Appendices A and B and Chapter 1 present a summary of the essential material that is needed; this should be primarily a review for the reader.

The intent of the text is to supply only the most relevant mathematical developments, keeping proofs and detailed derivations to a minimum, while maximizing the utility of computer algorithms. These referenced and well-tested algorithms have been gathered together in a computer-aided design (CAD) package called *Linear Algebra and Systems* (*L-A-S*). *L-A-S* is an interactive conversational software language that is supplied with this text. It is used extensively in the illustrative examples throughout the book, but the utility of *L-A-S* goes well beyond the scope of this text. The reader will find *L-A-S* to be a handy and easy-to-use tool for verifying an analysis technique or control design. It is assumed that the reader has access to a personal computer to work with *L-A-S*. The hardware recomendations are an IBM PC, AT, PS-2 or compatible with a minimum of 640k of memory, MS-DOS version 3.0 or higher, math co-processor and hard disk, CGA or higher graphics, dot-matrix or laser printer.

The motivation for this text is the underlying conviction that control engineers are not well prepared for significant design work at the completion of a basic undergraduate course. Computer technology has, on the other hand, brought a great deal of computing power to the desk of individual engineers and applied scientists. We believe that this text can provide a suitable bridge for students or professionals to learn complex modeling and analysis methods. To enhance the speed of learning, the main chapters provide a special section of application problems along with their solutions.

The book may be considered to emphasize three important areas:

(1) the theory of multivariable linear systems,
(2) the development of algorithms from the theory, and
(3) the *L-A-S* software to implement the algorithms.

This book is unique in the balanced presentation of these three areas. Other texts, e.g. those by Kailath, Brogan, and Chen, dealing with the same topics, offer only the first part. Texts which do offer areas (1) and (3), such as those by Jamshidi and others which combine *MATLAB*™ with control theory, generally do so at a beginning level and do not contain significant multivariable system discussion. Although one may extract "algorithms" from theoretical developments, it is, nevertheless, time-consuming and tedious work requiring good programming skills.

The subject matter is captured in the five chapter titles:

1. Introduction
2. System Discretization
3. System Modeling
4. Intermodel Conversion
5. System Identification

In Chapter 1 various basic concepts are presented in a review mode to bridge the gap between a first course in control systems and the multivariable system material. The topics of Chapter 2 concentrate on the conversion of system representations between the discrete-time (D-T) and the continuous-time (C-T) domains, including several conversion methods based on different assumptions regarding the sampling process. In addition to *discretization* procedures, Chapter 2 also offers robust algorithms for the inverse problem of *continualization*, which converts a D-T model into an "equivalent" C-T model. The understanding of multi-input, multi-output (MIMO) system structure is the subject of Chapter 3. In addition to the standard "canonical" forms, special emphasis is given to the use of *pseudo-controllable* (and *-observable*) *forms*, which generalize the standard forms and provide greater flexibility in achieving higher numerical accuracy in the modeling process. Also included in Chapter 3 is a detailed discussion of *matrix fraction descriptions (MFDs)*. MFDs represent an important alternative to the more standard state space and transfer function matrix models. Having presented the various system modeling concepts in the earlier chapters, Chapter 4 then provides a multitude of useful algorithms which can be used to convert any one form into any other. Finally, in Chapter 5 the "conversion" from input/output data to some specific system model, i.e. *identification*, is presented. The special identification techniques are based on the flexible structural considerations of Chapter 3.

In summary, this text presents a unified theory of linear MIMO system models, containing material that is unavailable outside of the "technical journal" literature. At this time there is no other published book which provides the depth and scope, as well as a professional level software package, on the topic of MIMO systems. Perhaps, more importantly, the material is presented in a fashion to be of immediate use to the reader due to its "algorithmic" approach.

The typical format for presenting material is to provide a brief introduction and discussion of the concepts followed by one or more algorithms for performing the required operations. The algorithms themselves are also implemented in *L-A-S* code and used in a few explanatory examples. Detailed code listings are included in Appendix C. The algorithms in this book are represented in a *pseudo-code* format as a neutral way of defining the algorithms. With this pseudo-code structure, the user may implement his or her own code using any available, or preferred, software package (such as MATLAB, Matrix/X, or Control-C), or a standard computer language (such as FORTRAN, Pascal, or C). If the user has no such preference, the *L-A-S* software will be found to be both powerful and efficient. The additional advantage of using *L-A-S*, is that the computer code is available, ready for use.

All algorithms in the book follow the same general format. The process can be illustrated by the "system block" diagram below. The simple, yet powerful, idea is that the algorithm implements a single command that "transforms" the input data into the desired output data. Both sets of data are usually combinations of arrays representing specific elements of a particular system model. The corresponding syntax used throughout the text is

$$\mathbf{A}_1, \cdots, \mathbf{A}_n\ (Algorithm) \rightarrow \mathbf{B}_1, \cdots, \mathbf{B}_m$$

where the $\mathbf{A}_i$, $i=[1,n]$, are the required input arrays, and the $\mathbf{B}_i$, $i=[1,m]$, are the desired output arrays.

Input Data
Algorithm
Output Data

As an aid to using this book as a classroom text, we recommend the following order of study:

- Appendix A, as a review of matrix fundamentals. This review could be supplemented by the instructor.
- Appendix C, to develop an early familiarity with the *L-A-S* software.
- Chapter 1, for an introduction to the notation and definitions used in the text. The instructor should determine if this chapter is a sufficient review, and, if not, provide some supplementary material.
- Chapter 2 is basic to the understanding of sampled systems and should follow next.

- Chapter 3 then provides the major link from SISO to MIMO systems.
- Appendix B may be useful to study at this point.
- Next, Chapter 4 is the culmination of the modeling process and should be exercised in analysis, or design, problems chosen by the instructor.
- If time permits, Chapter 5 presents a general approach to system identification, based on the previously studied MIMO structure.

We hope that you, the reader, find that our method of presentation facilitates your learning of the theoretical concepts, as well as helping you to apply them to nontrivial problems.

The authors would like to recognize the interest and help of the graduate students at Virginia Polytechnic Institute and State University, as well as those from Yugoslavia, Brazil and Kuwait; and also the co-authors of the research which led to the creation of this text, who in a special way inspired us and greatly contributed to this material.

S. Bingulac and H.F. VanLandingham

Contents

Chapter 2 System Discretization

Chapter 3 System Modeling

Appendix C Introduction to *L-A-S*

Glossary of Symbols and Abbreviations

■	This symbol denotes the end of a development or example or an important equation.
$\mathbf{A},\mathbf{B},\mathbf{C},\ldots$	Boldface, capital letters denote matrices.
$\mathbf{x},\mathbf{y},\mathbf{z},\ldots$	Boldface, lowercase letters denote vectors.
$f,g,h,\ldots$	Italic, lowercase letters represent scalar valued functions.
$\alpha,\beta,\gamma,\ldots$	Greek letters typically denote scalar factors.
$\boldsymbol{x}(s),\boldsymbol{G}(s)$	Boldface, italic letters denote the corresponding
$\boldsymbol{x}(z),\boldsymbol{G}(z)$	Laplace or z-transformed quantity. For example, the vector $\boldsymbol{x}(s) = \mathcal{L}[\mathbf{x}(t)]$.
$\rho(\mathbf{A})$, rank($\mathbf{A}$)	The rank of the matrix $\mathbf{A}$.
$\nu(\mathbf{A})$, nullity($\mathbf{A}$)	The nullity of the matrix $\mathbf{A}$.
$\lambda(\mathbf{A})$	The set of eigenvalues of the matrix $\mathbf{A}$.
$\mathbf{A}^T,\mathbf{x}^T$	The transpose of the matrix $\mathbf{A}$ and the vector $\mathbf{x}$.
det($\mathbf{A}$)	The determinant of the matrix $\mathbf{A}$.
deg($\boldsymbol{g}(s)$)	The degree of the vector of polynomials $\boldsymbol{g}(s) = \{g_i(s)\}$.
diag$\{a,b,c\}$	Denotes a diagonal matrix with the given values as diagonal elements.
adj($\mathbf{A}$)	Denotes the adjoint matrix of the matrix $\mathbf{A}$.

tr(**A**)	Denotes the trace of the matrix **A**.
$\|\mathbf{A}\|$	The norm of the matrix **A**, also Norm(A). The *Frobenius norm*, the square-root of the sum of the squares of all entries of **A**, is used throughout the text.
$\approx$	Equals approximately
$\mathbf{M}^{-1}$	Inverse of a (square) nonsingular matrix
$\mathbf{M}^{+}$	Pseudo- (generalized) inverse of an $(n\times m)$ matrix **M**, satisfying: $\mathbf{M}^{+}\,\mathbf{M}\,\mathbf{M}^{+} = \mathbf{M}^{+}$ and $\mathbf{M}\,\mathbf{M}^{+}\,\mathbf{M} = \mathbf{M}$
$\mathbf{0}_{n,m}$	$(n\times m)$ zero matrix, n or m may be zero
$\mathbf{I}_{n,m}$	$(n\times m)$ identity matrix, $\mathbf{I} = \{e_{ij}\}$, $e_{ii} = 1$, $e_{ij} = 0$ for $i \neq j$.
N(**A**)	$(m\times s)$ null space matrix of the $(n\times m)$ matrix **A**, satisfying: $\mathbf{A}\,\mathbf{N}(\mathbf{A}) = \mathbf{0}_{n,s}$, where $s = m-r$, $r = \text{rank}(\mathbf{A})$
R(**A**)	$(n\times r)$ range space matrix of an $(n\times m)$ matrix **A**, satisfying: $r = \text{rank}[\mathbf{R}(\mathbf{A})] = \text{rank}(\mathbf{A})$.
C-T, D-T	Continuous-time, discrete-time, as in D-T system
ADC,DAC	Analog-to-digital converter, digital-to-analog converter
ZOH	Zero-order hold
SISO	Single-input and single-output
SI,SO	Single-input, single-output
MIMO	Multiple-input and/or multiple output
MI,MO	Multiple-input, multiple-output
PCI,POI	Pseudo-controllability and pseudo-observability indices
PCF,POF	Pseudo-controllable and pseudo-observable forms
CCF,OCF	Controllable canonical form, observable canonical form
PMF	Polynomial matrix form

Computer representation of polynomials:

- An n^{th} order polynomial $$a(s) = \sum_{i=0}^{n} a_i s^i \qquad \text{(G1)}$$

  is represented in the computer by the $(n+1)$-dimensional row array:

  $$\mathbf{a} = [\,a_0 \;\; a_1 \;\; \cdots \;\; a_n\,] = \{\,a_i\,\} \qquad \text{(G2)}$$

The relationship between the polynomial $a(s)$ and the row $\mathbf{a}$ could formally be written as

$$a(s) = \mathbf{a}\cdot \mathbf{i}(s) \ , \quad \text{where} \quad \mathbf{i}(s) = \begin{bmatrix} 1 \\ s \\ \vdots \\ s^n \end{bmatrix} \tag{G3}$$

- An nth order, $(p \times m)$ polynomial matrix

$$A(s) = \sum_{i=0}^{n} \mathbf{A}_i s^i = \{ a_{ij}(s) \} \tag{G4}$$

where $a_{ij}(s)$, $1 \leq i \leq p$ and $1 \leq j \leq m$ are polynomials of up to nth order, i.e. with $\mathbf{a}_{ij} = [\, a_{ij0} \ \ a_{ij1} \ \ \dots \ \ a_{ijn} \,] = \{ a_{ijk} \}$

$$a_{ij}(s) = \sum_{k=0}^{n} a_{ijk} s^k = \mathbf{a}_{ij}\cdot \mathbf{i}(s) \tag{G5}$$

are represented in the following two forms:

1. *Polynomial matrix form (PMF)*, $\mathbf{A}_p$, a $(pm \times (n+1))$ matrix defined by

$$\mathbf{A}_p = \begin{bmatrix} \mathbf{a}_{11} \\ \mathbf{a}_{21} \\ \cdots \\ \mathbf{a}_{p1} \\ \mathbf{a}_{12} \\ \cdots \\ \mathbf{a}_{p2} \\ \cdots \\ \mathbf{a}_{pm} \end{bmatrix} = \begin{bmatrix} a_{110} & a_{111} & \cdots & a_{11n} \\ a_{210} & a_{211} & \cdots & a_{21n} \\ \vdots & \vdots & \cdots & \vdots \\ a_{p10} & a_{p11} & \cdots & a_{p1n} \\ a_{120} & a_{121} & \cdots & a_{12n} \\ \vdots & \vdots & \cdots & \vdots \\ a_{p20} & a_{p21} & \cdots & a_{p2n} \\ \vdots & \vdots & \cdots & \vdots \\ a_{pm0} & a_{pm1} & \cdots & a_{pmn} \end{bmatrix} \tag{G6}$$

2. *"Row" equivalent polynomial matrix form (PMF-r)*, $\mathbf{A}_r$, a $(p \times m(n+1))$ matrix defined by

$$\mathbf{A}_r = [\, \mathbf{A}_0 \quad \mathbf{A}_1 \ \cdots \ \mathbf{A}_n \,] = \{ \mathbf{A}_k \} \tag{G7}$$

The relationships between the polynomial matrix $A(s)$ and coefficient matrices $\mathbf{A}_p$ and $\mathbf{A}_r$ are given by:

$$A(s) = \mathbf{A}_r \mathbf{I}_m(s) \;, \quad \text{where} \quad \mathbf{I}_m(s) = \begin{bmatrix} \mathbf{I}_m \\ \mathbf{I}_m s \\ \vdots \\ \mathbf{I}_m s^n \end{bmatrix} \tag{G8}$$

with $\mathbf{I}_m$ the $(m \times m)$ identity matrix.
Sometimes, if it is more convenient, a polynomial, $a(s)$, may be represented by the transpose of Eq.(G3), i.e.

$$a(s) = \mathbf{i}^T(s) \cdot \mathbf{a}^T \tag{G9}$$

Similarly, a polynomial matrix $A(s)$ may sometimes be represented by its "column" equivalent *polynomial matrix form* (*PMF-c*), $\mathbf{A}_c$, i.e.

$$A(s) = \mathbf{I}_p(s)\,\mathbf{A}_c \;, \text{ where } \mathbf{I}_p(s) = \begin{bmatrix} \mathbf{I}_p & \mathbf{I}_p s & \cdots & \mathbf{I}_p s^n \end{bmatrix} \tag{G10}$$

with the $(p(n+1) \times m)$ matrix $\mathbf{A}_c$ given by $\quad \mathbf{A}_c = \begin{bmatrix} \mathbf{A}_0 \\ \mathbf{A}_1 \\ \vdots \\ \mathbf{A}_n \end{bmatrix}$.

Computational Procedures:

In this text the *computational procedures* are alternatively referred to as *algorithms*. Computational procedures operate on, or manipulate, input data arrays

$$\mathbf{A}_1, \mathbf{A}_2, \ldots, \mathbf{A}_n$$

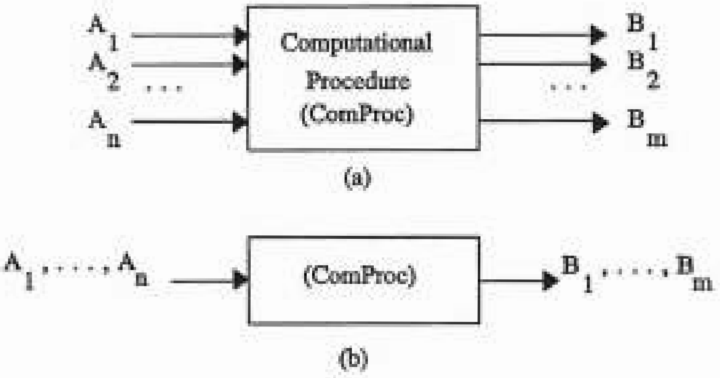
to produce (desired) output arrays

$$\mathbf{B}_1, \mathbf{B}_2, \ldots, \mathbf{B}_m$$

which may be interpreted in specific ways. Algorithms will be presented symbolically using specific input variables, output variables and the algorithm abbreviation. For the generic algorithm (abbreviated *ComProc* for computational procedure) and the associated input/output variables above the procedure would be represented as:

$$A_1, A_2, \ldots, A_n \ \ (ComProc) \rightarrow B_1, B_2, \ldots, B_m \qquad \text{(G11)}$$

Such a procedure can be visualized in the "operator" form as a "black box" block diagram as illustrated in the figure below.



Block Diagrams Representing a Generic Algorithm with Input and Output Variables: (a) Complete Form, (b) Abbreviated Form.

Either representation, symbolical or graphical, should be interpreted the same; namely, "*Apply the Algorithm 'ComProc' to the input data* $\{ A_i , 1 \le i \le n \}$ *in order to generate the output data* $\{ B_j , 1 \le j \le m \}$." It is worth mentioning that the above algorithm representation resembles the "post-fix," or reverse Polish notation, where the input arguments are specified first, followed by the algorithm name and ended by the output arguments.

Chapter 1 Introduction

A brief treatment of the background assumed for the remainder of the text is presented in this chapter. The presentation is not meant to be complete, but only indicative of the level of knowledge required. It is also appropriate that the reader review Appendix A for more details.

1.1 Systems

The investigations of engineers and applied mathematicians often require them to study complicated physical systems for the purpose of understanding and/or modifying their operation. A physical system is the starting point for the modeling process in which the engineer tries to formulate a mathematical description of the physical operation. The art of deriving a plant model is usually an iterative procedure of adding or deleting complexity to match observed performance, always with an eye toward obtaining the simplest model which matches the physical system measurements. The resulting model is an engineering compromise between complexity and model match which is naturally influenced by the computational power available for working with the model. For the remainder of the text the word *system* will refer to a mathematical model, not a physical system. The actual modeling process is not within the scope of the present study.

Models generally fall into one of two categories. One, *input output* models, also known as *external* models, are constructed from input output measurements without detailed knowledge of the internal mechanisms which produce the responses. The other, *internal* models, are usually well structured from "first principles," such as the laws of Newton for mechanical elements or of Kirchhoff for electrical interconnections. In the subsequent chapters different forms of models and their interrelations are considered. One form, *transfer functions*, is a basic external model type while another, *state models*, is an internal model type.

1.2 Scope of the Text

Most of the material in this text is oriented toward multivariable, linear, constant-parameter systems and deals with modeling and representation of such systems. Many different algorithms will be presented along with the theory of MIMO systems. The emphasis is on "learning by doing," working with the *L-A-S* software, or other means of implementing the algorithms, to more easily understand the theory and limitations of multivariable system modeling.

1.3 Background Material

The reader is assumed to have had a first course in control systems which typically covers single-input single-output (SISO) systems using classical frequency domain methods. This section provides a brief review of definitions from basic control theory. The topics include both the continuous-time (C-T) and discrete-time (D-T) state space models as well as transfer function matrices for both domains. In the next chapter additional discussion will be presented regarding the transformation of models between C-T and D-T domains.

1.3.1 Linearization

The basic techniques of this text deal with linear constant-parameter systems. The utility of these methods is based on the fact that such idealized system models are good representations of most physical systems near a controlled equilibrium point. For example, a large class of models can be represented in the C-T domain as follows:

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)), \quad \mathbf{x}(t_0) \\ \mathbf{y}(t) &= \mathbf{h}(\mathbf{x}(t), \mathbf{u}(t)) \end{aligned} \tag{1.1}$$

where $\mathbf{x}(t)$ is the $(n \times 1)$ *state vector* of the system, $\mathbf{u}(t)$ is an $(m \times 1)$ *vector of input signals* and $\mathbf{y}(t)$ is the $(p \times 1)$ *vector of output signals*. The general nonlinear dynamics are captured in the (assumed smooth) functions $\mathbf{f}(\mathbf{x},\mathbf{u})$ and $\mathbf{h}(\mathbf{x},\mathbf{u})$. It is because of these nonlinear dynamics that the system is typically analytically intractable. One method of reducing the scope of the model is to consider the linearization of Eqs.(1.1) about a known equilibrium solution given by $(\mathbf{x}_0, \mathbf{u}_0, \mathbf{y}_0)$ which, for simplicity, is taken to be a constant solution, i.e. each element of the 3-tuple is a constant vector and together they satisfy Eqs.(1.1) as shown in Eq.(1.3).

By formally expanding the above system in a Taylor series about the equilibrium point,

$$\begin{aligned} \dot{\mathbf{x}} &= \mathbf{f}|_0 + \frac{\partial \mathbf{f}}{\partial \mathbf{x}}|_0 \times (\mathbf{x}-\mathbf{x}_0) + \frac{\partial \mathbf{f}}{\partial \mathbf{u}}|_0 \times (\mathbf{u}-\mathbf{u}_0) \\ \mathbf{y} &= \mathbf{h}|_0 + \frac{\partial \mathbf{h}}{\partial \mathbf{x}}|_0 \times (\mathbf{x}-\mathbf{x}_0) + \frac{\partial \mathbf{h}}{\partial \mathbf{u}}|_0 \times (\mathbf{u}-\mathbf{u}_0) \end{aligned} \tag{1.2}$$

where the t-dependence has been dropped for notational convenience. The subscript notation of Eq.(1.2) indicates vectors or matrices evaluated at the equilibrium solution. Noting that by assumption,

$$\dot{\mathbf{x}}_0 = \mathbf{f}(\mathbf{x}_0, \mathbf{u}_0)$$
$$\mathbf{y}_0 = \mathbf{h}(\mathbf{x}_0, \mathbf{u}_0) \tag{1.3}$$

The linearized system becomes

$$\frac{d}{dt}\tilde{\mathbf{x}}(t) = \mathbf{A}\tilde{\mathbf{x}}(t) + \mathbf{B}\tilde{\mathbf{u}}(t), \quad \tilde{\mathbf{x}}(t_0)$$
$$\tilde{\mathbf{y}}(t) = \mathbf{C}\tilde{\mathbf{x}}(t) + \mathbf{D}\tilde{\mathbf{u}}(t) \qquad \blacksquare(1.4)$$

where the notation is that

$$\tilde{\mathbf{x}} = \mathbf{x} - \mathbf{x}_0, \quad \tilde{\mathbf{u}} = \mathbf{u} - \mathbf{u}_0, \quad \tilde{\mathbf{y}} = \mathbf{y} - \mathbf{y}_0 \tag{1.5}$$

and

$$\mathbf{A} = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}(\mathbf{x}_0, \mathbf{u}_0), \quad \mathbf{B} = \frac{\partial \mathbf{f}}{\partial \mathbf{u}}(\mathbf{x}_0, \mathbf{u}_0)$$
$$\mathbf{C} = \frac{\partial \mathbf{h}}{\partial \mathbf{x}}(\mathbf{x}_0, \mathbf{u}_0), \quad \mathbf{D} = \frac{\partial \mathbf{h}}{\partial \mathbf{u}}(\mathbf{x}_0, \mathbf{u}_0) \tag{1.6}$$

In keeping with the structure of the book we will introduce the first of many algorithms used to implement the theoretical developments. It is recommended that the reader implement the algorithm using the *L-A-S* code found in Appendix C. The best use of this text is to operate in a "hands-on" mode of exercising the algorithms as they appear in the reading. Some end-of-chapter problems are included to encourage computer usage. The purpose of this algorithm, denoted *LIN*, is to numerically calculate the linearized dynamic model (1.4) given a nonlinear model (1.1).

Algorithm *LIN*

Syntax: $\mathbf{p}, \mathbf{z}_0, \mathbf{dz}$ (*LIN*) $\Rightarrow$ **A, B, dif**

Purpose: Linearization of a system of nonlinear differential or difference equations, i.e. determination of the corresponding linearized state space representation.

Input/Output Arguments:

- $\mathbf{p} = \{p_i\}$, $i = 1, \ldots, k$, row containing parameters used in defining the nonlinear system.

- $\mathbf{z}_0$ = ($h \times 1$) column defining the nominal point at which the linearization is to be performed; $h = n + m$, n and m being the dimensions of the state, $\mathbf{x}(t)$, and input, $\mathbf{u}(t)$, vectors, respectively, i.e. $\mathbf{z}_0 = [\, \mathbf{x}_0^T \mid \mathbf{u}_0^T \,]^T$.
- $\mathbf{dz}$ = ($h \times 1$) column containing finite difference values; dx_i and du_j, for $i = 1, \ldots, n$ and $j = 1, \ldots, m$; i.e. $\mathbf{dz} = [\, \mathbf{dx}^T \mid \mathbf{du}^T \,]^T$.
- $\mathbf{A}$ = ($n \times n$) system matrix of the linearized model.
- $\mathbf{B}$ = ($n \times m$) input matrix of the linearized model.
- $\mathbf{dif}$ = ($n \times 1$) column defining the accuracy of the linearization.

Description: The system of nonlinear differential equations is given by:

$$\dot{\mathbf{x}}(t) = \mathbf{g}[\mathbf{x}(t), \mathbf{u}(t), \mathbf{p}] \tag{a}$$

where $\mathbf{x}(t)$, $\mathbf{u}(t)$ and $\mathbf{p}$ are the state, input and parameter vectors of dimensions n, m and k, respectively, while $\mathbf{g}(\cdot,\cdot,\cdot) = \{\, g_i(\cdot,\cdot,\cdot) \,\}$ is a n-dimensional vector-valued function.

The linearized model in the state space corresponding to (a) evaluated at

$$\mathbf{x}(t) = \mathbf{x}_0 \quad \text{and} \quad \mathbf{u}(t) = \mathbf{u}_0 \tag{b}$$

is given by

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

where the elements of $\mathbf{A} = \{\, a_{ij} \,\}$ and $\mathbf{B} = \{\, b_{ij} \,\}$ are calculated according to Eq.(1.6) by approximating the partial derivatives by finite differences.

The accuracy of the linearization process is measured by

$$\mathbf{dif} = \mathbf{g}(\mathbf{x}_0 + \mathbf{dx},\, \mathbf{u}_0 + \mathbf{du},\, \mathbf{p}) \;-\; \left(\mathbf{g}(\mathbf{x}_0, \mathbf{u}_0, \mathbf{p}) + [\mathbf{A} \;\; \mathbf{B}] \begin{bmatrix} \mathbf{dx} \\ \mathbf{du} \end{bmatrix} \right) \tag{d}$$

If $\mathbf{g}$ matches $\mathbf{f}$ in Eq.(1.1), then the first equation of Eq.(1.4) is forthcoming. Similarly, if $\mathbf{g}$ matches $\mathbf{h}$ in Eq.(1.1), then the second equation of Eq.(1.4) is obtained.

Note that in order to perform a linearization, a nonlinear vector-valued function $\mathbf{g}$ *appearing in (a) should first be defined.* The following notation is used in the algorithm steps:

$$\mathbf{z} = [\, \mathbf{x}^T ,\; \mathbf{u}^T \,]^T$$

Algorithm:

1. Define vectors $\mathbf{p}$, $\mathbf{z}_0$ and $\mathbf{dz}$
2. Define Algorithm *GZ* performing $\mathbf{p}, \mathbf{z}$ (*GZ*) $\Rightarrow \mathbf{g}$, i.e. calculating the vector-valued function $\mathbf{g}$
3. Set the number of rows (elements) in $\mathbf{z}_0 \Rightarrow h$

```
4.  Set $\begin{bmatrix} -1 & \cdots & -1 \\ & \mathbf{I}_h & \end{bmatrix} \Rightarrow \mathbf{T}$
5.  Set $\mathbf{p}$, $\mathbf{z}_0$ $(GZ) \Rightarrow \mathbf{g}_0$
6.  Set the number of rows (elements) in $\mathbf{g}_0 \Rightarrow n$
7.  Set $\mathbf{g}_0 \Rightarrow \mathbf{H}$
8.  Set $0 \Rightarrow i$
9.  Set $i + 1 \Rightarrow i$
10. Extract the $i^{th}$ element $dz(i) \Rightarrow dz_i$
11. Set 1 at the $i^{th}$ location of the $(h \times 1)$ zero-vector, $\mathbf{e}_0 \Rightarrow \mathbf{e}_i$
12. Set $\mathbf{z}_0 + \mathbf{e}_i\, dz_i \Rightarrow \mathbf{z}_i$
13. Set $\mathbf{p}$, $\mathbf{z}_i$ $(GZ) \Rightarrow \mathbf{g}_i$
14. Set $[\ \mathbf{H} \mid \mathbf{g}_i\ ] \Rightarrow \mathbf{H}$
15. If $i < h$ , go to 9; else, go to 16
16. Set diag$\{dz_1, \ldots, dz_h\} \Rightarrow \mathbf{D}$
17. Set $\mathbf{H}\ \mathbf{T}\ \mathbf{D}^{-1} \Rightarrow \mathbf{H}$
18. Partition $\mathbf{H} \Rightarrow [\ \mathbf{A} \mid \mathbf{B}\ ]$.  $\mathbf{A}$ has $n$ columns
19. Set $\mathbf{z}_0 + \mathbf{dz} \Rightarrow \mathbf{z}_1$
20. Set $\mathbf{p}$, $\mathbf{z}_1$ $(GZ) \Rightarrow \mathbf{g}_1$
21. Set $\mathbf{g}_1 - (\ \mathbf{g}_0 + \mathbf{H}\ \mathbf{dz}\ ) \Rightarrow \mathbf{dif}$
```

Algorithm Implementation:

The listing of the Algorithm *LIN* implemented using the *L-A-S* language is given in Appendix C. The vectors $\mathbf{g}_0$, $\mathbf{g}_i$ and $\mathbf{g}_1$ in Steps 5, 13, and 20 are calculated by the *L-A-S* subroutine *GZ*. As was emphasized earlier, prior to using Algorithm *LIN*, Algorithm *GZ* should be developed to calculate the vector function $\mathbf{g}(\mathbf{x}, \mathbf{u}, \mathbf{p})$.

Example 1.1 As an example of system linearization, consider the robot arm illustrated in Fig. 1.1. For a particular set of arm masses, lengths and inertias, the nonlinear equations of motion for the system are as follows:

$$\dot{w}_1 = \frac{1}{I}[T_1 - T_2 + .01\, w_1 w_2 \sin(2\theta_1)]$$

$$\dot{w}_2 = 100\, T_2 - \frac{1}{2} w_1^2 \sin(2\theta_2)$$

where

$$\dot{\theta}_1 = w_1\ , \quad \dot{\theta}_2 = w_2\ , \quad \text{and} \quad I = .07 + .06\cos^2(\theta_2) + .05\sin^2(\theta_2)$$

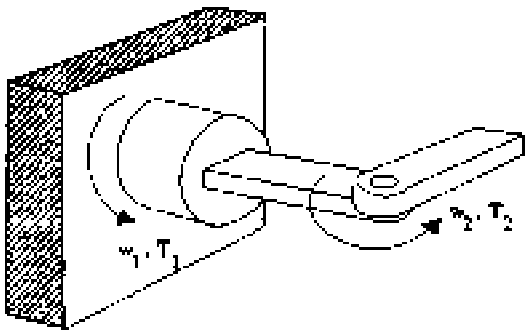


FIGURE 1.1 A Two Degree-of-Freedom (DOF) Robot Arm

Let the state vector, x, and input vector, u, be defined as follows:

$$\mathbf{x} = [w_1 \quad \theta_1 \quad w_2 \quad \theta_2]^T$$
$$\mathbf{u} = [T_1 \quad T_2]^T$$

Thus, the full-dimensional vector-valued function g(x, u, p) = g(z, p) which depends on the six-dimensional vector $\mathbf{z} = [\mathbf{x}^T, \mathbf{u}^T]^T$ is given by:

$$\mathbf{g} = [g_1 \quad g_2 \quad g_3 \quad g_4]^T$$

where

$$g_1 = \frac{p_1 z_1 z_3 \sin(2z_2) + z_5 - z_6}{p_2 + p_3 \cos^2(z_4) + p_4 \sin^2(z_4)}, \qquad g_2 = z_1$$

$$g_3 = -p_5 z_1^2 \sin(2z_4) + \frac{z_6}{p_1}, \qquad g_4 = z_3$$

The parameters in the above equation are the components of the parameter vector

$$\mathbf{p} = [.01 \quad .07 \quad .06 \quad .05 \quad .50]$$

The nonlinear differential equations described, i.e. the vector-valued function g(z, p), is linearized using Algorithm *LIN* for two nominal operating points: $\mathbf{z}_{01} = \mathbf{0}$ and $\mathbf{z}_{02} = [.1 \quad .2 \quad .3 \quad .4 \quad .5 \quad .6]^T$ while the "finite difference" vector dz = $[1 \quad 1 \quad 1 \quad 1 \quad 1 \quad 1]^T \times 10^{-1}$.

Using Algorithm *GZ* to develop the function g(z, p), and the vectors $\mathbf{z}_0$ and dz, previously defined, Algorithm *LIN* gives the following linearized pair {A, B}

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 7.692 & -7.692 \\ 0 & 0 \\ 0 & 100 \\ 0 & 0 \end{bmatrix}$$

The vector **dif**, defining the accuracy of the linearization is

$$\mathbf{dif} = [.15385E-12, \quad 0, \quad -.10000E-11, \quad 0]^T$$

Similarly, linearization about x_{e2} yields the pair {**A**, **B**}:

$$\mathbf{A} = \begin{bmatrix} .009 & .004 & .003 & -.043 \\ 1 & 0 & 0 & 0 \\ -.072 & 0 & 0 & -.007 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 7.783 & -7.783 \\ 0 & 0 \\ 0 & 100 \\ 0 & 0 \end{bmatrix}$$

with the accuracy of the linearization indicated by

$$\mathbf{dif} = [.08886E-08, \quad 0, \quad -.13940E-08, \quad 0]^T$$

1.3.2 State Models for Continuous-Time Systems

Based on the development of the previous section, we define a basic class of models for multi-input multi-output (MIMO) systems and discuss several fundamental system properties using this representation. Figure 1.2 illustrates the corresponding vector block diagram.

Definition 1.1 The *continuous time state (variable) model* is given by

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{A}\,\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t), \quad \mathbf{x}(t_0) \\ \mathbf{y}(t) &= \mathbf{C}\,\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t) \end{aligned} \qquad (1.7)$$

where **x** is an $(n \times 1)$ vector, **u** is an $(m \times 1)$ vector, **y** is a $(p \times 1)$ vector and the matrices **A**, **B**, **C**, and **D** are constant with compatible dimensions.

Typically, the coefficient matrices **A**, **B**, **C** and **D** are known numerically along with the initial state, $\mathbf{x}(t_0)$. By analogy to the scalar equation

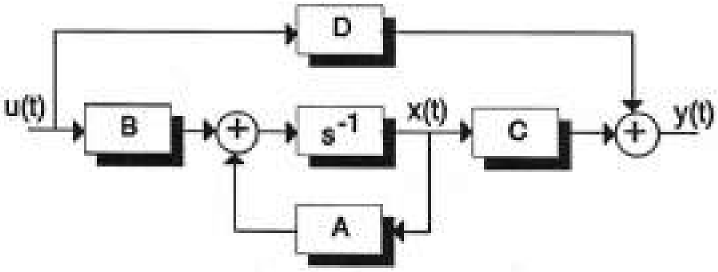


FIGURE 1.2 Vector Block Diagram of the State Model

$$\frac{d}{dt}x(t) = a x(t)$$

whose solution is $$x(t) = e^{at}\, x(t_0)$$

we introduce the definition of exp(**A***t*) for a square constant matrix, **A**, through the familiar infinite series for an exponential function.

Definition 1.2 The *transition matrix*, exp(**A***t*), for the $(n \times n)$ constant matrix **A** is

$$\exp(\mathbf{A}t) \triangleq \mathbf{I} + \mathbf{A}t + \mathbf{A}^2\frac{t^2}{2!} + \dots + \mathbf{A}^k\frac{t^k}{k!} + \dots \tag{1.8}$$

It is important to recognize that exp(**A***t*) has meaning *only* through Eq.(1.8), which itself is well defined since $\mathbf{A}^k$ is simply **A** multiplied by itself *k* times. The series Eq.(1.8) is absolutely convergent for any finite matrix **A***t*, this permits manipulation of the series on a term by term basis.

Several important results are reviewed in the following developments.

$$\frac{d}{dt}\exp(\mathbf{A}t) = \mathbf{A} + \mathbf{A}^2 t + \dots + \mathbf{A}^{k+1}\frac{t^k}{k!} + \dots \tag{1.9}$$

Clearly, by factoring out **A** as a pre- or post-multiplier,

$$\frac{d}{dt}\exp(\mathbf{A}t) = \mathbf{A}\,\exp(\mathbf{A}t) = \exp(\mathbf{A}t)\,\mathbf{A} \qquad ■ \tag{1.10}$$

showing that the matrices **A** and exp(**A***t*) commute.

Another important result is the familiar property of exponential function multiplication. Consider that

$$\exp(\mathbf{A}(t-\tau)) = \mathbf{I} + (t-\tau)\mathbf{A} + \frac{(t-\tau)^2}{2!}\mathbf{A}^2 + \cdots \tag{1.11}$$

Separately, it can be shown that

$$\begin{aligned} e^{\mathbf{A}t}e^{\mathbf{A}(-\tau)} &= (\mathbf{I} + t\mathbf{A} + \frac{t^2}{2!}\mathbf{A}^2 + \cdots)(\mathbf{I} - \tau\mathbf{A} + \frac{\tau^2}{2!}\mathbf{A}^2 - \cdots) \\ e^{\mathbf{A}t}e^{-\mathbf{A}\tau} &= (\mathbf{I} + (t-\tau)\mathbf{A} + \frac{(t-\tau)^2}{2!}\mathbf{A}^2 + \cdots) \end{aligned} \tag{1.12}$$

Thus, comparing Eqs.(1.11) and (1.12),

$$e^{\mathbf{A}t}e^{-\mathbf{A}\tau} = e^{\mathbf{A}(t-\tau)} \qquad \blacksquare(1.13)$$

Since it follows from Eq.(1.11) that

$$e^{\mathbf{A}(0)} = e^{0} = \mathbf{I} \qquad \blacksquare(1.14)$$

we readily deduce that

$$e^{-\mathbf{A}t} = [e^{\mathbf{A}t}]^{-1} \qquad \blacksquare(1.15)$$

by letting $\tau = t$ in Eq.(1.13), since the substitution gives $e^{\mathbf{A}t}\, e^{-\mathbf{A}t} = \mathbf{I}$.

With the above results the general solution to the state model will now be developed. Rewriting Eq.(1.7),

$$\frac{d}{dt}\mathbf{x}(t) - \mathbf{A}\mathbf{x}(t) = \mathbf{B}\mathbf{u}(t) \tag{1.16}$$

Upon premultiplication by exp(-At), the left-hand side becomes an exact derivative. The reader can easily check this using the relation that for $\mathbf{C} = \mathbf{AB}$, then

$$\dot{\mathbf{C}} = \dot{\mathbf{A}}\mathbf{B} + \mathbf{A}\dot{\mathbf{B}} \tag{1.17}$$

Integrating Eq.(1.16) from t_0 to t,

$$\begin{aligned} \int_{t_0}^{t} \frac{d}{d\tau}[e^{-\mathbf{A}\tau}\mathbf{x}(\tau)]d\tau &= \int_{t_0}^{t} e^{-\mathbf{A}\tau}\mathbf{B}\mathbf{u}(\tau)d\tau \\ e^{-\mathbf{A}t}\mathbf{x}(t) - e^{-\mathbf{A}t_0}\mathbf{x}(t_0) &= \int_{t_0}^{t} e^{-\mathbf{A}\tau}\mathbf{B}\mathbf{u}(\tau)d\tau \end{aligned} \tag{1.18}$$

Finally,

$$\mathbf{x}(t) = e^{\mathbf{A}(t-t_0)}\mathbf{x}(t_0) + \int_{t_0}^{t} e^{\mathbf{A}(t-\tau)}\mathbf{B}\mathbf{u}(\tau)d\tau \qquad \blacksquare(1.19)$$

is the *general solution to the state* in Eq.(1.7). Introducing Eq.(1.19) into the output equation of Eq.(1.7) with $t_0 = 0$,

$$\mathbf{y}(t) = \mathbf{y}_{ZI}(t) + \mathbf{y}_{ZS}(t) \qquad \blacksquare(1.20)$$

where

$$\mathbf{y}_{ZI}(t) = \mathbf{C}e^{\mathbf{A}t}\mathbf{x}(0) \qquad \blacksquare(1.21)$$

is the *zero-input response* and

$$\mathbf{y}_{ZS}(t) = \int_0^t \mathbf{C}e^{\mathbf{A}(t-\tau)}\mathbf{B}\mathbf{u}(\tau)d\tau + \mathbf{D}\mathbf{u}(t) \qquad \blacksquare(1.22)$$

is the *zero-state response*.

1.3.3 Discrete-Time State Models

In many cases the C-T system is to be interfaced with a digital computer. The usual analog-to-digital converters (ADCs) and digital-to-analog converters (DACs) are available on electronic boards which are connected to the computer and are jointly controlled by a synchronizing clock signal. The output of an ADC is therefore a sequence of numbers to be manipulated by the computer; however, each number is *quantized* due to the necessity of being represented as a finite length computer word. If the (usually small) errors between the ADC output and the ideal samples of the input signal are neglected, an acceptable model of the ADC interface is an ideal sampler, sampling at uniform intervals in time.

Similarly, the digital number sequence which is fed into the DAC is converted to a C-T signal by *holding* each sample constant until the next sample arrives. Control engineers refer to this type of action as a *zero-order hold* (*ZOH*).

Thus, if the above simplifications are made, a sampled-data system can be represented as shown in Fig. 1.3. The notation is that a D-T signal is given an argument of time equal to kT signifying that the values are defined only at integer multiples of the sample interval, T. The DAC interface is represented as a zero-order hold and the ADC interface, by an ideal sampler which is synchronized to the ZOH by a system clock signal, not explicitly shown in Fig.1.3.

The effect of the signal conversion into and out of the C-T system in Fig. 1.3 is to create an equivalent D-T system with input vector $\mathbf{u}(kT)$ and output vector $\mathbf{y}(kT)$. To establish the *ZOH equivalent model*, assume that the sampled state vector is known at $t_0 = kT$. From Eq.(1.19) with $t = kT+T$,

$$\mathbf{x}(kT+T) = e^{\mathbf{A}T}\mathbf{x}(kT) + \int_{kT}^{kT+T} e^{\mathbf{A}(kT+T-\tau)}\mathbf{B}\,d\tau\;\mathbf{u}(kT) \qquad (1.23)$$

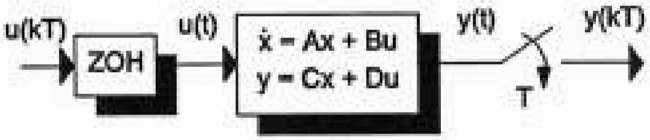


FIGURE 1.3 Sampled-Data System

where use has been made of the fact that $\mathbf{u}(t)$ is the output of the ZOH, i.e. that

$$\mathbf{u}(t) = \mathbf{u}(kT) \qquad \text{for} \quad kT \le t < kT + T \tag{1.24}$$

The resulting discrete-time model is given by

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{A}_d\mathbf{x}(k) + \mathbf{B}_d\mathbf{u}(k) , \quad \mathbf{x}(0) \\ \mathbf{y}(k) &= \mathbf{C}\,\mathbf{x}(k) + \mathbf{D}\,\mathbf{u}(k) \end{aligned} \qquad ■(1.25)$$

and is called the *ZOH equivalent model*. The notation is that

$$\mathbf{x}(k) = \mathbf{x}(kT), \quad \mathbf{u}(k) = \mathbf{u}(kT), \quad \mathbf{y}(k) = \mathbf{y}(kT) \tag{1.26}$$

where T is the sample interval. The matrices $\mathbf{A}_d$ and $\mathbf{B}_d$ are obtained from Eq.(1.23) with the change of variable $t = kT+T-\tau$ for the integral. The results are

$$\mathbf{A}_d = e^{\mathbf{A}T}, \qquad \mathbf{B}_d = \int_0^T e^{\mathbf{A}t}\mathbf{B}\,dt \qquad ■(1.27)$$

The output equation in Eq.(1.25) is simply the ideal sampled version of the output in Eq.(1.7). An alternative representation for $\mathbf{B}_d$ when $\mathbf{A}$ is nonsingular is given by

$$\mathbf{B}_d = \mathbf{A}^{-1}(e^{\mathbf{A}T}-\mathbf{I})\mathbf{B} = \sum_{i=0}^{\infty}\frac{(\mathbf{A}T)^i}{(i+1)!}\mathbf{B}T$$

As was done earlier in the chapter, we introduce the second algorithm used to implement the previous theoretical developments. Again, it is recommended that the reader implement the algorithm using the *L-A-S* code found in Appendix C. After having worked through the algorithms and the end-of-chapter exercises in this chapter, the reader will feel comfortable reviewing and exercising the algorithms in the remaining chapters. The purpose of this algorithm, denoted *EAT*, is to numerically calculate the transition matrix for a particular $\mathbf{A}$ matrix and scalar sampling interval, T.

Algorithm *EAT*

Syntax: T, $\mathbf{A}$, Nrm, N $(EAT) \Rightarrow \mathbf{A}_d$

Purpose: Calculation of the state transition matrix, $\mathbf{A}_d = e^{\mathbf{A}T}$

Input/Output Arguments:

- T = positive scalar
- $\mathbf{A}$ = $(n \times n)$ matrix
- Nrm = positive scalar, suggested value: $0.5 < Nrm < 1$
- N = integer defining number of terms in power series of Eq.(a)
- $\mathbf{A}_d$ = $(n \times n)$ matrix satisfying: $\mathbf{A}_d = \exp[\mathbf{A}T]$

Description: The matrix $\mathbf{A}_d$ is calculated using the truncated power series:

$$\mathbf{A}_d = \left[\sum_{i=0}^{N} \frac{(\mathbf{A}T/r)^i}{i!} \right]^k \quad , \text{ where } \quad r = 2^j, \tag{a}$$

The integer j is given by:

$$j = \left[\frac{\ln(\|\mathbf{A}\| T / Nrm)}{\ln 2} \right]_{\substack{integer\\part}} + 1 \tag{b}$$

Equation (b) guarantees that (see the Glossary for matrix norm)

$$\|\mathbf{A}T/r\| < Nrm \tag{c}$$

which leads to the satisfactory convergence of the power series of Eq.(a).

In order to save computational time and to reduce round-off errors, the N^{th} order polynomial $\mathbf{c}(\mathbf{A})$ used in Eq.(a),

$$\mathbf{c}(\mathbf{A}) = \sum_{i=0}^{N} \mathbf{A}^i c_i \, , \quad \text{where} \quad c_i = \frac{1}{i!}\left(\frac{T}{r}\right)^i \tag{d}$$

is evaluated by calculating the $(n\text{-}1)^{st}$ order polynomial $\mathbf{c}_r(\mathbf{A})$ given by

$$\mathbf{c}_r(\mathbf{A}) = \sum_{i=0}^{n-1} \mathbf{A}^i c_{ri} \tag{e}$$

where, according to the Cayley-Hamilton Theorem:

$$\begin{gathered} \mathbf{c}_r(\lambda_i) = \mathbf{c}(\lambda_i) \quad \text{for } \; i = 1, \cdots, n \\ \text{where} \quad \{\lambda_i\} = \lambda(\mathbf{A}) \end{gathered} \tag{f}$$

The coeficients $f_i = 1/i!$, $i=[1,N]$, are calculated by Algorithm *FACT*. Calculation of the coefficients c_{ri} of the polynomial $c_r(s)$ is done using Algorithm *POLR*. Calculation of the polynomial $\mathbf{c}_r(\mathbf{A})$ in Eq.(e) is accomplished with the *POM* algorithm. Algorithms such as *FACT*, *POLR* and *POM* below, not specifically discussed, are listed in Appendix C.

Algorithm:

```
1.  Define input arrays: $T$, $\mathbf{A}$, $Nrm$ and $N$
2.  Set $\left[\frac{\ln(\|\mathbf{A}\| T / Nrm)}{\ln(2)}\right]_{\substack{integer\\part}} + 1 \Rightarrow j$
3.  Set $2^j \Rightarrow r$
4.  Set $T/r \Rightarrow T_1$
5.  Set $1/i! \Rightarrow f_i$ and $f_i\, T_1^i \Rightarrow c_i$, for $i = 0, \cdots, N$
6.  Set $[\, c_0\ c_1 \cdots c_N \,] \Rightarrow \mathbf{C}$
7.  Set $\mathbf{C}$, $\mathbf{A}$ ($POLR$) $\Rightarrow \mathbf{C}_r$
8.  Set $\mathbf{C}_r$, $\mathbf{A}$ ($POM$) $\Rightarrow \mathbf{A}_d$
9.  If $j \le 0$, stop; else, go to 10
10. Set $0 \Rightarrow i$
11. Set $i+1 \Rightarrow i$
12. Set $\mathbf{A}_d\, \mathbf{A}_d \Rightarrow \mathbf{A}_d$
13. If $i < j$, go to 11; else, stop
```

Algorithm Implementation:

The listing of Algorithm *EAT* implemented using the *L-A-S* language is given in Appendix C. Algorithms *POLR* and *POM* are other algorithms also listed in Appendix C. The coefficients f_i of the $(1 \times N+1)$ row array $\mathbf{f}$ used by the algorithm are calculated by the *L-A-S* subroutine *FACT*. For more details see Chapter 2.

As is mentioned in Chapter 2, if the matrix $(n \times n)$ $\mathbf{A}$ is "diagonalizable," then $\mathbf{A}_d = e^{\mathbf{A}T}$ may also be calculated using Eq.(2.1), i.e.:

$$\mathbf{A}_d = \mathbf{M}\ \mathrm{diag}\{\, e^{\lambda_i T} \}\ \mathbf{M}^{-1}$$

where λ_i are eigenvalues of $\mathbf{A}$, and $\mathbf{M}$ is an $(n \times n)$ "modal" matrix containing n "ordinary" eigenvectors of $\mathbf{A}$ associated with eigenvalues λ_i. For more details see Chapter 2 as well as Appendices A and B.

In this *specific* case the matrix $\mathbf{A}_d$ may be calculated by Algorithm *EATj* given below which, in fact, implements Eq.(2.1).

Algorithm *EATj*

Syntax: T, $\mathbf{A}$ ($EATj$) $\Rightarrow \mathbf{A}_d$

For input/output arguments see Algorithm *EAT*.

Algorithm:

```
1. Define input arrays $T$ and $\mathbf{A}$
2. Set $\mathbf{A}$ ($JFR$) $\Rightarrow \mathbf{M}$
3. Set $\mathbf{A}$ ($EGV$) $\Rightarrow eg = \{ \lambda_i \}$
4. Set diag{ exp($\lambda_i T$) } $\Rightarrow$ ExJf
5. Set $\mathbf{M}$ ExJf $\mathbf{M}^{-1} \Rightarrow \mathbf{A}_d$
```

The listing of Algorithm *EATj*, implemented using the *L-A-S* language, is given in Appendix C. The calculations in Steps 2, 3 and 4 are performed using the algorithms:

JFR (the Jordan form of a diagonalizable square matrix),
EGV (the eigenvalues of a general square matrix), and
ExJf (a diagonal "Jordan" form having the scalars $e^{\lambda_i T}$ on the main diagonal),

respectively. All these algorithms are available in *L-A-S* as simple "operators." For details on the concept of *L-A-S* operators see Appendix C.

Linearly Interpolated Model

In a similar development one can work with higher-order hold devices, although they are not as commonly found in hardware form. A more accurate model is given by a linear interpolation between sampled values. This is referred to as the *trapezoidal rule* when used as an approximate integration technique. Thus, the C-T input signals are represented as straight lines between adjacent samples as illustrated in Fig. 1.4. From Fig. 1.4, for $kT \le t < kT+T$, we can write the relation

$$\mathbf{u}(t) = \frac{(t-kT)}{T}\mathbf{u}(kT+T) + \frac{(kT+T-t)}{T}\mathbf{u}(kT)$$

With this more elaborate model of the inputs the general solution to the state equation can be used (in much the same manner as was done in the previous development for the ZOH model) to arrive at the *linearly interpolated model*

$$\mathbf{x}(k+1) = \mathbf{A}_d\mathbf{x}(k) + \mathbf{B}_{d0}\mathbf{u}(k) + \mathbf{B}_{d1}\mathbf{u}(k+1) \qquad \blacksquare(1.28)$$

where the notation of Eq.(1.26) has been used to simplify the expression and

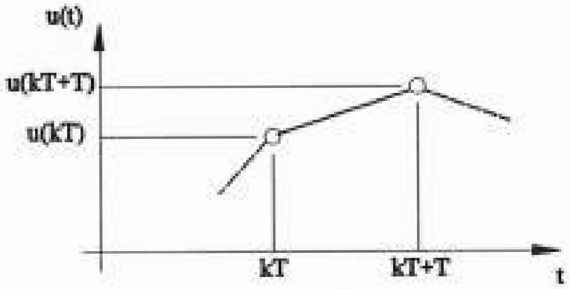


FIGURE 1.4 Linearly Interpolated Data

$$
\begin{aligned}
\mathbf{A}_d &= \sum_{i=0}^{\infty} \frac{T^i}{i!}\mathbf{A}^i \\
\mathbf{B}_{d0} &= \sum_{i=0}^{\infty} \frac{i+1}{(i+2)!} T^{i+1}\mathbf{A}^i\mathbf{B} \\
\mathbf{B}_{d1} &= \sum_{i=0}^{\infty} \frac{T^{i+1}}{(i+2)!}\mathbf{A}^i\mathbf{B}
\end{aligned}
\quad ■(1.29)
$$

Equation (1.28) evaluates the present state as a weighted sum of present input, past input, and past state. The coefficient matrices would have to be handled numerically as e.g. truncated versions of Eqs.(1.29). We will not pursue higher-order developments along this line; however, more details will be presented in the next chapter on algorithms for implementing this discretization. We summarize with the following definition.

Definition 1.3 The *discrete-time state (variable) model* is given by

$$
\begin{aligned}
\mathbf{x}(k+1) &= \mathbf{A}_d\mathbf{x}(k) + \mathbf{B}_d\mathbf{u}(k), \qquad \mathbf{x}(0) \\
\mathbf{y}(k) &= \mathbf{C}\,\mathbf{x}(k) + \mathbf{D}\,\mathbf{u}(k)
\end{aligned}
\qquad (1.30)
$$

where $\mathbf{x}$ is an $(n\times1)$ vector, $\mathbf{u}$ is an $(m\times1)$ vector, $\mathbf{y}$ is a $(p\times1)$ vector and the matrices $\mathbf{A}_d$, $\mathbf{B}_d$, $\mathbf{C}$ and $\mathbf{D}$ have corresponding compatible dimensions. Figure 1.5 illustrates the vector block diagram for this model.

Recursive Solution In the following development the subscript d is omitted for convenience. Working with Definition 1.3 and assuming that $\mathbf{x}(0)$ and $\mathbf{u}(k)$ are known for $k \geq 0$,

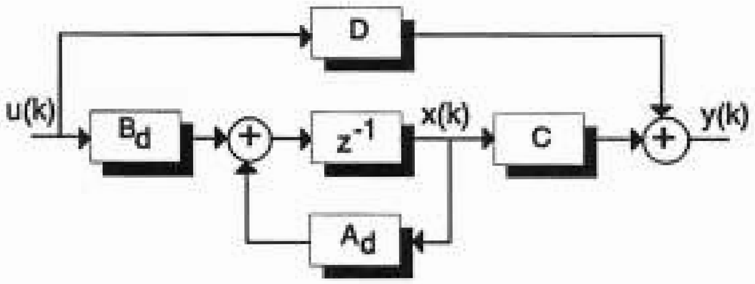


FIGURE 1.5 Discrete-Time State Model

$$\mathbf{x}(1) = \mathbf{A}\mathbf{x}(0) + \mathbf{B}\mathbf{u}(0)$$
$$\mathbf{x}(2) = \mathbf{A}\mathbf{x}(1) + \mathbf{B}\mathbf{u}(1) = \mathbf{A}^2\mathbf{x}(0) + \mathbf{A}\mathbf{B}\mathbf{u}(0) + \mathbf{B}\mathbf{u}(1)$$

Continuing this recursive process leads to the *general solution*:

$$\mathbf{x}(k) = \mathbf{A}^k\mathbf{x}(0) + \sum_{i=0}^{k-1} \mathbf{A}^{k-i-1}\mathbf{B}\mathbf{u}(i) \qquad \blacksquare(1.31)$$

Introducing Eq.(1.31) into the output equation of Eq.(1.30),

$$\mathbf{y}(k) = \mathbf{y}_{ZI}(k) + \mathbf{y}_{ZS}(k) \qquad \blacksquare(1.32)$$

where the *zero-input response*, $\mathbf{y}_{ZI}(k)$ is

$$\mathbf{y}_{ZI}(k) = \mathbf{C}\mathbf{A}^k\mathbf{x}(0) \qquad \blacksquare(1.33)$$

and the *zero-state response*, $\mathbf{y}_{ZS}(k)$ is

$$\mathbf{y}_{ZS}(k) = \sum_{i=0}^{k-1} \mathbf{C}\mathbf{A}^{k-i-1}\mathbf{B}\,\mathbf{u}(i) + \mathbf{D}\mathbf{u}(k) \qquad \blacksquare(1.34)$$

We will also review transform descriptions from the background of the state variable models. The Laplace and z-transforms provide these alternative descriptions of the systems of Definitions 1.1 and 1.3.

1.3.4 Controllability and Observability

Both *controllability* and *observability* are fundamental concepts in the design of control systems. The first answers the question of whether we can be assured of being able to influence the state of a system using the available inputs; and, the second answers a related question of whether all state variation is "visible" in some way through the measurements. In the following developments a D-T state space model of the form of Eq.(1.30) will be assumed as a starting point; but, since it is the structure of the state space model that is important and not whether the model is D-T or C-T, the end results will hold for both Eq.(1.30), as well as Eq.(1.7).

Controllability

By "controlling" a plant, we mean to use its available dynamic inputs (variables capable of being manipulated) and specify their time variations in order to obtain some desired response. We begin the discussion with the assumption that the D-T model in Eq.(1.30) is completely known and completely representative of the system to be controlled. Equation (1.30) has the general solution for its state given by Eq.(1.31). Here we recognize that it is the internal state and not just the output that is of concern.

Definition 1.4 The discrete-time state (variable) model given by Eq.(1.30) is *(completely state) controllable* if it is possible to force the state from any initial state $\mathbf{x}_0$ to an arbitrary "target" state $\mathbf{x}_f$ in a finite number of steps.

We will use this definition to derive a simple rank calculation to test for the property of controllability in a linear system. It is noted that for linear systems the problems concerning the transfer from an arbitrary initial state $\mathbf{x}_0$ to the origin $\mathbf{0}$, or the transfer from the origin $\mathbf{0}$ to an arbitrary final state $\mathbf{x}_f$ are equivalent. This latter perspective is often used to define the related concept of *reachability*. Recalling Eq.(1.31), $\mathbf{x}(k)$ is the state after k steps. Intuitively, if we can drive a system from one state to any other, then we can control the system in some more complicated manner. Expanding Eq.(1.31) and equating $\mathbf{x}(k)$ to $\mathbf{x}_f$, we can write

$$\mathbf{x}_f - \mathbf{A}^k\mathbf{x}(0) = [\mathbf{B} \quad \mathbf{AB} \quad \mathbf{A}^2\mathbf{B} \quad \cdots \quad \mathbf{A}^{k-1}\mathbf{B}]\begin{bmatrix}\mathbf{u}(k-1)\\ \mathbf{u}(k-2)\\ \cdots\\ \mathbf{u}(0)\end{bmatrix} \tag{1.35}$$

This expression is suggestive of solving for the set of input vectors which, when applied to the system, will cause the state to end up at $\mathbf{x}_f$ after k steps. Since the left side of the equation is arbitrary, the coefficient (partitioned) matrix must have full rank, i.e. n. However, we have not, as yet, specified k. Is it possible that the partitions $\mathbf{A}^{k-1}\,\mathbf{B}$ continue to generate linearly independent columns as k increases? In fact this is not the case. The Cayley-Hamilton theorem of matrix algebra tells us that $\mathbf{A}^n$ (where $\mathbf{A}$ is an $(n \times n)$ matrix) satisfies its own characteristic polynomial and, therefore, $\mathbf{A}^n$ can be written as a linear combination of powers of $\mathbf{A}$ less than n. Thus, with $k = n$ in Eq.(1.35) we maximize the number of linearly independent columns of the coefficient matrix. In this case the coefficient matrix is given a special name, i.e. the system *controllability matrix*.

Definition 1.5 The *controllability matrix* for the discrete-time state model given by Eq.(1.30) is defined as

$$\mathbf{Q}_c = [\,\mathbf{B} \quad \mathbf{AB} \quad \dots \quad \mathbf{A}^{n-1}\mathbf{B}\,]$$

Controllability is an inherent structural property of a system model, and equivalent systems will exhibit the same test results. The simple knowledge of whether a system is controllable, or not, is crucial to the subsequent state space control methods. Without controllability not all of the states can be "guided" by input manipulation. Unfortunately, the question of controllability gives rise to a *yes* or *no* answer and does not directly indicate the "degree of controllability," a measure of how close the system is to being uncontrollable. Yet another perspective is that if a particular model is not controllable, it simply means that additional actuation capability must be designed into the system.

We summarize this discussion with the following test and a subsequent algorithm for calculating the controllability matrix. It may be noted that in the Algorithm *Qc*, the definition of $\mathbf{Q}_c$ is slightly modified. In particular, it is known that for MIMO systems no new linearly independent columns of $\mathbf{Q}_c$ are added beyond the partition $\mathbf{A}^{n-m}\,\mathbf{B}$, where m is the number of (independent) columns in $\mathbf{B}$. Therefore, $\mathbf{Q}_c$ can be defined to end with the partition $\mathbf{A}^{n-m}\,\mathbf{B}$, rather than $\mathbf{A}^{n-1}\,\mathbf{B}$.

Controllability Test: The system described by Eq.(1.30), or that described by Eq.(1.7), is *controllable* if and only if its controllability matrix, $\mathbf{Q}_c$, given in Def. 1.5 has rank n, where n is the order of the system.

Algorithm *Qc*

Syntax: $\mathbf{A}, \mathbf{B}\ (Qc) \Rightarrow \mathbf{Q}_c$

Purpose: To calculate the $n \times (n\text{-}m+1)m$ matrix $\mathbf{Q}_c$=[$\mathbf{B}$ $\mathbf{AB}$ ... $\mathbf{A}^{n-m}\mathbf{B}$]

Description: The matrix $\mathbf{Q}_c$ is calculated by the following recursive process:

$$\mathbf{Q}_{ci} = [\,\mathbf{Q}_{c(i-1)} \mid \mathbf{A}^{i-1}\mathbf{B}\,]\,, \quad \text{for} \quad i = 1 \quad \text{to} \quad (n-m+1)$$

with initial condition that $\mathbf{Q}_{c0} = \mathbf{0}_{n,0}$. The matrix $\mathbf{Q}_c$ is equal to $\mathbf{Q}_{c\,(n-m+1)}$.

Notation: $\mathbf{0}_{n,0}$ represents a zero matrix with n rows and zero columns, and $[\mathbf{X}_1 \mid \mathbf{X}_2] \Rightarrow \mathbf{X}$ refers to concatenation "by columns," i.e.

$$\mathbf{X} = [\,\mathbf{X}_1 \mid \mathbf{X}_2\,]$$

Algorithm:

```
1.  Define matrices $\mathbf{A}$ and $\mathbf{B}$
2.  Set the number of columns in $\mathbf{A} \Rightarrow n$
3.  Set the number of columns in $\mathbf{B} \Rightarrow m$
4.  Set $n-m+1 \Rightarrow i_m$
5.  Set $\mathbf{B} \Rightarrow \mathbf{X}$
6.  Set $\mathbf{0}_{n,0} \Rightarrow \mathbf{Q}_c$
7.  Set $0 \Rightarrow i$
8.  Set $i + 1 \Rightarrow i$
9.  Set $[\mathbf{Q}_c \mid \mathbf{X}] \Rightarrow \mathbf{Q}_c$
10. Set $\mathbf{A}\,\mathbf{X} \Rightarrow \mathbf{X}$
11. If $i < i_m$, go to 8; else, stop
```

Algorithm Implementation:

The listing of the Algorithm *Qc* implemented using the *L-A-S* language is given in Appendix C. Note the striking similarity of the algorithm steps and the corresponding *L-A-S* operator statements.

Observability

As in the previous discussion, the D-T model of Eq.(1.30) will be assumed to accurately represent the system at hand. The concept of *observability* is a fundamental property of systems related to how the measurements, or outputs, interact with the system states. It has been shown that the simple problem of identifying the initial state, $\mathbf{x}(0)$, by observing a finite number of outputs is equivalent to knowing that the complete state information is transmitted to the outputs. Although we know from Eq.(1.32) that the general solution consists of two parts, only the zero-input response need be used to develop the condition under which the initial state can be identified from a finite number of outputs. The reason for this is that, since the model and inputs are known, the zero-state response could simply be calculated and subtracted away from the total solution.

Definition 1.6 The discrete-time state model given by Eq.(1.30) is *(completely state) observable* if it is possible to determine $\mathbf{x}(0)$ from knowledge of $\mathbf{u}(k)$ and $\mathbf{y}(k)$ over a finite number of time steps.

This definition will be used to develop a simple rank test for the property of observability of a system, similar to that developed for controllability above. Since without loss of generality we can assume that $\mathbf{u}(k) = \mathbf{0}$, as discussed previously, we can expand Eq.(1.33) to obtain

$$\begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \\ \cdots \\ \mathbf{CA}^k \end{bmatrix} \mathbf{x}(0) = \begin{bmatrix} \mathbf{y}(0) \\ \mathbf{y}(1) \\ \cdots \\ \mathbf{y}(k) \end{bmatrix}$$

We can solve for $\mathbf{x}(0)$ given the known vector on the right if and only if the n columns of the coefficient matrix on the left are linearly independent. Since the number of linearly independent columns of a matrix equals the number of linearly independent rows, we can add partitions $(\mathbf{C}\ \mathbf{A}^i)$ to have this affect. Again, as in the case of the controllability test, the maximal rank of the coefficient matrix is assured when the final partition is $(\mathbf{C}\ \mathbf{A}^{n-1})$. For this case the *observability matrix* is defined as follows.

Definition 1.7 The *observability matrix* for the discrete-time state model given by Eq.(1.30) is defined as

$$\mathbf{Q}_o = \begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \\ - \\ \mathbf{CA}^{n-1} \end{bmatrix}$$

Since $\mathbf{Q}_o$ has n columns and np rows, the maximum rank of $\mathbf{Q}_o$ is n. Thus, for an arbitrary set of n output measurements, we can solve for $\mathbf{x}(0)$ above if and only if $\mathbf{Q}_o$ has n linearly independent columns. Consequently, we have the following test.

Observability Test: The system described by Eq.(1.30), or that described by Eq.(1.7), is *observable* if and only if its observability matrix, $\mathbf{Q}_o$, given in Def. 1.7 has rank n, where n is the order of the system.

Like controllability, observability is an intrinsic property of a system. Equivalent state models exhibit identical test results. The test above provides a *yes* or *no* answer and, as with controllability, no direct measure of the "degree of observability." Since observability deals with how the sensors relate to the system dynamics, lack of observability can be interpreted as a need for more sensors for the system.

Algorithm *Qo*

Syntax: $\mathbf{A}, \mathbf{C}\ (Qo) \Rightarrow \mathbf{Q}_o$

Purpose: To calculate the $(n\text{-}p+1)p \times n$ matrix

$$\mathbf{Q}_o = \begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \\ \cdots \\ \mathbf{CA}^{n-p} \end{bmatrix}$$

where p is the number of rows in $\mathbf{C}$. Note that as in algorithm *Qc*, the rows of $\mathbf{Q}_o$ have been truncated, thereby redefining $\mathbf{Q}_o$ for ease of computation.

Description: The matrix $\mathbf{Q}_o$ is calculated by the following recursive process:

$$\mathbf{Q}_{oi} = \begin{bmatrix} \mathbf{Q}_{o(i-1)} \\ ---- \\ \mathbf{CA}^{i-1} \end{bmatrix}, \quad \text{for} \quad i = 1 \quad \text{to} \quad (n-p+1)$$

with initial condition that $\mathbf{Q}_{o0} = \mathbf{0}_{0,n}$. The matrix $\mathbf{Q}_o$ is equal to $\mathbf{Q}_{o\,(n-p+1)}$.

Notation: $\mathbf{0}_{0,n}$ = a zero matrix with zero rows and n columns. And

$$\begin{bmatrix} \mathbf{X}_1 \\ -- \\ \mathbf{X}_2 \end{bmatrix} \Rightarrow \mathbf{X}, \quad \text{means that} \quad \mathbf{X} = \begin{bmatrix} \mathbf{X}_1 \\ -- \\ \mathbf{X}_2 \end{bmatrix}$$

i.e. concatenation "by rows."

Algorithm:

1. Define matrices $\mathbf{A}$ and $\mathbf{C}$
2. Set the number of columns in $\mathbf{A} \Rightarrow n$
3. Set the number of rows in $\mathbf{C} \Rightarrow p$

```
4. Set $n-p+1 \Rightarrow i_p$
5. Set $\mathbf{C} \Rightarrow \mathbf{X}$
6. Set $\mathbf{0}_{0,n} \Rightarrow \mathbf{Q}_o$
7. Set $0 \Rightarrow i$
8. Set $i + 1 \Rightarrow i$
9. Set $\left[ \mathbf{Q}_o^T \mid \mathbf{X}^T \right]^T \Rightarrow \mathbf{Q}_o$
10. Set $\mathbf{X}\,\mathbf{A} \Rightarrow \mathbf{X}$
11. If $i < i_p$, go to 8; else, stop
```

Algorithm Implementation:
The listing of the Algorithm *Qo* implemented using the *L-A-S* language is given in Appendix C.

Duality Principle: It is found that for many types of calculations that a certain similarity exists. For example, in the previous tests for controllability and observability, there is a noticeable similarity in the calculations. Since this phenomenon shows up in several places, we will begin to explain with the following definition of *dual systems*.

Definition 1.8 If the discrete-time state model, *S*, is defined as

$$\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k)$$
$$\mathbf{y}(k) = \mathbf{C}\,\mathbf{x}(k) + \mathbf{D}\,\mathbf{u}(k)$$

then the *dual system*, *S'*, is given by

$$\mathbf{x}'(k+1) = \mathbf{A}^T\mathbf{x}'(k) + \mathbf{C}^T\mathbf{u}'(k)$$
$$\mathbf{y}'(k) = \mathbf{B}^T\mathbf{x}'(k) + \mathbf{D}^T\mathbf{u}'(k)$$

with its own states, inputs and outputs.

It is easy to see that the relationship of duality is "reflexive," i.e. if a system *S* is the dual of a system *S'*, then *S'* is also the dual of *S*. With regard to the previous tests of controllability and observability, we can say that:

- A system is controllable (observable) if and only if its dual system is observable (controllable). Specifically, $\mathbf{Q}_o$ may be calculated using *Qc* as follows:

$$\mathbf{A}^T,\ \mathbf{C}^T\ (Qc) \Rightarrow \mathbf{X}$$
$$\mathbf{X}^T \Rightarrow \mathbf{Q}_o$$

and similarly $\mathbf{Q}_c$ can be calculated using *Qo*. More will be said on this later.

1.3.5 Responses of State Space Models

Having created models of systems such as given in Eqs.(1.7) or (1.30), it is frequently necessary to numerically calculate and plot the responses of these systems from known initial conditions and input functions. In *L-A-S* it is convenient to symbolically represent the response of either D-T or C-T in the same way:

$$\mathbf{A},\ \mathbf{B},\ \mathbf{C},\ \mathbf{D},\ \mathbf{x}_0,\ \mathbf{u},\ T\ (CDSR) \rightarrow \mathbf{y}$$

where **A**, **B**, **C** and **D** represent the D-T or C-T state space model; $\mathbf{x}_0$ is the initial state; **u** is the $(m \times N)$ array of input vector samples, where N is the number of samples. The parameter T, shown as an input to Algorithm *CDSR* above, is a scalar which represents the total solution time for C-T models; for D-T models it should be set to zero or any negative scalar.

In the case of a C-T system, i.e. for $\mathbf{u}(t)$, the k^{th} column of **u** contains the vector $\mathbf{u}(t_k)$, where

$$t_k = \frac{(k-1)T}{N-1}, \quad \text{for} \quad 1 \le k \le N$$

The values of $\mathbf{u}(t)$ between samples are assumed to be linearly interpolated as illustrated in Fig. 1.4 and further described in the previous discussion of Linearly Interpolated Models. Finally, **y** is a $(p \times N)$ matrix containing solutions of either of the state models, Eqs.(1.7) or (1.30).

Plotting of the responses **y** may be accomplished in *L-A-S* by the commands:

```
*  y(T)=ytr             or by the MOS   *  y(T),T(DIS)=
*  ytr,T(DIS)=
```

where (T) is a matrix *transpose* operator. For more details see Appendix C.

1.3.6 Continuous-Time Transfer Matrices

Applying the Laplace transform to the state space model of Definition 1.1 with $t_0 = 0$,

$$\begin{aligned} s x(s) - x(0) &= \mathbf{A}x(s) + \mathbf{B}u(s) \\ y(s) &= \mathbf{C}x(s) + \mathbf{D}u(s) \end{aligned} \qquad (1.36)$$

Solving for $y(s)$,

$$y(s) = \mathbf{C}(s\mathbf{I}-\mathbf{A})^{-1}x(0) + [\mathbf{C}(s\mathbf{I}-\mathbf{A})^{-1}\mathbf{B} + \mathbf{D}]u(s) \qquad \blacksquare(1.37)$$

Definition 1.9 The *continuous-time transfer matrix*, $G(s)$, is the zero-state relation between the transformed input and output vectors, e.g.

$$G(s) = \mathbf{C}(s\mathbf{I}-\mathbf{A})^{-1}\mathbf{B} + \mathbf{D} \qquad \blacksquare(1.38)$$

Definition 1.10 The *characteristic polynomial* of the generic state model is the n^{th} order polynomial

$$a(s) = \det(s\mathbf{I} - \mathbf{A}) \qquad \blacksquare(1.39)$$

The transfer matrix $G(s)$ reduces to a scalar and is called the *transfer function* when the system has only one input and one output.

Definition 1.11 The transfer matrix $G(s)$ is said to be a *proper transfer matrix* if

$$\lim_{s\to\infty} G(s) = G_0 \qquad \blacksquare(1.40)$$

where G_0 is a constant (finite) matrix, not dependent on s.

Definition 1.12 The transfer matrix $G(s)$ is said to be a *strictly proper transfer matrix* if

$$\lim_{s\to\infty} G(s) = \mathbf{0} \qquad \blacksquare(1.41)$$

1.3.7 Discrete-Time Transfer Matrices

In a similar manner to the previous development the z-transform can be applied to the system of Definition 1.3, where the subscript d is omitted for convenience,

$$\begin{aligned} z\boldsymbol{x}(z) - z\boldsymbol{x}(0) &= \mathbf{A}\boldsymbol{x}(z) + \mathbf{B}\boldsymbol{u}(z) \\ \boldsymbol{y}(z) &= \mathbf{C}\boldsymbol{x}(z) + \mathbf{D}\boldsymbol{u}(z) \end{aligned} \qquad (1.42)$$

Thus, $$\boldsymbol{y}(z) = \mathbf{C}(z\mathbf{I}-\mathbf{A})^{-1}z\,\boldsymbol{x}(0) + [\mathbf{C}(z\mathbf{I}-\mathbf{A})^{-1}\mathbf{B} + \mathbf{D}]\,\boldsymbol{u}(z) \qquad \blacksquare(1.43)$$

Definition 1.13 The *discrete-time transfer matrix* $G(z)$ is the zero state relation between the z-transformed input and output vectors, e.g.

$$G(z) = \mathbf{C}(z\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D} \qquad \blacksquare(1.44)$$

The reader should recognize the similarity between the D-T and the C-T transfer matrices. Both are *algebraic* quantities so that Definitions 1.10, 1.11 and 1.12, as well as many others, may be interpreted in either the s- or the z-domain. In many places throughout this text we will rely on the readers' recognition that a certain operation performed in the s-domain would be identical in the z-domain. The following algorithm is one such case.

1.3.8 Leverrier's Algorithm

In the previous section it was seen that the *resolvent matrix*, $(s\mathbf{I} - \mathbf{A})^{-1}$, played an important roll in formulating the transfer matrix from the state-space model. Formally,

$$(s\mathbf{I} - \mathbf{A})^{-1} = \frac{\operatorname{adj}(s\mathbf{I} - \mathbf{A})}{\det(s\mathbf{I} - \mathbf{A})} \qquad \blacksquare(1.45)$$

where the denominator of Eq.(1.45) is an n^{th} order polynomial, $a(s)$, called the *characteristic polynomial* of the matrix $\mathbf{A}$. Explicitly,

$$a(s) = s^n + \alpha_{n-1}s^{n-1} + \alpha_{n-2}s^{n-2} + \cdots + \alpha_1 s + \alpha_0 \qquad \blacksquare(1.46)$$

From Eq.(1.45),

$$\operatorname{adj}(s\mathbf{I} - \mathbf{A})\cdot(s\mathbf{I} - \mathbf{A}) = \det(s\mathbf{I} - \mathbf{A}) = a(s)\,\mathbf{I} \qquad (1.47)$$

The adjoint matrix can be expanded as

$$\begin{aligned}\operatorname{adj}(s\mathbf{I} - \mathbf{A}) = \mathbf{I}s^{n-1} + (\mathbf{A} + \alpha_{n-1}\mathbf{I})s^{n-2} + (\mathbf{A}^2 + \alpha_{n-1}\mathbf{A} + \alpha_{n-2}\mathbf{I})s^{n-3} \\ + \cdots + (\mathbf{A}^{n-1} + \alpha_{n-1}\mathbf{A}^{n-2} + \cdots + \alpha_1\mathbf{I})\end{aligned} \qquad (1.48)$$

To see that this expansion is valid, the reader should take time to multiply Eq.(1.48) by $(s\mathbf{I} - \mathbf{A})$, thereby checking Eq.(1.47). Note that the *Cayley-Hamilton Theorem* requires that a (square) matrix satisfy its own characteristic equation, i.e. $a(\mathbf{A})=\mathbf{0}$, where $a(s)$ is given in Eq.(1.46). Let us formally write that

$$(s\mathbf{I} - \mathbf{A})^{-1} = \frac{1}{a(s)}\left[\mathbf{R}_{n-1}s^{n-1} + \mathbf{R}_{n-2}s^{n-2} + \cdots + \mathbf{R}_1 s + \mathbf{R}_0\right] = \frac{R(s)}{a(s)} \qquad (1.49)$$

The numerator polynomial matrix $R(s) = \{r_{ij}(s)\}$ is an $(n \times n)$ matrix of $(n\text{-}1)^{st}$ order polynomials, $r_{ij}(s)$, which can be expressed as follows:

$$R(s) = \sum_{l=0}^{n-1} \mathbf{R}_l s^l = \{ r_{ij}(s) \} \ , \quad r_{ij}(s) = \sum_{l=0}^{n-1} r_{ijl} s^l \tag{1.50}$$

It should be clear that the relation between the $(n \times n)$ real matrices $\mathbf{R}_l$ and the coefficients r_{ijl} of the polynomials $r_{ij}(s)$ is given by:

$$\mathbf{R}_l = \{ r_{ijl} \} \ , \quad \text{for} \ \ 1 \le i \le n \ , \ 1 \le j \le n \ , \ 0 \le l \le n-1$$

Comparing Eqs.(1.48) and (1.49), it may be concluded that:

$$\begin{gathered} \mathbf{R}_{n-1} = \mathbf{I} \ , \quad \mathbf{R}_{n-2} = \mathbf{R}_{n-1}\mathbf{A} + \alpha_{n-1}\mathbf{I} \ , \quad \mathbf{R}_{n-3} = \mathbf{R}_{n-2}\mathbf{A} + \alpha_{n-2}\mathbf{I} \ , \\ \cdots \ , \quad \mathbf{R}_0 = \mathbf{R}_1\mathbf{A} + \alpha_1\mathbf{I} \end{gathered} \tag{1.51}$$

Also, since the matrix $\mathbf{A}$ satisfies its own characteristic equation, $a(\mathbf{A}) = \mathbf{0}$,

$$\mathbf{R}_0\mathbf{A} + \alpha_0\mathbf{I} = \mathbf{0} \tag{1.52}$$

Leverrier's algorithm is a recursive method that calculates the coefficients of the characteristic polynomial in Eq.(1.46) as well as the matrix coefficients of adj($s\mathbf{I}$ - $\mathbf{A}$), as shown in Eq.(1.51). The recursion steps begin with a matrix result that the coefficient α_{n-1} in Eq.(1.46) is the negative of the sum of the eigenvalues of $\mathbf{A}$, which, in turn, is equal to the negative of the *trace* of $\mathbf{A}$. The *trace* of $\mathbf{A}$ is defined as the sum of the main diagonal elements of $\mathbf{A}$, denoted $tr(\mathbf{A})$.

$$\begin{aligned} &\mathbf{R}_{n-1} = \mathbf{I} \ , && \alpha_{n-1} = -tr(\mathbf{A}) \\ &\mathbf{R}_{n-2} = \mathbf{R}_{n-1}\mathbf{A} + \alpha_{n-1}\mathbf{I} \ , && \alpha_{n-2} = -\frac{1}{2}\, tr(\mathbf{R}_{n-2}\mathbf{A}) \\ &\cdots && \\ &\mathbf{R}_0 = \mathbf{R}_1\mathbf{A} + \alpha_1\mathbf{I} \ , && \alpha_0 = -\frac{1}{n}\, tr(\mathbf{R}_0\mathbf{A}) \end{aligned} \tag{1.53}$$

Equation (1.52) can be used as a numerical check on the above calculations.

Example 1.2 (Leverrier's Algorithm) Given the following matrix

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -2 & -4 & -3 \end{bmatrix}$$

We will calculate $(s\mathbf{I} - \mathbf{A})^{-1}$ using Algorithm *RESO* to implement Eq.(1.53) and check the result with Eq.(1.52). Following the recursion steps of Eq.(1.53),

$$\mathbf{R}_2 = \mathbf{I} \quad , \qquad \alpha_2 = 3$$

$$\mathbf{R}_1 = \mathbf{R}_2\mathbf{A} + \alpha_2\mathbf{I} = \begin{bmatrix} 3 & 1 & 0 \\ 0 & 3 & 1 \\ -2 & -4 & 0 \end{bmatrix}, \quad \alpha_1 = 4$$

$$\mathbf{R}_0 = \mathbf{R}_1\mathbf{A} + \alpha_1\mathbf{I} = \begin{bmatrix} 4 & 3 & 1 \\ -2 & 0 & 0 \\ 0 & -2 & 0 \end{bmatrix}, \quad \alpha_0 = 2$$

Equation (1.52) is satisfied identically; therefore,

$$(s\mathbf{I} - \mathbf{A})^{-1} = \frac{\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} s^2 + \begin{bmatrix} 3 & 1 & 0 \\ 0 & 3 & 1 \\ -2 & -4 & 0 \end{bmatrix} s + \begin{bmatrix} 4 & 3 & 1 \\ -2 & 0 & 0 \\ 0 & -2 & 0 \end{bmatrix}}{s^3 + 3s^2 + 4s + 2} = \frac{R(s)}{a(s)}$$

or, equally,

$$(s\mathbf{I} - \mathbf{A})^{-1} = \frac{\begin{bmatrix} s^2+3s+4 & s+3 & 1 \\ -2 & s^2+3s & s \\ -2s & -4s-2 & s^2 \end{bmatrix}}{s^3 + 3s^2 + 4s + 2} = \frac{[\, r_{ij}(s) \,]}{a(s)}$$

The reader is invited to further check the results by direct calculation of $(s\mathbf{I} - \mathbf{A})^{-1}$.

In the following we will discuss algorithms which not only calculate the resolvent matrix (Algorithm *RESO*), but also calculate the complete transfer matrix from a state space description (Algorithm *LALG*, using Leverrier's algorithm and Algorithm *SSTF*, which is not based on Leverrier's algorithm). In the sequel important notation is developed as well as additional examples for better understanding of MIMO system descriptions.

Algorithm *RESO*

Syntax: $\mathbf{A}$ (*RESO*) $\rightarrow$ $\mathbf{a}$, $\mathbf{R}_r$, $\mathbf{R}$

Purpose: Calculation of coefficients of the characteristic polynomial $a(s)$ and the numerator polynomial matrix $R(s)$ defining the resolvent $(s\mathbf{I} - \mathbf{A})^{-1}$ of the given (square) matrix $\mathbf{A}$ using the Leverrier algorithm.

Input/Output Arguments:

- $\mathbf{A}$ = given $(n \times n)$ matrix

- **a** = $(1 \times n+1)$ row containing coefficients a_i, $0 \le i \le n$, of $a(s)$. Coefficients are ordered by indices in increasing order.
- $\mathbf{R}_r$ = $(n \times n^2)$ matrix containing n $(n \times n)$ matrices $\mathbf{R}_i$, $0 \le i \le n$-1, defining the $(n \times n)$ polynomial matrix $R(s)$. Matrices $\mathbf{R}_i$ are ordered by indices in increasing order.
- $\mathbf{R}$ = $(n^2 \times n)$ matrix whose rows contain n coefficients, r_{ijh}, $0 \le h \le n$-1, of the polynomials $r_{ij}(s)$ defining the polynomial matrix $R(s)$. Rows r_{ij} are ordered "columnwise," i.e.

  row 1 contains coefficients of $r_{11}(s)$,
  row 2 contains coefficients of $r_{21}(s)$,
  ...
  row n contains coefficients of $r_{n1}(s)$,
  row $n+1$ contains coefficients of $r_{12}(s)$, (a)
  ...
  row $2n$ contains coefficients of $r_{n2}(s)$,
  ...
  row n^2 contains coefficients of $r_{nn}(s)$.

The matrix **R** is said to be in *polynomial matrix form* (*PMF*). The rows r_{ij} of **R** are:

$$r_{ij} = [\, r_{ij0} \quad r_{ij1} \quad \cdots \quad r_{ij(n-1)} \,]$$

Description: The expressions in (1.53) can be represented by the following recursive process:

$$\begin{aligned} \mathbf{R}_{n-i-1} &= \mathbf{R}_{n-i}\mathbf{A} + \mathbf{I}_n \alpha_{n-i} \\ \alpha_{n-i-1} &= -\frac{tr(\mathbf{R}_{n-i-1}\mathbf{A})}{i+1} \end{aligned} \qquad \text{(b)}$$

for $0 \le i \le n$, with initial conditions $\mathbf{R}_n = \mathbf{0}$ and $\alpha_n = 1$.
Note that the matrix $\mathbf{R}_{-1} = \mathbf{R}_0\mathbf{A} + \alpha_0\mathbf{I}$ calculated in the last step, i.e. for $i = n$, is not used in defining the numerator polynomial matrix $R(s)$. The norm of this matrix could be used for checking the accuracy of the calculation since:

$$\mathbf{R}_{-1} = \sum_{i=0}^{n} \alpha_i \mathbf{A}^i \qquad \text{(c)}$$

which according to the Cayley-Hamilton Theorem should be equal to the *zero* matrix.

In addition to the $(1 \times n+1)$ row **a** and the $(n \times n^2)$ matrix $\mathbf{R}_r$, namely

$$\begin{aligned} \alpha &= [\,\alpha_0 \quad \alpha_1 \quad \cdots \quad \alpha_{n-1} \quad \alpha_n\,] \\ \mathbf{R} &= [\,\mathbf{R}_0 \quad \mathbf{R}_1 \quad \cdots \quad \mathbf{R}_{n-1}\,] \end{aligned} \tag{d}$$

Algorithm *RESO* also calculates the $(n^2 \times n)$ matrix **R** whose rows contain the coefficients, r_{ijk}, of the $(n\text{-}1)^{st}$ order polynomials $r_{ij}(s)$, defining the numerator polynomial matrix $R(s)$ in Eq.(1.50). The arrays $\mathbf{R}_r$ and **R** contain the same scalars, r_{ijk}, but arranged differently. The reason for calculating both arrays is, as will become clear later, that some control algorithms require the form of $\mathbf{R}_r$, while others make use of the polynomials of **R** more directly.

Algorithm:

```
1.  Define square matrix $\mathbf{A}$
2.  Set number of columns in $\mathbf{A} \Rightarrow n$
3.  Set $\mathbf{I}_{n,n} \Rightarrow \mathbf{I}$
4.  Set $\mathbf{0}_{n,0} \Rightarrow \mathbf{R}_r$
5.  Set $\mathbf{I} \Rightarrow \mathbf{R}_i$
6.  Set $n^2 \Rightarrow nn$
7.  Set $\mathbf{0}_{nn,0} \Rightarrow \mathbf{R}$
8.  Set $1 \Rightarrow a$
9.  Set $0 \Rightarrow i$
10. Set $i+1 \Rightarrow i$
11. Set $\mathbf{R}_i \times \mathbf{A} \Rightarrow \mathbf{E}$
12. Set $-tr(\mathbf{E})/i \Rightarrow a_i$
13. Set $[\ a_i \mid a\ ] \Rightarrow a$
14. Set $[\ \mathbf{R}_i \mid \mathbf{R}_r\ ] \Rightarrow \mathbf{R}_r$
15. Set all $n$ columns of $\mathbf{R}_i$ into a single $n^2$ dimensional column $\Rightarrow \mathbf{r}_i$
16. Set $[\ \mathbf{r}_i \mid \mathbf{R}\ ] \Rightarrow \mathbf{R}$
17. Set $\mathbf{E} + a_i\,\mathbf{I} \Rightarrow \mathbf{R}_i$
18. If $i < n$, go to 10; else, stop
```

Algorithm Implementation: (See Appendix C for the *L-A-S* listing.)

1.3.9 Transfer Function Matrix Calculation

From the previous discussion it is clear that in addition to the resolvent matrix of Eq.(1.45) the transfer function matrix $G(s)$, defined in Eq.(1.38), or $G(z)$ in Eq.(1.44), is an important representation of a system. The next two algorithms were designed to calculate the transfer matrix from a given state space representation, {A,B,C,D}. Algorithm *LALG*, based on Leverrier's algorithm will be considered first. This algorithm calculates the coefficients of the characteristic polynomial $a(s)$ and the $(m \times p)$ numerator polynomial matrix, $W(s)$, related to the transfer matrix by

$$G(s) = \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D} = \frac{W(s)}{a(s)} \qquad (1.54)$$

for a given n^{th} order system with m-inputs and p-outputs.

Similarly, as in Eq.(1.49), the $(p \times m)$ polynomial matrix $W(s)$ can be represented as

$$W(s) = \sum_{h=0}^{n} \mathbf{W}_h s^h = \left\{ w_{ij}(s) = \sum_{h=0}^{n} w_{ijh} s^h \right\} \qquad (1.55)$$

where $\qquad \mathbf{W}_h = \{ w_{ijh} \}, \quad 1 \le i \le p, \quad 1 \le j \le m, \quad 0 \le h \le n$

Algorithm *LALG*

Syntax: $\qquad$ **A, B, C, D** (*LALG*) ⇒ **a, W**$_r$, **W**

Purpose: Calculation of coefficients of the characteristic polynomial $a(s)$ and the $(p \times m)$ numerator polynomial matrix $W(s)$ defining the transfer matrix $G(s)$ of a given state space representation {**A, B, C, D**} using the Leverrier algorithm.

Input/Output Arguments:

- {**A, B, C, D**} = state space representation of given system with n states, m inputs and p outputs.
- **a** = $(1 \times n+1)$ row array containing coefficients a_i, $0 \le i \le n$, of $a(s)$. Coefficients are ordered by indices in increasing order.
- **W**$_r$ = $(p \times (n+1)m)$ matrix containing $n+1$ $(p \times m)$ matrices $\mathbf{W}_i$, for $0 \le i \le n$, defining the $(p \times m)$ polynomial matrix $W(s)$ in (1.55). The matrices $\mathbf{W}_i$ are ordered by indices in increasing order.
- **W** = $(pm \times n+1)$ matrix whose rows contain $n+1$ coefficients w_{ijh}, for $0 \le h \le n$, of polynomials $w_{ij}(s)$ defining the polynomial matrix $W(s)$. As was true for Algorithm *RESO*, the rows w_{ij}, $1 \le i \le p$, $1 \le j \le m$, are ordered "columnwise," i.e.

  row 1 contains coefficients of $w_{11}(s)$,
  row 2 contains coefficients of $w_{21}(s)$,
  ...
  row p contains coefficients of $w_{p1}(s)$,
  row $p+1$ contains coefficients of $w_{12}(s)$, $\qquad$ (a)

...
row $2p$ contains coefficients of $w_{i2}(s)$,
...
row pm contains coefficients of $w_{im}(s)$.

The matrix **W** is said to be in the *polynomial matrix form* (*PMF*). The rows w_{ij} of **W** are :

$$w_{ij} = [\, w_{ij0} \quad w_{ij1} \quad \cdots \quad w_{ijn} \,]$$

Description: The calculation of the coefficients of $a(s)$ and matrices $\mathbf{W}_i$, $0 \le i \le n$, defining $G(s)$ can, as in algorithm *RESO*, be represented by the following recursive process:

$$\begin{aligned} \mathbf{R}_{n-i-1} &= \mathbf{R}_{n-i}\mathbf{A} + \mathbf{I}_n\,\alpha_{n-i} \\ \alpha_{n-i-1} &= -\frac{tr(\mathbf{R}_{n-i-1}\mathbf{A})}{i+1} \\ \mathbf{W}_{n-i} &= \mathbf{C}\,\mathbf{R}_{n-i}\mathbf{B} + \mathbf{D}\,\alpha_{n-i} \end{aligned} \tag{b}$$

for $0 \le i \le n$, with initial conditions $\mathbf{R}_n = \mathbf{0}$ and $\alpha_n = 1$.
Again note that the matrix $\mathbf{R}_{-1}$ calculated in the last step, i.e. for $i = n$, could be used for checking the accuracy of the algorithm.

In addition to the $(1 \times n+1)$ row matrix **a** and the $(p \times (n+1)m)$ matrix $\mathbf{W}_r$, namely

$$\begin{aligned} a &= [\, \alpha_0 \quad \alpha_1 \quad \cdots \quad \alpha_{n-1} \quad \alpha_n \,] \\ \mathbf{W}_r &= [\, \mathbf{W}_0 \quad \mathbf{W}_1 \quad \cdots \quad \mathbf{W}_{n-1} \quad \mathbf{W}_n \,] \end{aligned} \tag{c}$$

Algorithm *LALG* also calculates the $(pm \times n+1)$ matrix **W** whose rows contain the coefficients, w_{ijh}, of the $(n\text{-}1)^{st}$ order polynomials $w_{ij}(s)$, defining the numerator polynomial matrix $W(s)$ as in Eq.(1.55), i.e.

$$W(s) = \sum_{h=0}^{n} \mathbf{W}_h s^h = \left\{ w_{ij}(s) = \sum_{h=0}^{n} w_{ijh} s^h \right\} \tag{d}$$

where $\quad \mathbf{W}_h = \{ w_{ijh} \}\,, \quad 1 \le i \le p\,, \quad 1 \le j \le m\,, \quad 0 \le h \le n$

Algorithm:

1. Define square matrix **A**
2. Set number of columns in $\mathbf{A} \Rightarrow n$
3. Set $\mathbf{I}_{n,n} \Rightarrow \mathbf{I}$
4. Set $\mathbf{D} \Rightarrow \mathbf{W}_r$ and $\mathbf{I} \Rightarrow \mathbf{R}_i$

```
5.  Set all $m$ columns of $\mathbf{D}$ into a $pm$ dimensional column $\Rightarrow \mathbf{W}$
6.  Set $1 \Rightarrow a$
7.  Set $0 \Rightarrow i$
8.  Set $i+1 \Rightarrow i$
9.  Set $\mathbf{R}_i \times \mathbf{A} \Rightarrow \mathbf{E}$
10. Set $-tr(\mathbf{E})/i \Rightarrow a_i$
11. Set $\mathbf{C}\,\mathbf{R}_i\,\mathbf{B} + \mathbf{D}\,a_i \Rightarrow \mathbf{W}_i$
12. Set $[\, a_i \mid a \,] \Rightarrow a$
13. Set $[\, \mathbf{W}_i \mid \mathbf{W}_r \,] \Rightarrow \mathbf{W}_r$
14. Set all $m$ columns of $\mathbf{W}_i$ into a single $pm$ dimensional column $\Rightarrow \mathbf{w}_i$
15. Set $[\, \mathbf{w}_i \mid \mathbf{W} \,] \Rightarrow \mathbf{W}$
16. Set $\mathbf{E} + a_i\,\mathbf{I} \Rightarrow \mathbf{R}_i$
17. If $i < n$, go to 8; else, stop
```

Algorithm Implementation: (See Appendix C for the *L-A-S* listing.)

Example 1.3 (Transfer Matrix Calculation) For this example the given MIMO system {**A**, **B**, **C**, **D**} is (*in system matrix form*)

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} = \left[\begin{array}{ccc|cc} 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ -2 & -4 & -3 & 1 & 0 \\ \hline 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 2 & 0 & 0 \end{array}\right] \tag{1.56}$$

Note that *state matrix*, **A**, is identical to that used in Example 1.2 to illustrate the calculation of the resolvent matrix $(s\mathbf{I} - \mathbf{A})^{-1}$. In this example we are looking for the *transfer matrix*

$$G(s) = \mathbf{C}\,(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D} = \frac{\mathbf{W}(s)}{a(s)}$$

From applying Algorithm *LALG*:

$$a = [\,a_0 \;\; a_1 \;\; a_2 \;\; a_3\,] = [\,2 \;\; 4 \;\; 3 \;\; 1\,]$$

The *characteristic polynomial* is interpreted from this to be

$$a(s) = 2 + 4s + 3s^2 + s^3$$

In addition the algorithm provides $\mathbf{W}_r$, as follows

$$\mathbf{W}_r = [\mathbf{W}_0 \;\; \mathbf{W}_1 \;\; \mathbf{W}_2 \;\; \mathbf{W}_3] = \left[\begin{array}{cc|cc|cc|cc} 6 & 4 & 5 & 3 & 3 & 1 & 1 & 0 \\ -4 & 0 & -8 & -4 & 2 & 0 & 0 & 0 \end{array}\right]$$

and $\mathbf{W}$, which contains the same information in different form,

$$\mathbf{W} = \begin{bmatrix} w_{11} \\ w_{21} \\ w_{12} \\ w_{22} \end{bmatrix} = \begin{bmatrix} 6 & 5 & 3 & 1 \\ -4 & -8 & 2 & 0 \\ 4 & 3 & 1 & 0 \\ 0 & -4 & 0 & 0 \end{bmatrix}$$

From $\mathbf{W}$ we directly interpret that $W(s)$ is

$$W(s) = \begin{bmatrix} 6 + 5s + 3s^2 + s^3 & 4 + 3s + s^2 \\ -4 - 8s + 2s^2 & -4s \end{bmatrix}$$

which completes the transfer matrix $G(s) = W(s)/a(s)$.

As can be seen from the previous discussion, the Leverrier algorithm is both simple to understand and easy to inplement, but due to its recursive nature, it is susceptible to the accumulation of round-off errors. The next algorithm offers an alternative means for calculating the transfer matrix $G(s)$ without using the Leverrier algorithm.

Algorithm *SSTF*

Syntax: A, B, C, D (*SSTF*) ⇒ a, W

Purpose: Calculation of coefficients of the characteristic polynomial $a(s)$ and the $(p \times m)$ numerator polynomial matrix $W(s)$ defining the transfer matrix $G(s)$ of a given state space representation $\{\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}\}$ using polynomial manipulation.

Input/Output Arguments:

- $\{\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}\}$ = state space representation of given system with n states, m inputs and p outputs.
- $\mathbf{a}$ = $(1 \times n+1)$ row array containing coefficients a_i, $0 \leq i \leq n$, of $a(s)$. Coefficients are ordered by indices in increasing order.
- $\mathbf{W}$ = $(pm \times n+1)$ matrix whose rows contain $n+1$ coefficients w_{ijh}, for $0 \leq h \leq n$, of polynomials $w_{ij}(s)$ defining the polynomial matrix

$W(s)$. As in the case of Algorithm *RESO*, the rows w_{ij}, $1 \le i \le p$, $1 \le j \le m$, are ordered "columnwise," i.e.

row 1 contains coefficients of $w_{11}(s)$,
row 2 contains coefficients of $w_{21}(s)$,
...
row p contains coefficients of $w_{p1}(s)$,
row $p+1$ contains coefficients of $w_{12}(s)$,
...
row $2p$ contains coefficients of $w_{p2}(s)$,
...
row pm contains coefficients of $w_{pm}(s)$.

The matrix **W** is said to be in *polynomial matrix form* (*PMF*). The rows w_{ij} of **W** are

$$w_{ij} = [\, w_{ij0} \quad w_{ij1} \quad \cdots \quad w_{ijn} \,]$$

Description: The polynomials $w_{ij}(s)$ in the $(p \times m)$ matrix $W(s)$ can be calculated differently from Algorithm *LALG* starting with the following result:

$$w_{ij}(s) = v_{ij}(s) + d_{ij}\,a(s) \tag{a}$$

where $a(s)$ is the characteristic polynomial of **A**, d_{ij} is the ijth element of **D**, and $v_{ij}(s)$ is the $(n\text{-}1)$st order polynomial given by

$$v_{ij}(s) = \sum_{h=1}^{n} r_{ih}(s)\, b_{hj} \tag{b}$$

where b_{hj} is the hjth element of the input matrix **B** and the $(n\text{-}1)$st order polynomial $r_{ih}(s)$ is defined by

$$r_{ih}(s) = \det R_{ih}(s) \tag{c}$$

and $R_{ih}(s)$ is the $(n \times n)$ polynomial matrix obtained by substituting the hth row in $(s\mathbf{I} - \mathbf{A})$ by the ith row, c_i, of the output matrix **C**,

$$\mathbf{C} = \begin{bmatrix} c_1 \\ \cdots \\ c_i \\ \cdots \\ c_p \end{bmatrix} \tag{d}$$

Assuming that a computational procedure for calculating the characteristic polynomial of a square matrix is available (without using Leverrier's algorithm), the calculation of the polynomials $r_{ih}(s)$ could be performed by

$$r_{ih}(s) = r_{ihc}(s) - r_{hz}(s) \tag{e}$$

where $r_{ihc}(s)$ and $r_{hz}(s)$ are characteristic polynomials of the $(n \times n)$ matrices $\mathbf{R}_{ihc}$ and $\mathbf{R}_{hz}$, respectively, defined by

$$\mathbf{R}_{ihc} = \begin{bmatrix} a_1 \\ \cdots \\ a_{h-1} \\ c_i \\ a_{h+1} \\ \cdots \\ a_n \end{bmatrix}, \quad \mathbf{R}_{hz} = \begin{bmatrix} a_1 \\ \cdots \\ a_{h-1} \\ z \\ a_{h+1} \\ \cdots \\ a_n \end{bmatrix}, \quad \text{where} \quad \mathbf{A} = \begin{bmatrix} a_1 \\ \cdots \\ a_{h-1} \\ a_h \\ a_{h+1} \\ \cdots \\ a_n \end{bmatrix} \tag{f}$$

In other words, $\mathbf{R}_{ihc}$ is obtained from $\mathbf{A}$ by substituting the h^{th} row with the i^{th} row of $\mathbf{C}$, and $\mathbf{R}_{hz}$ is obtained from $\mathbf{A}$ by substituting the h^{th} row with the n-dimensional *zero* row, $\mathbf{z}$.

The alternate expression for calculating the polynomials $v_{ij}(s)$ in Eq.(a), to be used when $p > m$, is

$$v_{ij}(s) = \sum_{h=1}^{n} c_{ih}\, q_{hj}(s) \tag{g}$$

where c_{ih} is the ih^{th} element of the output matrix $\mathbf{C}$ and the $(n\text{-}1)^{st}$ order polynomial $q_{hj}(s)$ is defined by

$$q_{hj}(s) = \det Q_{hj}(s) \tag{h}$$

The $(n \times n)$ polynomial matrix $Q_{ij}(s)$ is obtained by substituting the h^{th} column in $(s\mathbf{I} - \mathbf{A})$ by the j^{th} column of the input matrix $\mathbf{B}$,

$$\mathbf{B} = [\,b_1 \;\cdots\; b_j \;\cdots\; b_m\,] \tag{i}$$

The calculation of polynomials $q_{ij}(s)$ can be performed by:

$$q_{hj}(s) = q_{hjb}(s) - q_{hz}(s) \tag{j}$$

where $q_{hjb}(s)$ and $q_{hz}(s)$ are characteristic polynomials of $(n \times n)$ matrices $\mathbf{Q}_{hjb}$ and $\mathbf{Q}_{hz}$, respectively, defined by

$$\begin{aligned} \mathbf{Q}_{hjb} &= \begin{bmatrix} a_1 & \cdots & a_{h-1} & c_j & a_{h+1} & \cdots & a_n \end{bmatrix} \\ \mathbf{Q}_{hz} &= \begin{bmatrix} a_1 & \cdots & a_{h-1} & z & a_{h+1} & \cdots & a_n \end{bmatrix} \\ \text{where} \quad \mathbf{A} &= \begin{bmatrix} a_1 & \cdots & a_{h-1} & a_h & a_{h+1} & \cdots & a_n \end{bmatrix} \end{aligned} \tag{k}$$

In other words, $\mathbf{Q}_{hjb}$ is obtained from $\mathbf{A}$ by substituting the h^{th} column with the j^{th} column of $\mathbf{B}$, and $\mathbf{Q}_{hz}$ is obtained from $\mathbf{A}$ by substituting the h^{th} column with the n-dimensional *zero* column, $\mathbf{z}$.

Algorithm:

```
1.  Set the number of columns in $\mathbf{A} \Rightarrow n$
2.  Set the number of columns in $\mathbf{B} \Rightarrow m$
3.  Set the number of rows in $\mathbf{C} \Rightarrow p$
4.  Set $n+1 \Rightarrow n_1$
5.  Set $0_{1,n} \Rightarrow \mathbf{z}$
6.  Set $\mathbf{I}_{n,n} \Rightarrow \mathbf{I}$
7.  Set $0_{n,0} \Rightarrow \mathbf{W}_C$
8.  Set $0_{0,n1} \Rightarrow \mathbf{W}_{BC}$
9.  Set $0 \Rightarrow j$
10. Set $j+1 \Rightarrow j$
11. Extract $j^{th}$ row from $\mathbf{C} \Rightarrow c_j$
12. Set $0_{0,n1} \Rightarrow \mathbf{W}_c$
13. Set $0 \Rightarrow i$
14. Set $i+1 \Rightarrow i$
15. Replace $i^{th}$ row of $\mathbf{A}$ by $c_j \Rightarrow A_{ci}$
16. Replace $i^{th}$ row of $\mathbf{A}$ by $z \Rightarrow A_{zi}$
17. Set coefficients of det $(s\mathbf{I} - \mathbf{A}_{ci}) \Rightarrow$ row $a_{ci}$
18. Set coefficients of det $(s\mathbf{I} - \mathbf{A}_{zi}) \Rightarrow$ row $a_{zi}$
19. Set $a_{zi} - a_{ci} \Rightarrow$ det
```

```
20. Set $\begin{bmatrix} \mathbf{W}_c \\ \det \end{bmatrix} \Rightarrow \mathbf{W}_c$
21. If $i < n$, go to 14; else, go to 22
22. Set $[\mathbf{W}_C \mid \mathbf{W}_c] \Rightarrow \mathbf{W}_C$
23. If $j < p$, go to 10; else, go to 24
24. Set $0 \Rightarrow i$
25. Set $i+1 \Rightarrow i$
26. Extract $i^{th}$ column from $\mathbf{B} \Rightarrow b_i$
27. Set $b_i^T \mathbf{W}_C \Rightarrow \mathbf{W}_{Cb}$
28. Rearrange $k\ n_1$ elements in the row $\mathbf{W}_{Cb} \Rightarrow (k \times n_1)$ matrix $\mathbf{W}_{Cbm}$
29. Set $\begin{bmatrix} \mathbf{W}_{BC} \\ \mathbf{W}_{Cbm} \end{bmatrix} \Rightarrow \mathbf{W}_{BC}$
30. If $i < m$, go to 25; else, go to 31
31. Set all $m$ columns of $\mathbf{D}$ into a $pm$ dimensional column $\Rightarrow \mathbf{d}_c$
32. Set coefficients of det $(s\mathbf{I} - \mathbf{A}) \Rightarrow$ row $\mathbf{a}$
33. Set $\mathbf{W}_{BC} + \mathbf{d}_c\ \mathbf{a} \Rightarrow \mathbf{W}$.
```

Remarks:

- Matrices $\mathbf{R}_{0c}$, $\mathbf{R}_{kc}$, and coefficients of the polynomials $r_{0c}(s)$, $r_{hc}(s)$ and $r_{ih}(s)$ are calculated in Steps 15 - 19;
- The matrix $\mathbf{R}_i$ contains the coefficients of $r_{ih}(s)$, $0 \leq h \leq n$, in Step 20; while the matrix $\mathbf{R}$ contains the coefficients of $r_{ih}(s)$, $1 \leq i \leq p$, $0 \leq h \leq n$, in Step 22;
- In Step 27 the matrix $\mathbf{V}_j$, containing coefficients of the polynomials $v_{ij}(s)$, $1 \leq i \leq k$, is formed;
- The matrix $\mathbf{V}$, formed by concatenating matrices $\mathbf{V}_j$ in Step 29, contains coefficients of all $v_{ij}(s)$; and, finally,
- The matrix $\mathbf{W}$, formed in Step 33, contains all coefficients of the polynomials $w_{ij}(s)$, defined by Eq.(a).

Algorithm Implementation: (See Appendix C for the *L-A-S* listing.)

It has been computationally verified that for higher-order systems, i.e. for $n > 10$, Algorithm *SSTF* is more accurate than Algorithm *LALG*.

The listing of Algorithm *SSTF* above, corresponding to Eqs.(a) to (f), should be used when $m < p$. The *L-A-S* implementation is given in Appendix C. If $p > m$, instead of using Eqs.(g) to (k), it is more convenient to use the concept of *duality* and apply the algorithm to the system representation given by

$$R_t = \{ \mathbf{A}^T, \mathbf{C}^T, \mathbf{B}^T, \mathbf{D}^T \}$$

and then to transpose the obtained $W(s)$. This sequence of operations is represented by the following three steps:

Set $\mathbf{A}^T, \mathbf{B}^T, \mathbf{C}^T, \mathbf{D}^T \Rightarrow \mathbf{A}_t, \mathbf{C}_t, \mathbf{B}_t, \mathbf{D}_t$.
Set $\mathbf{A}_t, \mathbf{C}_t, \mathbf{B}_t, \mathbf{D}_t$ $(SSTF) \Rightarrow$ $\mathbf{a}$, $\mathbf{W}_t$.
Set $\mathbf{W}_t^T \Rightarrow \mathbf{W}$.

1.4 Matrix Fraction Description (MFD)

An alternative representation to either the state space description or the transfer matrix description is the *matrix fraction description* (*MFD*). For a C-T MIMO system the MFD model is of the form

$$D(s)\,y(s) = N(s)\,u(s) \tag{1.57}$$

where $y(s)$ is the $(p\times 1)$ system output and $u(s)$ is the $(m\times 1)$ system input. The matrices $D(s) = \{ d_{ij}(s) \}$ and $N(s) = \{ n_{ij}(s) \}$ are *left coprime* $(p\times p)$ and $(p\times m)$ polynomial matrices. The orders of polynomials $d_{ij}(s)$ and $n_{ij}(s)$ satisfy:

$$\begin{aligned} 0 &\le \deg[d_{ij}(s)] \le k \\ 0 &\le \deg[n_{ij}(s)] \le k \end{aligned} \tag{1.58}$$

where $k \le n$, n being the order of the system.

In keeping with the notation already established, polynomials $d_{ij}(s)$ and $n_{ij}(s)$ will be represented by:

$$d_{ij}(s) = \sum_{h=0}^{k} d_{ijh}\,s^h \quad \text{and} \quad n_{ij}(s) = \sum_{h=0}^{k} n_{ijh}\,s^h \tag{1.59}$$

Similarly, polynomial matrices $D(s)$ and $N(s)$ may be written as:

$$D(s) = \sum_{h=0}^{k} \mathbf{D}_h\,s^h \quad \text{and} \quad N(s) = \sum_{h=0}^{k} \mathbf{N}_h\,s^h \tag{1.60}$$

where

$$\mathbf{D}_h = \begin{bmatrix} d_{11h} & \cdots & d_{1ph} \\ \vdots & \cdots & \vdots \\ d_{p1h} & \cdots & d_{pph} \end{bmatrix} \quad \text{and} \quad \mathbf{N}_h = \begin{bmatrix} n_{11h} & \cdots & n_{1mh} \\ \vdots & \cdots & \vdots \\ n_{p1h} & \cdots & n_{pmh} \end{bmatrix}$$

Two polynomial matrices are *left coprime* if they do not have common terms, or if:

$$\operatorname{rank}[\, D(s) \;|\; N(s) \,] = p \quad \text{for all} \quad s$$

In other words, it is assumed that all existing common terms in $D(s)$ and $N(s)$ have been cancelled. In some relevant literature the MFD model is referred to as an *auto-regressive-moving average* (*ARMA*) model. As is the case with state space models, the MFD representation is not unique, i.e. there are more than one pair of polynomial matrices $\{D(s),\, N(s)\}$ that will represent a given system.

One variation of an MFD model is the following model:

$$y(s) = \tilde{N}(s)\,\tilde{D}^{-1}(s)\,u(s) \tag{1.61}$$

which is sometimes expressed as

$$\begin{aligned} y(s) &= \tilde{N}(s)\,v(s) \\ \tilde{D}(s)\,v(s) &= u(s) \end{aligned} \tag{1.62}$$

where $v(s)$ is an auxiliary m dimensional vector.

It can be concluded that the MFD model is related to the system transfer matrix, $G(s)$ by

$$G(s) = D^{-1}(s)\,N(s) = \tilde{N}(s)\,\tilde{D}^{-1}(s) \tag{1.63}$$

Similarly, the $(p \times m)$ and $(m \times m)$ matrices $\tilde{N}(s)$ and $\tilde{D}(s)$ are *right coprime* if:

$$\operatorname{rank}\left[\, \tilde{N}^T(s) \;|\; \tilde{D}^T(s) \,\right] = m \quad \text{for all} \quad s \tag{1.64}$$

It is worth mentioning that in the case of SISO models, i.e. for $p = m = 1$, matrices $D(s)$ and $N(s)$ become scalar polynomials $d(s)$ and $n(s)$, respectively, and the coprime condition reduces to:

$$\operatorname{rank}[\, d(s) \;|\; n(s) \,] = 1 \quad \text{for all} \quad s \tag{1.65}$$

The condition of Eq.(1.65), in fact, implies that polynomials $d(s)$ and $n(s)$ have no common factors, i.e. there is no value $s = s_0$ for which both $d(s_0)$ and $n(s_0)$ are equal to zero. In other words, for $s = p_i$, $i=[1,n]$, i.e. system poles, $d(p_i) = 0$, but $n(p_i) \neq 0$; i.e. the transfer function $g(s) = n(s)/d(s)$ does not have any pole-zero cancellations. Similarly, if there are no common factors, then for $s = z_i$, i.e. system zeros for which $n(z_i) = 0$, $d(z_i) \neq 0$.

In the case of SISO systems, it is typically assumed that $d(s)$ is a *monic polynomial*, i.e.

$$d(s) = \sum_{i=1}^{n} d_i s^i \quad \text{where} \quad d_n = 1 \tag{1.66}$$

In Chapter 3 we extend this "normalization" concept to MIMO systems.

1.5 Summary

In this chapter a general background of knowledge has been set. The reader is expected to have a basic understanding of *linear control systems* such as one might acquire with a first course in Control Systems. The direction of the material of this text is to extend this fundamental knowledge to include a working computational facility with MIMO linear systems. The authors feel that understanding MIMO systems is complemented by the exercise obtained from studying the algorithms that are used to work with these systems.

The concept of system *linearization* was discussed early in the chapter since linearization is the basis of obtaining the models of concern from real-world models. In the remainder of the chapter *state space* models were used to describe various fundamental relationships between models of different types. The two most important relationships are:

(1) The relation between the *continuous-time* (*C-T*) models and the corresponding *discrete-time* (*D-T*) models which is required for most computer-aided calculations; and
(2) The relation between the *time domain* models, either C-T or D-T, usually specified as *state space* models, and the corresponding *frequency domain* models. The reader is expected to be familiar with both the s-domain and the z-domain.

In the latter portions of the chapter, starting with Leverrier's algorithm, the important problem of converting from a state space representation to a transfer matrix representation was considered. In the process of presenting the computational algorithms useful notation was introduced. Finally, in Section 1.4 the useful *matrix fraction description* (*MFD*) method of system representation was introduced.

The emphasis in this chapter has been on definitions and notation. In the subsequent chapters the emphasis will be on computational methods of converting between model types as well as accomplishing various operations that are useful in the analysis and design of control systems.

1.6 References

Although this chapter is presented as a transition chapter between the expected background of a classical control course and the subsequent study of multivariable systems, there are, no doubt, several topics for which the reader might want to obtain further information. This is a typical reference section in that at the end of each chapter a similar section gives suggestions for further reading, more or less, by chapter section.

Much of the material in this chapter can be found in more detailed form in many existing texts. One that is very attractive because of the many worked-out problems is Brogan (1991). In Chapter 15 of Brogan the reader can find an excellent review of *nonlinear system linearization*. Similarly, Chapters 3, 9 and 11 of Brogan offer relevant discussions of this chapter's topics. Other books that fall in the catagory of general references for this chapter are listed below. Another general text which emphasizes a similar "computer-aided" approach is Jamshidi (1992), particularly Chapters 2 and 3.

Specific references for the D-T models developed in Section 1.3.3 are VanLandingham (1985) and Haykin (1972). For details on the calculation of transfer functions see Bingulac (1975a and 1975b), and for controllability and observability, Bingulac and Luse (1990).

Bingulac, S. (1975), "On the calculation of the transfer function matrix," *IEEE Trans. on Automatic Control*, AC-20, 1, 134-135.

Bingulac, S. (1975), "On the calculation of matrix polynomials," *IEEE Trans. on Automatic Control*, AC-20, 3, 435-437.

Bingulac, S. and W. Luse (1990), "Computational simplification in controllability and Observability tests," *Proceeding of the 28th Allerton Conference*, University of Illinois, October 3-5, 1990, 527-528.

Brogan, W.L. (1991), *Modern Control Theory, 3rd Edition*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Haykin, S.S. (1972), "A unified treatment of recursive digital filtering," *IEEE Trans. on Automatic Control*, February 1972, pp 113-116.

Jamshidi, M. et al (1992), *Computer-Aided Analysis and Design of Linear Control Systems*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

VanLandingham, H.F. (1985), *Introduction to Digital Control Systems*, Macmillan Publishing Co., New York, NY.

1.7 Exercises

1.1 Using the following system state space representation, given in the system matrix partitioned form:

$$R_s = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}$$

namely,

$$R_s = \left[\begin{array}{ccccc|ccc} -1.0 & 1.0 & 1.0 & .0 & .0 & 1.0 & .0 & .0 \\ -.5 & -1.5 & .5 & -.5 & .0 & .5 & .5 & .0 \\ -.5 & .5 & -1.5 & .5 & .0 & -.5 & .5 & .0 \\ .0 & 1.0 & .0 & -2.0 & 1.0 & .0 & .0 & 1.0 \\ .5 & -.5 & -.5 & -.5 & -2.0 & .5 & -.5 & .0 \\ \hline .0 & 1.0 & 1.0 & 1.0 & 2.0 & .0 & .0 & .0 \\ .0 & 1.0 & .0 & 2.0 & 1.0 & .0 & .0 & .0 \end{array}\right]$$

Calculate:

(a) —the controllability matrix $\mathbf{Q}_c$ of the pair $\{\mathbf{A},\mathbf{B}\}$,
(b) —the observability matrix $\mathbf{Q}_o$ of the pair $\{\mathbf{A},\mathbf{C}\}$,
(c) —the ranks of both $\mathbf{Q}_c$ and $\mathbf{Q}_o$ to check the controllability and observability of R_s,
(d) —the resolvent matrix $\mathbf{R}$ in PMF and the characteristic polynomial $a(s)$ of $\mathbf{A}$,
(e) —the resolvent matrix $\mathbf{R}_r$ in a PMF-r form, and
(f) —the system transfer function matrix $G(s)$ in the form $G(s) = W(s)/d(s)$. Express $W(s)$ in PMF, i.e. determine the array $\mathbf{W}$.

Hints:

- Define the matrices of R_s and the scalar ϵ using the *L-A-S* operator *DMA*.
- Use operators *Qo* and *Qc* to calculate $\mathbf{Q}_o$ and $\mathbf{Q}_c$, respectively.
- The rank of a matrix is obtained using the operator *NRS*.
- Calculate the resolvent matrices $\mathbf{R}$ and $\mathbf{R}_r$ with the *L-A-S* subroutine *RESO.SUB*.
- The transfer function matrix $\mathbf{W}$ in PMF may be obtained with the operator *SSTF*.

- The results may be displayed on screen by using the flag *T* with the operator *OUT*.
- The results may be written to the *L-A-S* print file by using the flag *L* with the operator *OUT*.
- Use the subroutine *SYSM.SUB* to build the system matrix R_s.
- The individual matrices of R_s may be extracted by the subroutine *M14.SUB*.
- Store your program on a Disk Program file using the Interpreter Command (IC) *WPF* (or simply *W*).
- Recall your program from a Disk Program file using the IC *RPF* (or simply *R*).
- Remember that information on *L-A-S* operators or IC syntax may be obtained by: `HELP,<xyz>` or simply `h,<xyz>` and for subroutine syntax: `HELP,SUB,<xyz>` or `h,sub<xyz>`.

A version of an *L-A-S* program which solves this exercise is available in the *L-A-S* subdirectory C:\LAS\DPF\EXER11.DPF.

1.2 Linearize the nonlinear mathematical model of the robot arm, Fig. 1.1, Section 1.3.1. As elements of the parameter vector **p**, defining the system dynamics, use

$$\mathbf{p} = [\ 0.0125 \quad 0.07 \quad 0.06 \quad 0.05 \quad 0.4\ ]$$

Linearize the model around the following nominal point, $\mathbf{z}_0$

$$\mathbf{z}_0 = [\ 0.2 \quad 0.2 \quad 0.4 \quad 0.4 \quad 0.6 \quad 0.6\ ]^T$$

using for "finite differences," $\mathbf{dz} = [\ 1 \ \ 1 \ \ 1 \ \ 1 \ \ 1 \ \ 1\ ]^T\ 10^{-5}$.
Your results should show the resulting matrices **A** and **B**, as well as estimate the accuracy of the linearization.

Hints:

- Define vectors using the DMA operator.
- The subroutine LIN.SBR can be used for performing the linearization.
- The subroutine GZ.SUB, defining the system dynamics, is available in the *L-A-S* master subdirectory C:\LAS\SUB\ and will be called by LIN.SBR.
- See also the hints following Exercise 1.1.

A version of an *L-A-S* program which solves this exercise is available in the *L-A-S* subdirectory C:\LAS\DPF\EXER12.DPF.

1.3 A 5th order "weakly" controllable and "weakly" observable state space representation with $m=1$ input and $p=2$ outputs is given below:

Determine:

(a) —the eigenvalues λ_i, $i=[1,n]$, of $\mathbf{A}$,
(b) —the degrees of controllability and observability of each λ_i, $i=[1,n]$, of $\mathbf{A}$.
(c) Estimate the least controllable and the least observable eigenvalue λ_i, $i=[1,n]$, of $\mathbf{A}$.

$$R_s = \left[\begin{array}{ccccc|c} -1.0 & 2.0 & -.2 & .6 & .0 & -.202 \\ -2.0 & -1.0 & .4 & .8 & .0 & .405 \\ .0 & .0 & -2.0 & 1.0 & .0 & -.006 \\ .0 & .0 & -1.0 & -2.0 & .0 & -.012 \\ 2.0 & 2.0 & -.4 & -.8 & 1.0 & .595 \\ \hline -.03 & 1.06 & -2.0 & 1.03 & 1.0 & .0 \\ -.06 & -.03 & -1.0 & -2.0 & .0 & .0 \end{array}\right]$$

Hints:

- Define the representation R using either the operator DMA or INPM.
- The eigenvalues of $\mathbf{A}$ may be calculated using the operator EGV.
- The degrees of controllability and observability may be estimated using the subroutine COTS.SBR. See Section B.5 for more details on this topic.
- To plot the eigenvalues of the "auxiliary" matrices $\mathbf{A}_{cc}$ and $\mathbf{A}_{oo}$ use operator NIK. For scaling of the axes operator YXSC may be used.

A version of an *L-A-S* program which solves this exercise is available in the *L-A-S* subdirectory C:\LAS\DPF\EXER13.DPF.

Chapter 2 System Discretization

As pointed out in Chapter 1, with the widespread use of computers in control loops it is inevitable that control engineers will face problems associated with sampled-data systems. Such systems by their very definition contain a mixture of continuous-time and discrete-time signals. A common problem that arises with sampled-data control systems is to find the equivalent effect of continuous-time operations as seen by the computer in the loop. Typically, the modeling of the signal converters assumes an ideal uniform sampler for the analog-to-digital converter and a simple (zero-order) hold device synchronized with the samples for a digital-to-analog converter. With these assumptions one may find in many references the standard *zero-order hold* model, also known as the *step invariant* (*SI*) model which will be discussed subsequently.

2.1 Introduction

In addition to simple plant modeling with *SI* equivalents there are occasions, such as in digital redesign or system identification, that demand more accuracy between a given continuous-time (C-T) system and its discrete-time (D-T) equivalent model. In these instances higher-order discrete models are required. Two models which have been introduced for this purpose are the *bilinear transformation* (*BT*) (without prewarping) and a method which assumes a linearly interpolated input, also known as the *trapezoidal rule*. This latter method is referred to as a *ramp-invariant* (*RI*) model in contrast to the standard *ZOH* model being a *step-invariant* (*SI*) model. This model was introduced as a *linearly interpolated* (*input*) model in Chapter 1; see Eq.(1.29). There are many other useful models, but this chapter will focus on only these three methods of discretization as being the most useful in practice.

The reverse problem, called *continualization*, is that of reconstructing a C-T model from a given D-T model. This problem could arise, for instance, when measured discrete data are used to identify a C-T system. The particular method of continualization selected would depend on how the discrete data was derived (if known). The method of continualization is presented for each of the three discretization techniques, thereby offering the designer a great deal of flexibility in going between the continuous and the discrete domains.

For the SI and RI models both the forward, *discretization*, and reverse, *continualization*, problems may be viewed as functional transformations on a given matrix $\mathbf{A}$, i.e. in calculating $\exp(\mathbf{A}_c T)$ for discretization or $\ln(\mathbf{A}_d)/T$ for continualization. If the matrix $\mathbf{A}$ is transformed into its Jordan canonical form, $\mathbf{A}_J$, then

$$\mathbf{A} = \mathbf{Q}\mathbf{A}_J\mathbf{Q}^{-1}$$

The *modal* matrix $\mathbf{Q}$ contains as columns the eigenvectors and/or generalized eigenvectors of $\mathbf{A}$, depending on the eigenstructure of $\mathbf{A}$. Then, relating the problem at hand, it is well known that

$$f(\mathbf{A}) = \mathbf{Q}f(\mathbf{A}_J)\mathbf{Q}^{-1} \tag{2.1}$$

when the scalar function $f(x)$ is analytic at the eigenvalues of $\mathbf{A}$. This approach is convenient if $\mathbf{A}_J$ is diagonal because $f(\mathbf{A}_J)$ is then itself diagonal. However, in the general case this approach is very restrictive in that it is not so straightforward to evaluate either the matrix $\mathbf{Q}$ or $f(\mathbf{A}_J)$. Since it is desired to have robust algorithms to solve the continualization and discretization problems which are completely general, this method will not be pursued here.

Using basic properties of the exponential and logarithmic functions, a unified approach is presented in this chapter which provides simple robust algorithms for system discretization, as well as system continualization, using the three methods mentioned above. Examples are presented to illustrate the effectiveness of the algorithms, showing convergence properties versus the computation parameters used for truncation and scaling. In addition, practical guidelines are discussed, specifically for selecting the computation parameters and, more generally, for efficient computation of the matrix power series involved in the procedures.

2.2 Discretization Procedures

In the area of systems and controls, as well as related areas such as signal processing, it is useful to be able to discretize a given continuous-time system. This problem and its reverse problem of continualizing a discrete-time system are considered here. We assume a basic state variable representation for a continuous-time system as follows. A *state space realization* for a linear, continuous-time, constant parameter system consists of a 4-tuple of matrices; namely,

$$R_c = \{\mathbf{A}_c, \mathbf{B}_c, \mathbf{C}_c, \mathbf{D}_c\}$$

which defines the state model

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{A}_c\mathbf{x}(t) + \mathbf{B}_c\mathbf{u}(t) \\ \mathbf{y}(t) &= \mathbf{C}_c\mathbf{x}(t) + \mathbf{D}_c\mathbf{u}(t) \end{aligned} \tag{2.2}$$

where $\mathbf{x}(t)$, $\mathbf{u}(t)$ and $\mathbf{y}(t)$ are the state, input and output vectors with dimensions n, m and p, respectively, while the matrices $\mathbf{A}_c$, $\mathbf{B}_c$, $\mathbf{C}_c$ and $\mathbf{D}_c$ are constant matrices with compatible dimensions.

2.2.1 The Step-Invariant Model

The familiar *step-invariant* (SI) or ZOH equivalent discrete-time (D-T) model introduced in Section 1.3.3 assumes that the input vector $\mathbf{u}(t)$ in Eq.(2.2) is constant between (uniform) samples. The equivalent D-T model can be represented as

$$R_d = \{\mathbf{A}_d, \mathbf{B}_d, \mathbf{C}_d, \mathbf{D}_d\} \tag{2.3}$$

which implies the D-T state model

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{A}_d\mathbf{x}(k) + \mathbf{B}_d\mathbf{u}(k) \\ \mathbf{y}(k) &= \mathbf{C}_d\mathbf{x}(k) + \mathbf{D}_d\mathbf{u}(k) \end{aligned} \tag{2.4}$$

The matrices $\mathbf{A}_d$ and $\mathbf{B}_d$ are related to $\mathbf{A}_c$ and $\mathbf{B}_c$ in Eq.(2.2) by the relations, repeated here from Chapter 1:

$$\mathbf{A}_d = e^{\mathbf{A}_cT} = \sum_{i=0}^{\infty} \frac{(\mathbf{A}_cT)^i}{i!} \tag{2.5}$$

and
$$\mathbf{B}_d = \int_0^T e^{\mathbf{A}_ct}\mathbf{B}_c dt = \sum_{i=0}^{\infty} \frac{(\mathbf{A}_cT)^i}{(i+1)!}\mathbf{B}_cT \tag{2.6}$$

Also, if $\mathbf{A}_c$ is nonsingular,

$$\mathbf{B}_d = \mathbf{A}_c^{-1}(e^{\mathbf{A}_cT} - \mathbf{I})\mathbf{B}_c \tag{2.7}$$

And since the output $y(t)$ in Eq.(2.2) is assumed to be ideally sampled, the matrices $\mathbf{C}_d = \mathbf{C}_c$ and $\mathbf{D}_d = \mathbf{D}_c$.

The following algorithm, (*SI-C-D*), can be used to calculate the SI (ZOH) equivalent model of a continuous-time linear system. In particular, this algorithm is a numerically robust procedure for calculating $\mathbf{A}_d$ and $\mathbf{B}_d$ described above. The standard general method for calculating $\mathbf{A}_d$ is to compute a truncated version of Eq.(2.5). The problem with this approach is that for matrices $\mathbf{A}_c$ and sampling intervals T satisfying that

$$|\mathbf{A}_cT| > 1 \tag{2.8}$$

a truncated version of Eq.(2.5) may either require large N, leading to considerable round-off errors, or may not converge at all. The algorithm presented here is completely general.

It is easily shown that the SI model can be calculated using an intermediate matrix $\mathbf{E}$ as follows:

$$\mathbf{A}_d = \mathbf{I} + \mathbf{E}\mathbf{A}_cT \quad \text{and} \quad \mathbf{B}_d = \mathbf{E}\mathbf{B}_cT \quad \text{where} \quad \mathbf{E} = \sum_{i=0}^{\infty} \frac{(\mathbf{A}_cT)^i}{(i+1)!} \tag{2.9}$$

To resolve the problem associated with Eq.(2.8), it is possible to utilize the property of the exponential function that

$$\exp(x) = e^x = (e^{(x/r)})^r \tag{2.10}$$

The following algorithm extends this technique to permit calculation of both $\mathbf{A}_d$ and $\mathbf{B}_d$ under the condition of Eq.(2.8) as well as the condition that $\mathbf{A}_c$ may be singular.

First let us define a scaling factor $r = 2^j$ in terms of the scalar *Nrm*, *Nrm* < 0.5, and the integer *scaling parameter j* given by

$$j = \left[\frac{\ln(\|\mathbf{A}_c T\| / Nrm)}{\ln(2)}\right]_{\substack{integer\\ part}} + 1 \tag{2.11}$$

The series

$$\mathbf{A}_{d1} = \sum_{i=0}^{N} \frac{(\mathbf{A}_c T/r)^i}{i!} \tag{2.12}$$

will converge satisfactorily with the value of j given in Eq.(2.11) since $\|\mathbf{A}_c T/r\| < Nrm$. It is easily developed from the property in Eq.(2.10) that the series in Eq.(2.5), truncated to $N+1$ terms, satisfies the following recursive process

$$(\mathbf{A}_d)_{k+1} = (\mathbf{A}_d)_k^2\ , \quad for \quad k = 1, 2, \cdots, j \tag{2.13}$$

From Eq.(2.9) we formally obtain the recursion relationships

$$\begin{aligned}(\mathbf{A}_d)_k &= \mathbf{I} + \mathbf{E}_k \mathbf{A}_c T_k \\ (\mathbf{A}_d)_{k+1} &= \mathbf{I} + \mathbf{E}_{k+1} \mathbf{A}_c T_{k+1}\end{aligned} \tag{2.14}$$

Introducing Eq.(2.14) into Eq.(2.13), we obtain the following:

$$\begin{aligned}(\mathbf{I} + \mathbf{E}_k \mathbf{A}_c T_k)^2 &= \mathbf{I} + \mathbf{E}_{k+1} \mathbf{A}_c T_{k+1} \\ \mathbf{I} + 2\mathbf{E}_k \mathbf{A}_c T_k + (\mathbf{E}_k \mathbf{A}_c T_k)^2 &= \mathbf{I} + \mathbf{E}_{k+1} \mathbf{A}_c T_{k+1}\end{aligned} \tag{2.15}$$

This last equation leads us to the final recursion

$$\begin{aligned}T_{k+1} &= 2T_k \\ \mathbf{E}_{k+1} &= \mathbf{E}_k(\mathbf{I} + \mathbf{E}_k \mathbf{A}_c T_k/2)\end{aligned} \tag{2.16}$$

which must be initialized with

$$T_1 = \frac{T}{r} \quad \text{and} \quad \mathbf{E}_1 = \sum_{i=0}^{N} \frac{(\mathbf{A}_c T/r)^i}{(i+1)!} \tag{2.17}$$

The desired $\mathbf{E} = \mathbf{E}_{j+1}$. By the arguments given above for Eq.(2.12), the convergence of Eq.(2.17) is guaranteed. Once $\mathbf{E}$ has been calculated, $\mathbf{A}_d$ and $\mathbf{B}_d$ can be obtained using Eq.(2.9). Thus, this algorithm is similar to the introductory algorithm of Chapter 1, *EAT*, but it is more powerful in that the complete discrete-time *SI* equivalent model can be determined, not just the transition matrix. The algorithm is summarized in the following.

Algorithm *SI-C-D*

Syntax: T, $\mathbf{A}_c$, $\mathbf{B}_c$, *Nrm*, N (*SI-C-D*) $\Rightarrow \mathbf{A}_d$, $\mathbf{B}_d$

Purpose: Calculation of the SI D-T model

Input/Output Arguments:

T	= positive scalar
$\mathbf{A}_c$	= $(n \times n)$ matrix
$\mathbf{B}_c$	= $(n \times m)$ matrix
Nrm	= positive scalar, ≤ 0.5, defining the norm of the matrix $\mathbf{A}_c T/r$
N	= integer for truncation (suggested value $N \geq 16$)
$\mathbf{A}_d$	= $(n \times n)$ matrix satisfying Eq.(2.5)
$\mathbf{B}_d$	= $(n \times m)$ matrix satisfying Eq.(2.6)

Description: The matrices $\mathbf{A}_d$ and $\mathbf{B}_d$ are calculated using the truncated power series:

$$\mathbf{E} = \sum_{i=0}^{N} \frac{(\mathbf{A}_c T)^i}{(i+1)!} \tag{2.18}$$

modified according to the development given in Eqs.(2.11) to (2.17). Subsequently,

$$\mathbf{A}_d = \mathbf{I} + \mathbf{E}\mathbf{A}_c T\,, \qquad \mathbf{B}_d = \mathbf{E}\mathbf{B}_c T \tag{2.19}$$

as stated in Eq.(2.9).

Algorithm:

1. Define input arrays $\mathbf{A}_c$ and $\mathbf{B}_c$, scalars T and *Nrm* and integer N. The suggested values for *Nrm* and N are $Nrm \leq 0.5$ and $N \geq 16$
2. Calculate j using Eq.(2.11)
3. Calculate $\mathbf{E}_1$ and T_1 using Eq.(2.17)
4. For $k = 1, 2, \cdots, j$ calculate $\mathbf{E}_{k+1}$ recursively from Eq.(2.16)
5. Set $\mathbf{E}_{j+1} \Rightarrow \mathbf{E}$ and $\mathbf{I} + \mathbf{E}\mathbf{A}_c T \Rightarrow \mathbf{A}_d$, $\mathbf{B}_d \Rightarrow \mathbf{E}\mathbf{B}_c T$ from Eq.(2.19)

Algorithm Implementation:

The listing of Algorithm *SI-C-D* implemented using the *L-A-S* language is given in Appendix C. As in Algorithm *EAT*, Section 1.3.3, the matrix $\mathbf{E}_1$ in Eq.(2.17) is calculated using Algorithms *POLR* and *POM*, while the coefficients $f_i = 1/i!$, $i=[0,N]$, are obtained by Algorithm *FACT*.

2.2.2 Ramp-Invariant (Linearly Interpolated) Model

In Chapter 1 the *linearly interpolated model* was introduced which assumed that the input samples are interpolated as in Fig. 1.4, i.e. straight lines connecting the individual sampled values. Since this model is *ramp invariant* in the same way that the SI (ZOH) equivalent is *step invariant*, we will refer to this model as the *ramp-invariant* (RI) equivalent model. This model may be used for situations which require increased accuracy of discretization over the SI equivalent model of the previous section.

Although Eqs.(1.28) to (1.30) describe the basic approach, several important developments are necessary before achieving the desired robust conversion algorithm. First we note that there is one extra input matrix. The five-matrix state space model in the discrete domain will be represented by

$$R_{dr} = \{\mathbf{A}_d, \mathbf{B}_{d0}, \mathbf{B}_{d1}, \mathbf{C}_d, \mathbf{D}_d\} \tag{2.20}$$

which, in turn, can be written as

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{A}_d\mathbf{x}(k) + \mathbf{B}_{d0}\mathbf{u}(k) + \mathbf{B}_{d1}\mathbf{u}(k+1) \\ \mathbf{y}(k) &= \mathbf{C}_d\mathbf{x}(k) + \mathbf{D}_d\mathbf{u}(k) \end{aligned} \tag{2.21}$$

Later in the chapter an algorithm will be presented for the conversion of such five-matrix models to a standard four-matrix model.

The matrices $\mathbf{A}_d$, $\mathbf{E}$, $\mathbf{C}_d$ and $\mathbf{D}_d$ were described in the previous section, see Eqs.(2.9) and (2.10). To specify the remaining matrices, we define the series

$$\mathbf{F} = \sum_{i=0}^{\infty} \frac{(\mathbf{A}_c T)^i}{(i+2)!} \tag{2.22}$$

from which we obtain

$$\mathbf{B}_{d0} = (\mathbf{E} - \mathbf{F})\mathbf{B}_c T\,, \qquad \mathbf{B}_{d1} = \mathbf{F}\mathbf{B}_c T \tag{2.23}$$

Also, if $\mathbf{A}_c$ is nonsingular,

$$\mathbf{F} = (\mathbf{A}_d - \mathbf{I} - \mathbf{A}_c T)(\mathbf{A}_c T)^{-2} \tag{2.24}$$

Following the guidelines of Algorithm *SI-C-D*, it is desirable to create an algorithm which allows the condition of Eq.(2.8) and singular $\mathbf{A}_c$ matrices. The following is a development of this algorithm, referred to as Algorithm *RI-C-D*.

By comparing the power series in Eq.(2.9) with that of Eq.(2.22), it may be determined that the matrices **E** and **F** satisfy the following relation:

$$\mathbf{E} = \mathbf{I} + \mathbf{F}\mathbf{A}_c T \tag{2.25}$$

With j and r as previously defined in Eq.(2.11) let

$$\mathbf{E}_1 = \sum_{i=0}^{N} \frac{(\mathbf{A}_c T/r)^i}{(i+1)!} \tag{2.26}$$

Using Eq.(2.25), we can write, as was done to arrive at Eq.(2.15), the recursion equations

$$\mathbf{E}_k = \mathbf{I} + \mathbf{F}_k \mathbf{A}_c T_k \tag{2.27}$$

$$\mathbf{E}_{k+1} = \mathbf{I} + \mathbf{F}_{k+1} \mathbf{A}_c T_{k+1} \tag{2.28}$$

Now using Eq.(2.16) and eliminating $\mathbf{E}_k$ and $\mathbf{E}_{k+1}$ from Eqs.(2.27) and (2.28), the following relationship between $\mathbf{F}_k$ and $\mathbf{F}_{k+1}$ can be derived:

$$\begin{aligned} T_{k+1} &= 2\,T_k \\ \mathbf{F}_{k+1} &= 0.5\,\mathbf{F}_k + 0.25\,(\mathbf{I} + \mathbf{F}_k \mathbf{A}_c T_k)^2 \end{aligned} \tag{2.29}$$

which must be initialized with

$$T_1 = \frac{T}{r} \quad \text{and} \quad \mathbf{F}_1 = \sum_{i=0}^{N} \frac{(\mathbf{A}_c T/r)^i}{(i+2)!} \tag{2.30}$$

The desired $\mathbf{F} = \mathbf{F}_{j+1}$. The series will obviously converge satisfactorily with the value of j given in Eq.(2.11) since $\|\mathbf{A}_c T/r\| < Nrm$. Once **F** has been calculated, $\mathbf{A}_d$, $\mathbf{B}_{d0}$ and $\mathbf{B}_{d1}$ can be obtained using Eqs.(2.25), (2.19) and (2.23). Thus, this algorithm is similar to *SI-C-D*, but it is more powerful in that the more accurate discrete-time RI equivalent model can be determined. The algorithm is summarized in the following.

Algorithm *RI-C-D*

Syntax: T, $\mathbf{A}_c$, $\mathbf{B}_c$, Nrm, N (*RI-C-D*) $\Rightarrow$ $\mathbf{A}_d$, $\mathbf{B}_{d0}$, $\mathbf{B}_{d1}$

Purpose: Calculation of the RI D-T model

Input/Output Arguments:

T = positive scalar
$\mathbf{A}_c$ = $(n \times n)$ matrix
$\mathbf{B}_c$ = $(n \times m)$ matrix
Nrm = positive scalar, ≤ 0.5, defining the norm of matrix $\mathbf{A}_c T/r$
N = integer for truncation (suggested value $N = 16$)
$\mathbf{A}_d$ = $(n \times n)$ matrix satisfying Eq.(2.5)
$\mathbf{B}_{d0}$ = $(n \times m)$ matrix satisfying Eq.(2.23)
$\mathbf{B}_{d1}$ = $(n \times m)$ matrix satisfying Eq.(2.23)

Description: The matrices $\mathbf{A}_d$, $\mathbf{B}_{d0}$ and $\mathbf{B}_{d1}$ are calculated using the truncated power series:

$$\mathbf{F} = \sum_{i=0}^{N} \frac{(\mathbf{A}_c T)^i}{(i+2)!} \tag{2.31}$$

in the derived recursive form of Eqs.(2.29) and (2.30). Subsequently, once $\mathbf{F}$ is known

$$\mathbf{E} = \mathbf{I} + \mathbf{F}\mathbf{A}_c T\,, \qquad \mathbf{A}_d = \mathbf{I} + \mathbf{E}\mathbf{A}_c T \tag{2.32}$$

and

$$\mathbf{B}_{d0} = (\mathbf{E} - \mathbf{F})\mathbf{B}_c T\,, \qquad \mathbf{B}_{d1} = \mathbf{F}\mathbf{B}_c T \tag{2.33}$$

as stated in Eq.(2.23).

Algorithm:

1. Define input arrays $\mathbf{A}_c$ and $\mathbf{B}_c$, scalars T and Nrm and integer N. The suggested values for Nrm and N are $Nrm \leq 0.5$ and $N \geq 16$.
2. Calculate j using Eq.(2.11).
3. Calculate $\mathbf{F}_1$ and T_1 using Eq.(2.30).
4. For $k = 1, 2, \cdots, j$ calculate $\mathbf{F}_{k+1}$ recursively from Eq.(2.29).
5. Set $\mathbf{F}_{j+1} = \mathbf{F}$ and solve for $\mathbf{E}$, Eq.(2.25), $\mathbf{A}_d$, Eq.(2.19), $\mathbf{B}_{d0}$ and $\mathbf{B}_{d1}$, Eq.(2.23).

Algorithm Implementation:

The listing of Algorithm *RI-C-D* implemented using the *L-A-S* language is given in Appendix C. See also Algorithm *R5R4* below. For more details see either Algorithm *SI-C-D*, or Algorithm *EAT*, in Section 1.3.3.

General C-T ⇒ D-T Algorithm *EATF*

Recall that in order to calculate the output arrays $\mathbf{A}_d$, $\mathbf{B}_{d0}$ and $\mathbf{B}_{d1}$, Algorithm *RI-C-D* must calculate matrices $\mathbf{F}$ and $\mathbf{E}$ given by Eqs.(2.31) and (2.32), respectively. Note also that the same matrix $\mathbf{E}$ is needed in Algorithm *SI-C-D* for calculating $\mathbf{B}_d$, Eq(2.19). Thus, both SI D-T and RI D-T models could be calculated using a single algorithm if it has as output arguments the arrays:

$$\mathbf{A}_d,\ \mathbf{E}\ \text{ and }\ \mathbf{F}$$

Then, the desired SI or RI D-T model (or both) could be obtained simply by using Eqs.(2.19) and (2.33). To achieve this, the algorithm referred as *EATF* (the name stems from the calculation of e^{AT} using the matrix $\mathbf{F}$) is formulated:

Syntax: $T,\ \mathbf{A}_c,\ Nrm,\ N\ (EATF) \Rightarrow \mathbf{A}_d,\ \mathbf{E},\ \mathbf{F}$

while the input arguments are exactly the same as in *EAT*, *SI-C-D* and *RI-C-D*.

It is interesting to note that in the version:

$$T,\ \mathbf{A}_c,\ Nrm,\ N\ (EATF) \Rightarrow \mathbf{A}_d$$

where only one output argument is specified, Algorithm *EATF* is "formally" equal to Algorithm *EAT*, discussed in Chapter 1.

The listing of Algorithm *EATF*, implemented using the *L-A-S* language is given in Appendix C. Due to its generality and flexibility, it is recommended that this algorithm be used whenever either of the SI D-T or RI D-T models, or even just the matrix $\mathbf{A}_d$ is sought. In fact, this algorithm may be considered numerically advantageous over both of the Algorithms *EAT* and *SI-C-D*, since the terms in the truncated power series, Eq.(2.30), are divided by $(i+2)!$, while in *EAT* and *SI-C-D* the same terms are divided by $i!$ and $(i+1)!$, respectively, see Eqs.(2.12) and (2.17), which improves the convergence properties of the Algorithm *EATF*.

2.2.3 Bilinear Transformation

This algorithm is included because of its popularity with the signal processing community. The method is also known in the signal processing literature as *Tustin's approximation*. We will see in the development that the technique was designed for models in the transform domain. This algorithm, referred to as (*BT-C-D*), is known in the transform domain as a conversion from the s-domain to the z-domain using the direct substitution:

$$s = \frac{2}{T}\frac{(z-1)}{(z+1)} \tag{2.34}$$

Thus, introducing Eq.(2.34) into the C-T state equation of Eq.(1.36) and collecting terms to match Eq.(1.28), the *BT-C-D* Algorithm provides a five matrix discrete-time model as in Eq.(2.21) where, in this case, (with $a = 2/T$)

$$\begin{aligned} \mathbf{A}_d &= (a\mathbf{I} - \mathbf{A}_c)^{-1}(a\mathbf{I} + \mathbf{A}_c) \\ \mathbf{B}_{d0} &= \mathbf{B}_{d1} = (a\mathbf{I} - \mathbf{A}_c)^{-1}\mathbf{B}_c \end{aligned} \tag{2.35}$$

And, as in the previous results, $\mathbf{C}_d = \mathbf{C}_c$ and $\mathbf{D}_d = \mathbf{D}_c$. Algorithm *BT-C-D* may be symbolically represented by:

$$\mathbf{A}_c,\ \mathbf{B}_c,\ T\ (BT\text{-}C\text{-}D) \Rightarrow \mathbf{A}_d,\ \mathbf{B}_{d0},\ \mathbf{B}_{d1},\ \mathbf{P}$$

The listing of Algorithm *BT-C-D*, implemented using the *L-A-S* language, is given in Appendix C. Note that in the *L-A-S* implementation its name is BCDC, and the syntax is:

$$\mathbf{A}_c,\ \mathbf{B}_c,\ T,\ Icdc\ (BCDC) \Rightarrow \mathbf{A}_d,\ \mathbf{B}_{d0},\ \mathbf{B}_{d1},\ \mathbf{P}$$

where, for reasons explained later, the algorithm "flag" *Icdc* should have the value *Icdc* = 1. For more details see also Section 2.5.

Algorithm *R5R4*

Since both Algorithm *RI-C-D* and Algorithm *BT-C-D* result in a non-standard five-matrix model, it is useful to have a method of converting to a standard model as given in Eq.(2.4). Specifically, we describe the transformation from Eq.(2.21) to the following *equivalent* model:

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{A}_{de}\,\mathbf{x}(k) + \mathbf{B}_{de}\,\mathbf{u}(k) \\ \mathbf{y}(k) &= \mathbf{C}_{de}\,\mathbf{x}(k) + \mathbf{D}_{de}\,\mathbf{u}(k) \end{aligned} \tag{2.36}$$

The simplest computational procedure for obtaining the conversion to standard state model is derived using the identity of transfer function matrices, i.e.

$$\begin{aligned} &\mathbf{C}_d(z\mathbf{I} - \mathbf{A}_d)^{-1}(\mathbf{B}_{d0} + z\mathbf{B}_{d1}) + \mathbf{D}_d \\ &= \mathbf{C}_{de}(z\mathbf{I} - \mathbf{A}_{de})^{-1}\mathbf{B}_{de} + \mathbf{D}_{de} \end{aligned} \tag{2.37}$$

The detailed algorithm, referred to as Algorithm *R5R4*, is presented in the following.

In this development first consider the two D-T state space representations:

$$R_5 = \{\mathbf{A},\ \mathbf{B}_0,\ \mathbf{B}_1,\ \mathbf{C},\ \mathbf{D}\} \quad \text{and} \quad R_4 = \{\mathbf{A}_e,\ \mathbf{B}_e,\ \mathbf{C}_e,\ \mathbf{D}_e\} \tag{2.38}$$

defining the state models of Eqs.(2.21) and (2.4), respectively, where the *d* notation has been dropped for convenience. Since these two models represent the same D-T system, the corresponding transfer function matrices should be the same. Thus, we obtain the following equality:

$$\mathbf{C}(z\mathbf{I}-\mathbf{A})^{-1}(\mathbf{B}_0+z\mathbf{B}_1)+\mathbf{D} = \mathbf{C}_e(z\mathbf{I}-\mathbf{A}_e)^{-1}\mathbf{B}_e+\mathbf{D}_e \qquad (2.39)$$

Also the two transfer matrices should have identical characteristic polynomials. So, without loss of generality, it may be assumed that in both representations the system and output matrices are equal, i.e.

$$\mathbf{A}_e = \mathbf{A} \qquad \text{and} \qquad \mathbf{C}_e = \mathbf{C} \qquad (2.40)$$

In each of the five-matrix representations given in Eqs.(2.21) and (2.35), as well as in the conversions yet to come, there is a distinct relationship between matrices $\mathbf{B}_0$ and $\mathbf{B}_1$. It can be verified from Eqs.(2.23) and (2.35), respectively, that this relationship is given be

$$\mathbf{B}_1 = \mathbf{P}\,\mathbf{B}_0 \qquad (2.41)$$

where the $n\times n$ matrix $\mathbf{P}$ is expressible in each case by

$$\mathbf{P} = \mathbf{F}(\mathbf{E}-\mathbf{F})^{-1} \quad \text{and} \quad \mathbf{P} = \mathbf{I}_n \qquad (2.42)$$

respectively. Using Eq.(2.41) and the identity

$$(z\mathbf{I}-\mathbf{A})^{-1}z = \mathbf{I}+(z\mathbf{I}-\mathbf{A})^{-1}\mathbf{A} \qquad (2.43)$$

Eq.(2.39) may be written as

$$\mathbf{C}(z\mathbf{I}-\mathbf{A})^{-1}[(\mathbf{I}+\mathbf{AP})\mathbf{B}_0-\mathbf{B}_e]+(\mathbf{CPB}_0+\mathbf{D}-\mathbf{D}_e) = \mathbf{0} \qquad (2.44)$$

Since Eq.(2.44) should be satisfied for all z, it reduces to

$$\begin{aligned}\mathbf{C}(z\mathbf{I}-\mathbf{A})^{-1}[(\mathbf{I}+\mathbf{AP})\mathbf{B}_0-\mathbf{B}_e] &= \mathbf{0}\\ \mathbf{D}_e &= \mathbf{CPB}_0+\mathbf{D}\end{aligned} \qquad (2.45)$$

We now introduce the following notation

$$\mathbf{C}(z\mathbf{I}-\mathbf{A})^{-1} = \frac{V(z)}{d(z)} \qquad (2.46)$$

where $$V(z) = \mathbf{C}\,\mathrm{adj}[z\mathbf{I}-\mathbf{A}] \quad , \quad d(z) = \det[z\mathbf{I}-\mathbf{A}] \qquad (2.47)$$

The $p\times n$ polynomial matrix $V(z) = \{v_{ij}(z)\}$, consisting generally of $(n\text{-}1)^{st}$ order polynomials, can also be represented as a matrix polynomial with real-number $p\times n$ matrices, i.e.

$$V(z) = \sum_{i=0}^{n-1} V_i z^i \tag{2.48}$$

Using Eqs.(2.47) and (2.48) and defining the arrays

$$\mathbf{V} = \begin{bmatrix} V_0 \\ V_1 \\ \vdots \\ V_{n-1} \end{bmatrix}, \qquad I(z) = \begin{bmatrix} I_p & zI_p & \cdots & z^{n-1}I_p \end{bmatrix} \tag{2.49}$$

Eq.(2.45) becomes

$$\mathbf{V}\,[(\mathbf{I} + \mathbf{AP})\mathbf{B}_0 - \mathbf{B}_e] = \mathbf{0} \tag{2.50}$$

It is easily shown that if the pair {**A**, **C**} is observable, **V** is a full (column) rank matrix and that the unknown matrix $\mathbf{B}_e$ becomes

$$\mathbf{B}_e = (\mathbf{I} + \mathbf{AP})\mathbf{B}_0 \tag{2.51}$$

However, if {**A**, **C**} is not an observable pair, the general solution to Eq.(2.50) may be written as

$$\mathbf{B}_e = (\mathbf{I} + \mathbf{AP})\mathbf{B}_0 + \mathbf{NT} \tag{2.52}$$

where **N** is an $n \times h$ "null space" matrix (h is the *nullity* or dimension of the null space of **V**) satisfying that

$$\mathbf{VN} = \mathbf{0} \tag{2.53}$$

and **T** is an arbitrary $h \times m$ matrix, which, if desired, may be chosen to be a zero matrix. If, however, **T** is selected as

$$\mathbf{T} = -\mathbf{N}^{+}(\mathbf{I} + \mathbf{AP})\mathbf{B}_0 \tag{2.54}$$

where

$$\mathbf{N}^{+} = (\mathbf{N}^T\mathbf{N})^{-1}\mathbf{N}^T$$

is the *pseudo-inverse* of **N**, then $\mathbf{B}_e$ may be written as

$$\mathbf{B}_e = (\mathbf{I} - \mathbf{NN}^{+})(\mathbf{I} + \mathbf{AP})\mathbf{B}_0 \tag{2.55}$$

which represents the minimum norm solution for $\mathbf{B}_e$. It should be mentioned that even when the pair {**A**, **C**} is unobservable, the matrix $\mathbf{B}_e$ given in Eq.(2.51) satisfies the transfer function matrix identity Eq.(2.39).

The result of the previous development is the following computational procedure for converting from a five-matrix model to a four-matrix model.

Algorithm *R5R4*

Syntax: $\mathbf{A}, \mathbf{B}_0, \mathbf{P}, \mathbf{C}, \mathbf{D}$ (*R5R4*) $\rightarrow \mathbf{B}_e, \mathbf{D}_e$

Purpose: Transformation from a five-matrix model to a (standard) four-matrix model.

Input/Output Variables:

$\mathbf{A}$	$= (n \times n)$ system matrix of the five-matrix model
$\mathbf{B}_0$	$= (n \times m)$ first-input matrix of the five-matrix model
$\mathbf{P}$	$= (n \times n)$ transform matrix between $\mathbf{B}_0$ and $\mathbf{B}_1$ (see Eq.(2.41))
$\mathbf{C}$	$= (p \times n)$ output matrix of the five-matrix model
$\mathbf{D}$	$= (p \times m)$ feedthrough matrix of the five-matrix model
$\mathbf{B}_e$	$= (n \times m)$ input matrix of the four-matrix model
$\mathbf{D}_e$	$= (p \times m)$ feedthrough matrix of the four-matrix model

Description: From a given *five-matrix* model, e.g. Eq.(2.21) or Eq.(2.35) and Eq.(2.41), a standard *four-matrix* model with equivalent transfer matrix is generated using Eq.(2.45).

Algorithm:

1. Define the matrices $\mathbf{A}$, $\mathbf{B}_0$, $\mathbf{P}$, $\mathbf{C}$ and $\mathbf{D}$.
2. If the pair $\{\mathbf{A}, \mathbf{C}\}$ is observable, calculate the unknown matrices $\mathbf{B}_e$ and $\mathbf{D}_e$ from Eqs.(2.51) and (2.45), respectively.
3. If $\{\mathbf{A}, \mathbf{C}\}$ is an unobservable pair, Eq.(2.51) may be substituted for Eq.(2.55) which requires the evaluation of the polynomial matrix $V(z)$, Eq.(2.47), building the $(pn \times n)$ matrix $\mathbf{V}$, Eq.(2.49), and calculation of the null space matrix $\mathbf{N}$, Eq.(2.53).

The listing of Algorithm *R5R4*, implemented in *L-A-S*, is given in Appendix C.

Algorithm *R4R5*

Transformation from a standard four-matrix representation to an equivalent five-matrix representation is the reverse process of the previous Algorithm, *R5R4*, and is used primarily as an intermediate step in the subsequent continualization procedure of Algorithm *RI-C-D*. The relation indicated in Eqs.(2.41) and (2.42) will be used in this procedure to ensure that a unique four-matrix state space representation is obtained. Thus, assuming Eqs.(2.40) and (2.41), only $\mathbf{B}_0$ and $\mathbf{D}$

are unknown. Following the same line of reasoning as in the previous algorithm, if the pair $\{\mathbf{A}, \mathbf{C}\}$ is observable, then from Eq.(2.51) we obtain

$$\mathbf{B}_0 = (\mathbf{I} + \mathbf{A}\mathbf{P})^{-1}\,\mathbf{B}_r \tag{2.56}$$

while from Eq.(2.45)

$$\mathbf{D} = \mathbf{D}_r - \mathbf{C}\mathbf{P}\mathbf{B}_0 \tag{2.57}$$

If $\{\mathbf{A}, \mathbf{C}\}$ is an unobservable pair, the minimum norm solution for $\mathbf{B}_0$ can be obtained in a manner similar to the development of Eq.(2.55) from

$$\mathbf{B}_0 = (\mathbf{I} - \mathbf{N}\mathbf{N}^+)(\mathbf{I} + \mathbf{A}\mathbf{P})^{-1}\mathbf{B}_r \tag{2.58}$$

where $\mathbf{N}$ was defined above, see Eq.(2.53).

Thus, from the transfer matrix equivalency Eq.(2.39) we are led to the following algorithm for converting from a standard four-matrix model to an equivalent five-matrix model.

Algorithm *R4R5*

Syntax: $\mathbf{A}, \mathbf{B}_r, \mathbf{C}, \mathbf{D}_r, \mathbf{P}$ (*R4R5*) $\rightarrow \mathbf{B}_0, \mathbf{D}$

Purpose: Transformation from a (standard) four-matrix model to a five-matrix model.

Input/Output Arguments:

$\mathbf{A}$	$= (n \times n)$ system matrix of the four-matrix model
$\mathbf{B}_r$	$= (n \times m)$ input matrix of the four-matrix model
$\mathbf{C}$	$= (p \times n)$ output matrix of the four-matrix model
$\mathbf{D}_r$	$= (p \times m)$ feedthrough matrix of the four-matrix model
$\mathbf{P}$	$= (n \times n)$ transform matrix between $\mathbf{B}_0$ and $\mathbf{B}_1$ (see Eq.(2.41))
$\mathbf{B}_0$	$= (n \times m)$ first input matrix of the five-matrix model
$\mathbf{D}$	$= (p \times m)$ feedthrough matrix of the five-matrix model

Description: From a given *four-matrix* model a *five-matrix* model is generated using the transfer matrix equivalency of Eq.(2.39).

Algorithm:

1. Define the matrices $\mathbf{A}$, $\mathbf{B}_r$, $\mathbf{C}$, $\mathbf{D}_r$ and $\mathbf{P}$.
2. If the pair $\{\mathbf{A}, \mathbf{C}\}$ is observable, calculate the unknown martices $\mathbf{B}_0$ and $\mathbf{D}$ from Eqs.(2.56) and (2.57), respectively.
3. If $\{\mathbf{A}, \mathbf{C}\}$ is an unobservable pair, Eq.(2.58) may be used in place of Eq.(2.56). This necessitates the evaluation of the polynomial matrix

$V(z)$, Eq.(2.47), building $\mathbf{V}$, Eq.(2.49), and calculation of the matrix $\mathbf{N}$ in Eq.(2.53).

The listing of Algorithm *R4R5*, implemented using *L-A-S*, can be found in Appendix C.

2.3 Continualization Procedures

The reverse process of converting from a D-T model to an *equivalent* C-T model will now be considered, i.e. converting from the model in Eq.(2.4) to the model in Eq.(2.2), $R_d \to R_c$ in the SI case, or from Eq.(2.21) to Eq.(2.2), $R_{df} \to R_c$ in the RI sense and from Eq.(2.35) to Eq.(2.2) in the BT sense. Of course, by itself R_d has no information regarding the signal values between samples so that model conversion in this direction should be taken in the context of some prior knowledge regarding the type of inputs used.

2.3.1 SI to Continuous-Time Model

The algorithms for continualization require *logarithmic* operations instead of matrix exponentiation. When $(\mathbf{A}_d - \mathbf{I})$ or $\mathbf{A}_c$ is invertible, it is easily concluded that the matrices of R_c in Eq.(2.2) can be found from Eqs.(2.5) and (2.7) by:

$$\mathbf{A}_c = \frac{1}{T}\ln(\mathbf{A}_d)\,, \qquad \mathbf{B}_c = (\mathbf{A}_d - \mathbf{I})^{-1}\mathbf{A}_c\mathbf{B}_d \tag{2.59}$$

with the understanding that $\mathbf{C}_c = \mathbf{C}_d$ and $\mathbf{D}_c = \mathbf{D}_d$ as before, thus completing the continuous-time model in Eq.(2.2). To begin the development, consider the Taylor series expansion for the function $\ln(x)$ in the neighborhood of $x = 1$ which leads to

$$\mathbf{A}_c = \frac{1}{T}\sum_{i=1}^{\infty}\frac{(\mathbf{A}_d - \mathbf{I})^i}{i}(-1)^{(i+1)} \tag{2.60}$$

The problem of using a truncated version of Eq.(2.60) is that for matrices $\mathbf{A}_d$ with

$$\lambda_m \triangleq |\lambda_{\max}| > 0.5 \tag{2.61}$$

where $\lambda_{\max}$ is the maximum magnitude eigenvalue of $(\mathbf{A}_d - \mathbf{I})$, the series may require large N, leading to considerable round-off errors if it converges at all. As will be seen, the present algorithm will resolve this problem by using the following basic property of the logarithm function.

$$\ln(x) = r\ln\left[(x)^{1/r}\right] = r\sum_{i=1}^{\infty}\frac{(x^{1/r}-1)^i}{i}(-1)^{(i+1)} = -r\sum_{i=1}^{\infty}\frac{(1-x^{1/r})^i}{i} \tag{2.62}$$

With this approach the truncated series for calculation becomes

$$\mathbf{A}_c = -\frac{r}{T}\sum_{i=1}^{N}\frac{(\mathbf{I}-\mathbf{A}_d^{1/r})^i}{i} \tag{2.63}$$

where the integer j satisfies that

$$|\lambda(\mathbf{A}_d^{1/r}-\mathbf{I})|_{max} < \lambda_m\,, \quad \text{with} \quad r = 2^j \tag{2.64}$$

An algorithm which calculates $\mathbf{A}_c$ according to Eq.(2.63) is referred to as Algorithm *LNM*. It has been experimentally verified that the accuracy of using Eq.(2.63) is satisfactory even for matrices $\mathbf{A}_d$ where some eigenvalues of $\mathbf{L} = \mathbf{A}_d - \mathbf{I}$ have magnitude greater than one. From the previous development the following algorithm is formalized.

Algorithm *LNM*

Syntax: T, $\mathbf{A}_d$, λ_m, N, $(LNM) \Rightarrow \mathbf{A}_c$

Purpose: The calculation of the natural logarithm of an $(n\times n)$ matrix $\mathbf{A}_d$.

Input/Output Arguments:

T	= sampling interval used in the discrete-time model
$\mathbf{A}_d$	= $(n\times n)$ system matrix of the discrete-time model
λ_m	= scaling parameter, see Eq.(2.64)
N	= series truncation parameter
$\mathbf{A}_c$	= $(n\times n)$ system matrix of the continuous-time model

Description: A continuous-time equivalent system matrix is constructed from a corresponding discrete-time system matrix.

Algorithm:

```
1. Define the matrix $\mathbf{A}_d$, scalars $T$ and $\lambda_m$ and integer $N$ — (suggested
   values for $N$ and $\lambda_m$ are $N \geq 36$ and $\lambda_m \leq 0.25$)
2. Set $0 \Rightarrow j$ and $\mathbf{A}_d \Rightarrow \mathbf{A}_j$
3. Set $\mathbf{I} - \mathbf{A}_j \Rightarrow \mathbf{L}$
4. If $|\lambda(\mathbf{L})|_{max} < \lambda_m$, go to 6; else, go to 5
5. Set $j+1 \Rightarrow j$ ;  $(\mathbf{A}_j)^{1/2} \Rightarrow \mathbf{A}_j$  and go to 3
6. Set $2^j \Rightarrow r$  and calculate $\mathbf{A}_c$ using $\mathbf{A}_j \Rightarrow \mathbf{A}_d^{1/r}$ in Eq.(2.63)
```

The square root of the matrix $\mathbf{A}_j$ is calculated by Algorithm *SQM*, described later. Calculation of $\mathbf{A}_c$ in Step 6 is accomplished using Algorithms *POLR* and *POM*, mentioned earlier. Calculation of the coefficients $f_i = 1/i$, $i=[1,N]$, required by

POLR is done by Algorithm *FLN*. The listing of Algorithm *LNM*, implemented using *L-A-S*, is given in Appendix C.

Having determined $\mathbf{A}_c$, the remaining matrices in the SI C-T equivalent state space model of Eq.(2.2) could be calculated using Eq.(2.59) if $\mathbf{A}_c$ is nonsingular. If, however, $\mathbf{A}_c$ is singular, then the matrix $\mathbf{E}$, appearing in Eq.(2.9) should be calculated using the procedure given in Algorithm *SI-C-D*. It follows that $\mathbf{C}_c = \mathbf{C}_d$, $\mathbf{D}_c = \mathbf{D}_d$ and

$$\mathbf{B}_c = \frac{1}{T}\mathbf{E}^{-1}\mathbf{B}_d \tag{2.65}$$

In the spirit of algorithm formulation and algorithm "naming," Eqs.(2.59) and (2.65) could be symbolically represented by Algorithm *SI-D-C*, i.e.

$$\mathbf{A}_d,\ \mathbf{B}_d,\ T\ (SI\text{-}D\text{-}C) \Rightarrow \mathbf{A}_c,\ \mathbf{B}_c$$

For reasons explained later, there is no direct *L-A-S* "counter-part" to Algorithm *SI-D-C*. However, as will be shown in Section 2.5, there is an *L-A-S* algorithm which, among other tasks, performs the task of Algorithm *SI-D-C*.

Algorithm *LNMj*

An alternate algorithm, referred to as *LNMj*, applicable for calculating $\mathbf{A}_c$, given by (2.59), in the *specific* case when the matrix $\mathbf{A}_d$ is "diagonalizable," is given below. It is worth mentioning that this algorithm is "in a way" equivalent to Algorithm *EATj*, mentioned in Chapter 1, Section 1.3.3.

Syntax: $T,\ \mathbf{A}_d\ (LNMj) \Rightarrow \mathbf{A}_c$

For input/output arguments see Algorithm *LNM*.

Algorithm:

```
1. Define the matrix $\mathbf{A}_d$ and the scalar $T$
2. Set $\mathbf{A}_d\ (JFR) \Rightarrow \mathbf{M}$
3. Set $\mathbf{A}_d\ (EGV) \Rightarrow \mathbf{egd} = \{\lambda_{di}\}$
4. Set diag{ $\ln(\lambda_{di})$ } $\Rightarrow \mathbf{LnJf}$
5. Set $\mathbf{M}\ \mathbf{LnJf}\ \mathbf{M}^{-1}/T \Rightarrow \mathbf{A}_c$
```

The listing of Algorithm *LNMj*, implemented using *L-A-S*, is given in Appendix C. Note that the steps in this algorithm are similar to the steps of Algorithm *EATj*, discussed in Section 1.3.3. The only difference is that in Step 4 the array **LnJf** contains in its main diagonal the natural log of the eigenvalues λ_{di}, i.e.

$$\ln(\lambda_{di})$$

while in Algorithm *EAT*), the array **ExJf** contains the terms $\exp(\lambda_i T)$.

Algorithm *SQM*

The square root of the matrix $\mathbf{A}_y$, required in Step 5 of Algorithm *LNM*, could be calculated by Algorithm *SQM* given below. This algorithm is based on the standard recursive procedure:

$$x_{i+1} = 0.5(x_i + \frac{b}{x_i}) \tag{2.66}$$

used to determine the square root $x = (b)^{1/2}$ of a positive scalar b.

Algorithm *SQM*

Syntax: $\mathbf{A}, \epsilon$ (*SQM*) $\Rightarrow$ $\mathbf{X}$

Purpose: To calculate the square root of a positive-definite matrix.

Input/Output Arguments:

$\mathbf{A}$	= Given square positive definite matrix
ϵ	= Small scalar parameter used to terminate the recursion
$\mathbf{X}$	= The square-root matrix of $\mathbf{A}$

Description: Determination of the square root of an $n \times n$ matrix $\mathbf{A}$, $\mathbf{X} = (\mathbf{A})^{1/2}$.

Algorithm:

```
1. Define the matrix $\mathbf{A}$ and a small scalar parameter $\epsilon << 1$
2. Set $\mathbf{X}_0 = \mathbf{I}$ and $i = 0$
3. Set $i = i+1$ and $\mathbf{X}_{i+1} = 0.5\ (\mathbf{X}_i + \mathbf{A}\ \mathbf{X}_i^{-1})$
4. If $\|\ \mathbf{X}_{i+1} - \mathbf{X}_i\ \| > \epsilon$, go to 3; else, stop
```

The listing of Algorithm *SQM*, implemented using *L-A-S*, can be found in Appendix C.

2.3.2 RI to Continuous-Time Model

It is easily determined that the C-T model in Eq.(2.2) can be obtained from the five-matrix model in Eq.(2.21) by using Algorithm *LNM* to calculate $\mathbf{A}_c$, and from the availability of **F** in Eq.(2.31), i.e. Algorithm *RI-C-D*, solving Eq.(2.33) to get

$$\mathbf{B}_c = \frac{1}{T}\mathbf{F}^{-1}\mathbf{B}_{d1} = \frac{1}{T}(\mathbf{E}-\mathbf{F})^{-1}\mathbf{B}_{d0} \tag{2.67}$$

Note that from Eq.(2.67), or from Eq.(2.42)

$$\mathbf{B}_{d1} = \mathbf{P}\mathbf{B}_{d0}\,, \quad \text{where} \quad \mathbf{P} = \mathbf{F}(\mathbf{E}-\mathbf{F})^{-1} \tag{2.68}$$

The required five-matrix D-T model of Eq.(2.21) can be obtained as an initial step from a standard four-matrix D-T model in Eq.(2.36) by applying Algorithm *R4R5* presented earlier. Similarly, as in the case of Algorithm *SI-D-C*, Section 2.3.1, Eq.(2.59), together with Eqs.(2.67) and (2.68), could symbolically be represented by

$$\mathbf{A}_d,\, \mathbf{B}_{d0},\, \mathbf{B}_{d1},\, T\ (RI\text{-}D\text{-}C) \rightarrow \mathbf{A}_c,\, \mathbf{B}_c$$

As has already been mentioned, there is no *L-A-S* algorithm which directly corresponds to *RI-D-C*. This will be made clear in Section 2.5.

2.3.3 Bilinear to Continuous-Time Model

The C-T model of Eq.(2.2), which corresponds to the bilinear transformed model specified in Eq.(2.35), can be obtained by a direct substitution of

$$z = \frac{a+s}{a-s}\,, \quad \text{where} \quad a = \frac{2}{T} \tag{2.69}$$

into the z-domain transfer function, thereby providing an s-domain transfer function from which R_c could be derived. Specifically, taking the z-transform of Eq.(2.4), introducing Eq.(2.69) and converting back to the time domain:

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{A}_c\mathbf{x}(t) + \mathbf{B}_{c0}\mathbf{u}(t) + \mathbf{B}_{c1}\dot{\mathbf{u}}(t) \\ \mathbf{y}(t) &= \mathbf{C}_c\mathbf{x}(t) + \mathbf{D}_c\mathbf{u}(t) \end{aligned} \tag{2.70}$$

where (with $a = 2/T$)

$$\begin{aligned} \mathbf{A}_c &= a(\mathbf{A}_d+\mathbf{I})^{-1}(\mathbf{A}_d-\mathbf{I})\,, \quad \mathbf{B}_{c0} = a(\mathbf{A}_d+\mathbf{I})^{-1}\mathbf{B}_d \\ \mathbf{B}_{c1} &= -(\mathbf{A}_d+\mathbf{I})^{-1}\mathbf{B}_d\,, \quad \mathbf{C}_c = \mathbf{C}_d\,, \quad \text{and} \quad \mathbf{D}_c = \mathbf{D}_d \end{aligned} \tag{2.71}$$

As was discussed previously in terms of the five-term model of Eq.(2.21), if a four-term C-T model is required, Algorithm *R5R4* can be applied to Eq.(2.70) to obtain an equivalent standard model of the form in Eq.(2.2). Note that in this case Eq.(2.41) holds with

$$\mathbf{P} = -\frac{T}{2}\mathbf{I}_n \tag{2.72}$$

Similarly, as with Eqs.(2.35) and Algorithm *BT-C-D*, Eqs.(2.71) may be symbolically represented by Algorithm *BT-D-C*, i.e.

$$\mathbf{A}_d,\ \mathbf{B}_d,\ T\ (BT\text{-}C\text{-}D) \rightarrow \mathbf{A}_c,\ \mathbf{B}_{c0},\ \mathbf{B}_{c1},\ \mathbf{P}$$

However, its *L-A-S* implementation requires:

$$\mathbf{A}_d,\ \mathbf{B}_d,\ T,\ Icdc\ (BCDC) \rightarrow \mathbf{A}_c,\ \mathbf{B}_{c0},\ \mathbf{B}_{c1},\ \mathbf{P}$$

where now, as opposed to the case of Algorithm *BT-D-C*, the algorithm flag *Icdc* should have the value *Icdc* = 2. The listing of Algorithm *BCDC*, implemented using *L-A-S*, and performing the tasks of both *BT-D-C* and *BT-C-D*, is given in Appendix C. For more details see also Section 2.5.

This completes the three methods of continualization. The reader now has the algorithmic tools to discretize a C-T system using piecewise constant inputs (SI), piecewise linear inputs (RI) and the bilinear transformation (BT), as well as to perform the inverse operation of continualization corresponding to each of these methods. In converting a physically sampled C-T system to a D-T model the SI method most closely approximates the common digital-to-analog device operation. However, any one of the three techniques may be used when it is desired to mimic a linear C-T process with a D-T model. Such a situation might arise, for instance, for preprocessing data in a computer by developing a filter algorithm from a known frequency filter in the C-T domain. In this instance, it would be prudent to compare the frequency responses of the D-T models with the desired frequency response. Yet another area of utility is system identification. The RI method can be an effective approach to identifying a system from discrete data because of the additional accuracy inherent in the method. In the remainder of this chapter we present several examples which illustrate the convergence and robustness of both the discretization and the continualization procedures.

2.4 Examples

Three examples are presented in this section. They have been selected to illustrate the computational accuracy that can be achieved using the exponential and the logarithmic matrix calculations discussed previously. The first example demonstrates convergence rates when calculating $\mathbf{A}_d$ from a given 5 × 5 singular matrix $\mathbf{A}_c$, followed by a similar development in the second example in calculating $\mathbf{A}_c$ given $\mathbf{A}_d$. The third example illustrates the remaining discretization and continualization procedures mentioned in the chapter. The calculations were performed using the algorithms discussed earlier.

Example 1. For this example the matrix $\mathbf{A}_c$ is given by

$$\mathbf{A}_c = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ -4 & -4 & -3 & 1 & 4 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & .5 & 0 & 0 & 0 \end{bmatrix} \tag{2.73}$$

The eigenvalues of $\mathbf{A}_c$ are $\lambda(\mathbf{A}_c) = \{0,\ -1,\ -1,\ -1+j1,\ -1-j1\}$ (2.74)

Note that $\mathbf{A}_c$ is singular and has multiple eigenvalues. In addition, the Jordan form, $\mathbf{A}_J$, corresponding to $\mathbf{A}_c$ is not diagonal. The selection of this matrix was motivated by the fact that some widely used software packages are not capable of calcuating either the Jordan form, or the natural logarithm, of non-diagonalizable matrices. The well known packages *MATLAB* and *MATHEMATICA* are examples of this deficiency. It is suggested that the reader repeat the calculations in these examples with another package at his or her disposal. The desired sampling interval for the discretization is $T = 2$ sec.; and the norm of $\mathbf{A}_cT$ is calculated to be 15.65. Eqs.(2.5) and (2.10) to (2.12) combined provide the following truncated summation, which is similar to Algorithm *SI-C-D* for calculating the exponential matrix.

$$\mathbf{A}_d = \left[\sum_{i=0}^{N} \frac{(\mathbf{A}_cT/r)^i}{i!} \right]^r \tag{2.75}$$

As in *SI-C-D*, $r = 2^j$ where j is given in Eq.(2.11). Both the truncation number N and the scaling parameter j are of key interest to this development. To emphasize the dependence of our calculated matrix $\mathbf{A}_d$ on these parameters, we will use the notation $\mathbf{A}_d(N,j)$. Results will be presented for the following 36 parameter combinations:

$$j = 0,\ 1,\ 2,\ 3,\ 4,\ 5$$

and
$$N = 16,\ 14,\ 12,\ 10,\ 8,\ 6 \tag{2.76}$$

Each $\mathbf{A}_d$ is compared to the "exact" matrix $\mathbf{A}_{de}$ given by

$$\mathbf{A}_{de} = \begin{bmatrix} -.99900\,E{-1} & .50637\,E{+0} & .19165\,E{+0} & .14761\,E{+0} & .10999\,E{+1} \\ -.76662\,E{+0} & -.31657\,E{+0} & -.68595\,E{-1} & .44044\,E{-1} & .76662\,E{+0} \\ .27438\,E{+0} & -.10893\,E{+0} & -.11078\,E{+0} & -.11264\,E{+0} & -.27438\,E{+0} \\ .00000\,E{+0} & .00000\,E{+0} & .00000\,E{+0} & .13534\,E{+0} & .00000\,E{+0} \\ -.54995\,E{+0} & .25318\,E{+0} & .95827\,E{-1} & .73805\,E{-1} & .15500\,E{+1} \end{bmatrix}$$

The "exact" matrix $\mathbf{A}_{de}$ above was calculated by transforming $\mathbf{A}_c$ *manually* into its Jordan canonical form $\mathbf{A}_J$ and then using

$$e^{\mathbf{A}_c T} = \mathbf{Q} e^{\mathbf{A}_J T} \mathbf{Q}^{-1} \tag{2.77}$$

It may be verified that for $\mathbf{A}_c$ given in Eq.(2.73),

$$\mathbf{A}_J = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{Q} = \begin{bmatrix} 2 & 0 & 2 & 0 & 1 \\ -2 & 2 & -2 & 2 & 0 \\ 0 & -4 & 2 & -4 & 0 \\ 0 & 0 & 0 & 2 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix}$$

and for $T = 2$ seconds

$$e^{\mathbf{A}_J T} = \begin{bmatrix} e^{-T}\cos T & e^{-T}\sin T & 0 & 0 & 0 \\ -e^{-T}\sin T & e^{-T}\cos T & 0 & 0 & 0 \\ 0 & 0 & e^{-T} & Te^{-T} & 0 \\ 0 & 0 & 0 & e^{-T} & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} .056 & .123 & 0 & 0 & 0 \\ -.123 & .056 & 0 & 0 & 0 \\ 0 & 0 & .135 & .271 & 0 \\ 0 & 0 & 0 & .135 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Note that $\mathbf{A}_J$ is given in the *real number* Jordan form. See Section B.4.

The $\log_{10}$ of the norm of the error matrix $\mathbf{E}_d = \mathbf{A}_{de} - \mathbf{A}_d(N,j)$ is tabulated for each combination in Eq.(2.76) in Table 2.1 below. Since the particular norm used is not critical, the *Frobenius* norm, defined as the square root of the sum of squares of all matrix elements, is used. From Table 2.1 and corresponding Fig. 2.1 it can be seen that $N = 16$ terms is sufficient for $\mathbf{A}_d$ in Eq.(2.75) even for matrices $\mathbf{A}_c T$ with relatively high norms. And, as we can see from Table 2.1, N may be chosen as low as $N = 6$ with judicious choice of the scaling parameter j.

TABLE 2.1.
$\text{Log}_{10}(\|\mathbf{E}_d\|)$ vs. Truncation Number N and Scaling Parameter j

N	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
16	-3.783	-10.069	-14.422	-14.448	-14.551	-14.551
14	-3.783	-8.079	-12.114	-14.613	-14.551	-14.551
12	-2.305	-6.035	-9.631	-13.100	-14.551	-14.551
10	-1.241	-4.337	-7.157	-9.937	-12.684	-14.440
8	-0.101	-2.601	-4.965	-7.198	-9.362	-11.496
6	0.637	-1.155	-2.780	-4.361	-5.908	-7.435

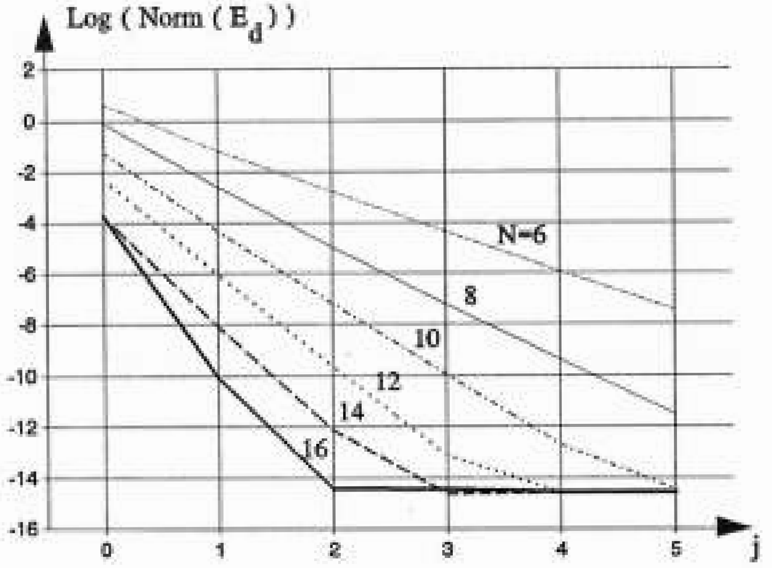


FIGURE 2.1 Log (Norm($\mathbf{E}_d$)) vs. Computation Parameters

Example 2. In this example the matrix $\mathbf{A}_d$ is taken to be $\mathbf{A}_{d\epsilon}$ given above. The calculation used to determine $\mathbf{A}_c$ is the truncated series in Eq.(2.63). We note that the eigenvalues of $\mathbf{L} = \mathbf{I} - \mathbf{A}_d$, influencing the convergence of the series, can exceed unit magnitude. In particular, $\lambda(\mathbf{L})$ are:

$$\lambda(\mathbf{I} - \mathbf{A}_d) = \{0, -0.86, -0.86, -1.06 + j0.12, -1.06 - j0.12\} \tag{2.78}$$

To illustrate the convergence properties, the power series Eq.(2.63) was evaluated for all combinations of the parameters N and j given by

$$j = 0, 1, 2, 3, 4, 5$$

and
$$N = 35, 30, 25, 20, 15, 10 \tag{2.79}$$

As in Example 1, the error matrix is defined to be

$$\mathbf{E}_c = \mathbf{A}_c - \mathbf{A}_c(N, j) \tag{2.80}$$

where the explicit notation $\mathbf{A}_c(N,j)$ is used to emphasize the dependence of the calculated matrix on the computation parameters N and j. The log_{10} of the norm of the matrix $\mathbf{E}_c$ is tabulated for the combinations indicated in Eqs.(2.79) in Table 2.2. As in Example 1 the Frobenius norm is used for convenience.

We see from the results in Table 2.2 that the series Eq.(2.63) can be truncated as high as $N = 35$ even when the maximum eigenvalue of $\mathbf{L}$ is greater than unity. It is also noted that the truncation may be as low as $N = 10$ provided that the scaling parameter j is appropriately selected. In practice, N can be fixed at a nominal value, say 20, and j can be varied over 3 or 4 values to ensure good convergence to the desired matrix. This is true whether the problem requires *discretization* or *continualization*.

As before, the information of Table 2.2 is given in graphical form in Fig. 2.2 to illustrate the convergence of the series Eq.(2.63).

TABLE 2.2.
$\text{Log}_{10}(\|\mathbf{E}_r\|)$ vs. Truncation Number N
and Scaling Parameter j

N	$j=0$	$j=1$	$j=2$	$j=3$	$j=4$	$j=5$
35	1.015	-2.565	-9.252	-10.123	-10.123	-10.123
30	0.915	-1.998	-7.845	-10.123	-10.123	-10.123
25	0.748	-1.753	-6.403	-10.123	-10.123	-10.123
20	0.678	-1.212	-4.962	-9.758	-10.123	-10.123
15	0.784	-0.779	-3.519	-7.194	-10.095	-10.123
10	0.896	-0.374	-2.081	-4.490	-7.015	-9.594

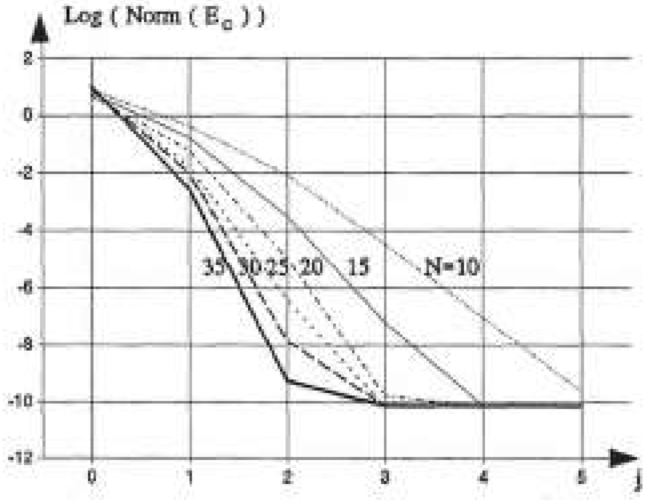


FIGURE 2.2 Log (Norm($\mathbf{E}_0$)) vs. Computation Parameters

Example 3. In this example the following C-T state space representation is considered:

$$R_c = \begin{bmatrix} \mathbf{A}_c & \mathbf{B}_c \\ \mathbf{C}_c & \mathbf{D}_c \end{bmatrix} = \left[\begin{array}{ccccc|ccc} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ -4 & -4 & -3 & 1 & 4 & 0 & 0 & 1 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 \\ 0 & 0.5 & 0 & 0 & 0 & 1 & 0 & 0 \\ \hline 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \qquad (2.81)$$

Note that $\mathbf{A}_c$ is the same matrix used in Example 1, Eq.(2.73). The input signal $\mathbf{u}(t)$ is specified over the interval $0 < t < 8T$, with $T = 2$ seconds by the components defined in Table 2.3.

TABLE 2.3. Input Signal for Example 3.

Time Interval	$u_1(t)$	$u_2(t)$	$u_3(t)$
$0 \le t \le 4T$	$t/(4T)$	$\sin(\pi t/(4T))$	$0.5\ \{\cos(\pi t/(4T)) - 1\}$
$4T \le t \le 8T$	$2 - t/(4T)$	0	$t/(4T) - 2$

Discretization: Using the sampling interval $T = 2$, the representation R_c is discretized into the following equivalents, each represented in the partitioned *system matrix* form of the given state space model in Eq.(2.81):

(a) $R_{ds} = \{\ \mathbf{A}_{ds}\ ,\ \mathbf{B}_{ds}\ ,\ \mathbf{C}_{ds}\ ,\ \mathbf{D}_{ds}\ \}$, D-T step-invariant (SI) equivalent
(b) $R_{dr} = \{\ \mathbf{A}_{dr}\ ,\ \mathbf{B}_{dr}\ ,\ \mathbf{C}_{dr}\ ,\ \mathbf{D}_{dr}\ \}$, D-T ramp-invariant (RI) equivalent
(c) $R_{db} = \{\ \mathbf{A}_{db}\ ,\ \mathbf{B}_{db}\ ,\ \mathbf{C}_{db}\ ,\ \mathbf{D}_{db}\ \}$, D-T bilinear transform (BT) equivalent

As was pointed out in Sec. 2.2, in the cases of (b) and (c) the five-matrix D-T models of Eqs.(2.21) and (2.35) were first calculated. This was followed by a conversion to the standard four-matrix D-T model using Algorithm *R5R4*. These three results are given below:

$$R_{ds} = \begin{bmatrix} \mathbf{A}_{ds} & \mathbf{B}_{ds} \\ \mathbf{C}_{ds} & \mathbf{D}_{ds} \end{bmatrix} = \left[\begin{array}{ccccc|ccc} -.100 & .506 & .192 & .148 & 1.100 & .787 & .127 & .275 \\ -.767 & -.317 & -.069 & .044 & .767 & 1.100 & .148 & .192 \\ .274 & -.109 & -.111 & -.113 & -.274 & .767 & .044 & -.069 \\ 0 & 0 & 0 & .135 & 0 & 0 & .865 & 0 \\ -.550 & .253 & .096 & .074 & 1.550 & 2.394 & .064 & .137 \\ \hline 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

$$R_{dr} = \begin{bmatrix} \mathbf{A}_{dr} & \mathbf{B}_{dr} \\ \mathbf{C}_{dr} & \mathbf{D}_{dr} \end{bmatrix} = \left[\begin{array}{ccccc|ccc} -.100 & .506 & .192 & .148 & 1.100 & 2.084 & .238 & .309 \\ -.767 & -.317 & -.069 & .044 & .767 & 1.235 & .070 & -.034 \\ .274 & -.109 & -.111 & -.113 & -.274 & -.135 & -.104 & -.177 \\ 0 & 0 & 0 & .135 & 0 & 0 & .374 & 0 \\ -.550 & .253 & .096 & .074 & 1.550 & 3.042 & .119 & .154 \\ \hline 1 & 0 & 0 & 0 & 0 & 1.196 & .035 & .098 \\ 0 & 1 & 0 & 0 & 0 & .394 & .064 & .137 \end{array}\right]$$

$$R_{db} = \begin{bmatrix} \mathbf{A}_{db} & \mathbf{B}_{db} \\ \mathbf{C}_{db} & \mathbf{D}_{db} \end{bmatrix} = \left[\begin{array}{ccccc|ccc} .200 & .800 & .200 & .100 & .800 & 1.840 & .180 & .260 \\ -.800 & -.200 & .200 & .100 & .800 & 1.040 & .080 & .060 \\ -.800 & -1.200 & -.800 & .100 & .800 & .240 & -.020 & -.140 \\ 0 & 0 & 0 & 0 & 0 & 0 & .500 & 0 \\ -.400 & .400 & .100 & .050 & 1.400 & 2.920 & .090 & .130 \\ \hline 1 & 0 & 0 & 0 & 0 & 1.400 & .050 & .100 \\ 0 & 1 & 0 & 0 & 0 & .400 & .050 & .100 \end{array}\right]$$

Responses of these models to samples of the input signal $\mathbf{u}(t)$ at $t_i = iT$ for $i = 0, 1, 2, 3, 4$, are given in Table 2.4 below. Also included in Table 2.4 for comparison are the samples of the C-T system response. The norms of the differences between the C-T response, $\mathbf{y}_c(t_i)$, and those of the three D-T models are:

$$\begin{aligned} \Delta_{ds} &= \| \mathbf{y}_c(t_i) - \mathbf{y}_{ds}(t_i) \| = 1.6975 \\ \Delta_{dr} &= \| \mathbf{y}_c(t_i) - \mathbf{y}_{dr}(t_i) \| = 0.59173 \times 10^{-3} \\ \Delta_{db} &= \| \mathbf{y}_c(t_i) - \mathbf{y}_{db}(t_i) \| = 0.87992 \times 10^{-1} \end{aligned}$$

TABLE 2.4. Simulation Results for D-T Equivalents

t_i (sec.)		0	2	4	6	8
$\mathbf{y}_c(t_i)$	y_1	0.000	0.309	1.228	3.011	5.594
	y_2	0.000	0.123	0.555	0.966	1.374
$\mathbf{y}_{d1}(t_i)$	y_1	0.000	0.250	0.747	2.107	4.290
	y_2	0.000	0.000	0.351	0.790	1.171
$\mathbf{y}_{d2}(t_i)$	y_1	0.000	0.309	1.228	3.011	5.594
	y_2	0.000	0.123	0.555	0.966	1.374
$\mathbf{y}_{d3}(t_i)$	y_1	0.000	0.371	1.249	2.994	5.615
	y_2	0.000	0.121	0.508	0.987	1.383

In order to illustrate the application of continualization procedures, i.e. the determination of C-T models from a given D-T model, we will first "generate" a D-T model using an input/output identification procedure which will be presented in Chapter 5 in detail.

Identification from Sampled Input/Output Data: As is well known, in order to perform a successful identification, the input signal selected should be sufficiently long and *sufficiently rich*. To this end the selected input vector $\mathbf{u}^*(t)$ is defined by

$$\begin{aligned}
u_1^*(t) &= u_1(t) + u_1(t - 10T)\\
u_2^*(t) &= u_2(t) + u_2(t - 12T)\\
u_3^*(t) &= u_3(t) + u_3(t - 14T)
\end{aligned}$$

where $u_i(t)$ are given in Table 2.3. Using $\mathbf{u}^*(t)$, the response $\mathbf{y}^*(t)$ of the system in Eq.(2.81) was calculated in the time interval $0 \le t \le 22T = 44$ seconds. The simulation of the C-T system in Eq.(2.81) was accomplished by solving the state-space differential equations at points 0.5 seconds apart. No measurement noise was added to the system response. The signals $\{\mathbf{u}^*(t), y^*(t)\}$, shown in Figs. 2.3 and 2.4, were then sampled at intervals of $T = 2$ seconds yielding the input-output samples $\{\mathbf{u}^*(t_i), y^*(t_i)\}$ of a C-T system to be identified. For present purposes it suffices that we obtain the "identified D-T" four-matrix model given by the following system matrix:

$$R_d = \begin{bmatrix} \mathbf{A}_d & \mathbf{B}_d \\ \mathbf{C}_d & \mathbf{D}_d \end{bmatrix} = \left[\begin{array}{ccccc|ccc} 0 & 0 & 1 & 0 & 0 & 2.084 & .238 & .309 \\ 0 & 0 & 0 & 1 & 0 & 1.235 & .070 & -.034 \\ .124 & .058 & .876 & 1.020 & 1.530 & 3.737 & .178 & .088 \\ 0 & 0 & 0 & 0 & 1 & .352 & -.090 & -.096 \\ -.030 & -.017 & .030 & -.086 & .281 & .015 & -.037 & -.008 \\ \hline 1 & 0 & 0 & 0 & 0 & 1.196 & .035 & .098 \\ 0 & 1 & 0 & 0 & 0 & .394 & .064 & .137 \end{array}\right]$$

With the identification procedure used, the representation R_d is in the *pseudo-observable* form. (Canonical forms will be discussed in detail in Chapter 3). As an admissible set of pseudo-observability indices, $\{n_i\}$, the following was selected, namely $\{n_i\} = \{2, 3\}$. The unique set of observability indices of R_c is $\{n_i\} = \{3, 2\}$. As was mentioned earlier, more details on this identification procedure will be presented in Chapter 5. Finally, the "identified D-T model" will now be used as a basis for illustrating the continualization techniques.

Continualization: Using the sampling interval $T = 2$, the D-T representation R_d above is *continualized* into:

(a) R_{cs} - C-T step-invariant (SI) equivalent
(b) R_{cr} - C-T ramp-invariant (RI) equivalent
(c) R_{cb} - C-T bilinear transform (BT) equivalent

As was pointed out in Sec. 2.3, to determine R_{cr}, it was first necessary to convert R_d into an equivalent five-matrix representation, R_{d5}, using Algorithm *R4R5*. Subsequently, using Eqs.(2.63), (2.64) and (2.67), the desired R_{cr} was obtained. To determine R_{cb}, the identified R_d was first converted to a five-matrix C-T repre-

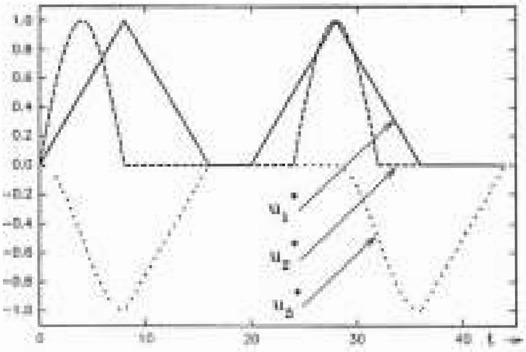


FIGURE 2.3 Excitations for Example 3

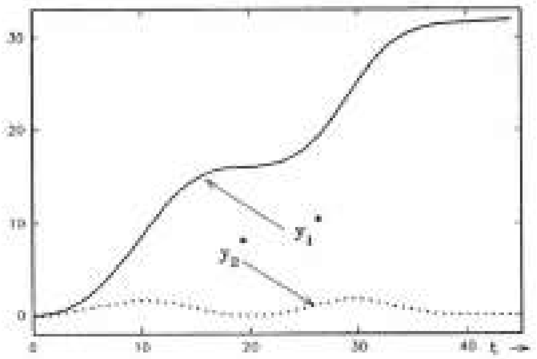


FIGURE 2.4 Responses for Example 3

sentation using Eqs.(2.69) to (2.71). Following this, Algorithm *R5R4* was used to obtain the desired four-matrix model, R_{dt}. The C-T representations thus obtained are given below:

$$R_{cs} = \begin{bmatrix} \mathbf{A}_{cs} & \mathbf{B}_{cs} \\ \mathbf{C}_{cs} & \mathbf{D}_{cs} \end{bmatrix} = \left[\begin{array}{ccccc|ccc} -.001 & 1.000 & .001 & -.002 & -.010 & .394 & .064 & .137 \\ -4.070 & -3.712 & 4.070 & -6.702 & 5.474 & .552 & .076 & .096 \\ .000 & .000 & -.000 & 1.001 & -.004 & 1.629 & .134 & .104 \\ .344 & .147 & -.344 & -.033 & 1.061 & .415 & -.030 & -.081 \\ -.074 & -.038 & .074 & -.295 & -.260 & .033 & -.036 & -.015 \\ - & - & - & - & - & - & - & - \\ 1 & 0 & 0 & 0 & 0 & 1.196 & .035 & .098 \\ 0 & 1 & 0 & 0 & 0 & .394 & .064 & .137 \end{array}\right]$$

$$R_{cr} = \begin{bmatrix} \mathbf{A}_{cr} & \mathbf{B}_{cr} \\ \mathbf{C}_{cr} & \mathbf{D}_{cr} \end{bmatrix} = \left[\begin{array}{ccccc|ccc} -.001 & 1.000 & .001 & -.002 & -.010 & -.003 & -.005 & .000 \\ -4.070 & -3.712 & 4.070 & -6.702 & 5.474 & .012 & .016 & -.003 \\ .000 & .000 & .000 & 1.001 & -.004 & 1.100 & .148 & .192 \\ .344 & .147 & -.344 & -.033 & 1.061 & .766 & .044 & -.069 \\ -.074 & -.038 & .074 & -.295 & -.260 & .121 & -.057 & -.044 \\ - & - & - & - & - & - & - & - \\ 1 & 0 & 0 & 0 & 0 & 1.000 & .001 & .000 \\ 0 & 1 & 0 & 0 & 0 & -.003 & -.004 & .001 \end{array}\right]$$

$$R_{cb} = \begin{bmatrix} \mathbf{A}_{cb} & \mathbf{B}_{cb} \\ \mathbf{C}_{cb} & \mathbf{D}_{cb} \end{bmatrix} = \left[\begin{array}{ccccc|ccc} -1.158 & -.076 & 1.158 & -1.148 & -.487 & -.832 & -.030 & .285 \\ -.051 & -1.028 & .051 & 1.847 & -1.502 & 1.142 & .316 & .265 \\ .158 & .076 & -.158 & 1.148 & .487 & 1.507 & .222 & .163 \\ .051 & .028 & -.051 & -.847 & 1.502 & .651 & -.059 & -.158 \\ -.051 & -.028 & .051 & -.153 & -.502 & .026 & -.056 & -.017 \\ \hline 1 & 0 & 0 & 0 & 0 & .858 & -.061 & -.126 \\ 0 & 1 & 0 & 0 & 0 & -.503 & -.064 & .084 \end{array}\right]$$

It is worth mentioning that the eigenvalues of the matrix $\mathbf{A}_{cs}$ (= $\mathbf{A}_{cr}$) obtained by the continualization of $\mathbf{A}_d$ are:

$$\{0\ ,\quad -0.9844\ ,\quad -1.0211\ ,\quad -1.0004 + j1.0002\ ,\quad -1.0004 - j1.0002\}$$

which are only slightly different from those of $\mathbf{A}_c$ given by Eq.(2.74).

Having determined the C-T models above, the responses of these models to the four samples of the input signal $\mathbf{u}(t)$, Table 2.3, were calculated, as was done for the D-T models, Table 2.4. In order to assess the accuracy of the proposed continualization procedures, only the samples of these C-T responses at the sampling instants are considered. Table 2.5 contains these results as well as the samples of the identified D-T model for comparison. The norms of the differences between $\mathbf{y}_d(t_i)$ and the responses of the three derived C-T models are as follows:

$$\Delta_{cs} = \|\ \mathbf{y}_d(t_i) - \mathbf{y}_{cs}(t_i)\ \| = 2.0550$$

$$\Delta_{cr} = \|\ \mathbf{y}_d(t_i) - \mathbf{y}_{cr}(t_i)\ \| = 0.12935 \times 10^{-4}$$

$$\Delta_{cb} = \|\ \mathbf{y}_d(t_i) - \mathbf{y}_{cb}(t_i)\ \| = 0.73568 \times 10^{-1}$$

From the normed differences, both for the discretization and the continualization, it may be concluded that the RI transformation is superior to either of the other two, primarily because of the particular selection of $\mathbf{u}^*(t)$ which does not contain step discontinuities. This should be expected since the SI transformation assumes constant values of input between samples, and the bilinear transformation (BT) is only satisfactory if $|p_iT| < 0.5$ for all poles p_i of the C-T system, which is not the case for this example.

TABLE 2.5. Simulation Results for C-T Equivalents

t_i (sec.)		0	2	4	6	8
$\mathbf{y}_c(t_i)$	y_1	0.000	0.309	1.228	3.011	5.594
	y_2	0.000	0.123	0.555	0.966	1.374
$\mathbf{y}_{cu}(t_i)$	y_1	0.000	0.567	1.927	4.108	7.110
	y_2	0.000	0.322	0.765	1.161	1.585
$\mathbf{y}_{ci}(t_i)$	y_1	0.000	0.309	1.228	3.011	5.594
	y_2	0.000	0.123	0.555	0.966	1.374
$\mathbf{y}_{cd}(t_i)$	y_1	0.000	0.253	1.219	3.001	5.578
	y_2	0.000	0.164	0.561	0.956	1.374

With the accuracy given in Tables 2.4 and 2.5 it cannot be seen just how well the C-T *identified* system output $\mathbf{y}_{ci}(t_i)$ matches that of the original C-T system, but the largest magnitude difference between the two, component by component, is 0.421×10^{-4}. The reader is urged to verify parts of the above examples using the same, or different, input data.

We conclude this chapter with a short section on an efficient method for the calculation of truncated power series.

Polynomial Reduction Using the Cayley-Hamilton Theorem

In several developments presented earlier, it was required that an $n \times n$ matrix, say $\mathbf{A}$, be calculated using a truncated power series of the form

$$\mathbf{A} = \sum_{i=0}^{N} c_i \mathbf{X}^i \tag{2.82}$$

In this section we discuss an efficient method for the calculation of truncated power series, referred to as Algorithm *POLR*. The series of Eq.(2.82) can be interpreted as an evaluation of the matrix polynomial $\mathbf{C}(\mathbf{X})$ of the matrix $\mathbf{X}$ where the N^{th} order polynomial $C(s)$ is given by

$$C(s) = \sum_{i=0}^{N} c_i s^i \tag{2.83}$$

As a result of the Cayley-Hamilton theorem, the calculation of Eq.(2.82) can be reduced to the evaluation of an $(n\text{-}1)^{st}$ order matrix polynomial $\mathbf{R}(\mathbf{X})$ of the matrix $\mathbf{X}$ where the coefficients r_i of the scalar polynomial $R(s)$ satisfy the following n conditions:

$$R(\lambda_i) = C(\lambda_i)\,, \quad \text{for} \quad i = 1,2,\ldots,n \tag{2.84}$$

where λ_i is the i^{th} eigenvalue of the $n \times n$ matrix $\mathbf{X}$. Using this approach, it can be verified that given the matrix $\mathbf{X}$ and the $N+1$ coefficients c_i of the polynomial $C(s)$, the n coefficients of the polynomial $R(s)$ can be obtained with the following:

Algorithm *POLR*

Syntax: $\mathbf{c}, \mathbf{X}$ $(POLR) \Rightarrow \mathbf{r}$

Algorithm:

```
1. Set $c_i \Rightarrow r_i$, for $0 \le i \le N$
2. Set $N+1 \Rightarrow k$ and $\det(s\mathbf{I} - \mathbf{X}) \Rightarrow f(s)$
3. Set $k-1 \Rightarrow k$
4. Set $r_{k-n+j} - r_k f_j \Rightarrow r_{k-n+j}$, for $0 \le j \le n-1$
5. If $k > n$, go to 3; else, stop
```

A listing of Algorithm *POLR*, implemented in *L-A-S*, may be found in Appendix C.

If the coefficients f_j define the (monic) characteristic polynomial of $\mathbf{X}$,

$$f(s) = \det(s\mathbf{I} - \mathbf{X}) \triangleq s^n + \sum_{j=0}^{n-1} f_j s^j \tag{2.85}$$

then, the first n coefficients r_j, $0 \le j \le n-1$ define the $(n-1)^{st}$ order polynomial $R(s)$ satisfying Eq.(2.84).

From Algorithm *POLR* it is clear that evaluating the matrix $\mathbf{A}$ in Eq.(2.82) is equivalent to evaluating

$$\mathbf{A} = \sum_{i=0}^{n-1} r_i \mathbf{X}^i \tag{2.86}$$

thereby considerably reducing the computational time and, more importantly, the accumulation of round-off errors. This method works well even if $\mathbf{X}$ is completely general with multiple eigenvalues. The *POLR* algorithm given above may be considered as a computational simplification of a standard procedure based on the Cayley-Hamilton theorem of matrix algebra. This standard procedure calculates coefficients r_i of the polynomial $r(s)$ from

$$r^{(k)}(\lambda_i) = c^{(k)}(\lambda_i)\,, \quad \text{for} \quad i = 1,2,\ldots,m \quad \text{and} \quad k = 1,2,\ldots,n_i \tag{2.87}$$

where λ_i is an eigenvalue of $\mathbf{X}$, n_i is its algebraic multiplicity and m is the number of distinct eigenvalues. Obviously, if all of the eigenvalues of $\mathbf{X}$ are distinct, then $m = n$ and $n_i = 1$ for all i. The notation of Eq.(2.87) is defined by

$$r^{(k)}(\lambda_i) = \frac{d^k r(\lambda)}{d\lambda^k}\Big|_{\lambda = \lambda_i} \tag{2.88}$$

The computational simplification of the *POLR* algorithm is useful in that when $\mathbf{X}$ has multiple eigenvalues, it is neither necessary to determine the algebraic multiplicities, nor to evaluate the derivatives in Eq.(2.88). The *POLR* algorithm also works for matrices having a spectral radius greater than 0.5.

2.5 Summary

To summarize the developments in this chapter a set of numerically robust algorithms was presented. These algorithms deal with the often encountered problems of *discretization* of continuous-time (C-T) models as well as the inverse problem of recreating a C-T model from a given discrete-time (D-T) model. This latter operation we have referred to as *continualization*. The algorithms described comprise, in addition to the standard SI (*ZOH*) procedures, two methods which are commonly referred to in the signal processing literature, namely the *bilinear transformation* (BT) and a method called the *ramp-invariant* (RI) method that is equivalent to the next higher order approximation beyond the SI (ZOH), representing a piecewise linear approximation to the input functions. With these algorithms the design engineer has complete flexibility to move between the continuous and discrete model domains.

To assist readers in the "maze" of "time-domain" conversion algorithms introduced in this chapter, as well as in Chapter 1, and to relate these algorithms to their *L-A-S* implementations given in Appendix C, let us review these algorithms once more in a slightly different way. The list below relates algorithms that have been discussed with the names of their *L-A-S* "counter-parts," i.e. *L-A-S* operators, or subroutines:

Algorithm Name	***L-A-S* Operator or Subroutine Name**
EATj	*EATj.SUB*
EAT	*EAT.SBR*
EATF	*EATF* and *EATF.SBR*
LNM	*LNM* and *LNM.SBR*
LNMj	*LNMj.SUB*
SI-C-D	*SICD.SBR*
RI-C-D	*RICD.SBR*

BT-C-D	*BCDC.SUB*
BT-D-C	*BCDC.SUB*
SI-D-C	not available
RI-D-C	not available
"Service" Algorithms:	
SQM	*SQM* and *SQM.SUB*
POLR	*POLR* and *POLR.SUB*
POM	*POM* and *POM.SUB*
FACT	*FACT.SUB*
FLN	*FLN.SUB*
R5R4	*R5R4.SUB*
R4R5	*R4R5.SUB*
JFR	*JFR*
EGV	*EGV*
EFJF	*EFJF*

Note that whenever an *L-A-S* operator is available, it is more convenient to use the operator version, rather than the corresponding subroutine, since its execution is much faster. The listings of certain subroutines are given to show how they might be implemented using the *L-A-S* language, or any other CAD package. A close correspondence should be noticed between the steps of an algorithm and the associated *L-A-S* implementation.

Historically, the development of the *L-A-S* language progressed by constantly undergoing modifications and upgrading. Early on, all algorithms were implemented as subroutines. As specific algorithms saw increasing use, users requested "single step operations" for speed and convenience. As *L-A-S* continues to grow, this trend of "upgrading" subroutines to operators will, no doubt, continue. Thus, in the future some algorithms which are implemented as subroutines in this text will be implemented as operators.

Recall that both Algorithms *BT-C-D* and *BT-D-C* are implemented by a single *L-A-S* algorithm with the syntax:

$$\mathbf{A}_1,\ \mathbf{B}_1,\ T,\ Icdc\ (BCDC) \rightarrow \mathbf{A}_2,\ \mathbf{B}_2,\ \mathbf{P}$$

where, in the case of the algorithm flag, *Icdc*:

Icdc = 1, it performs the task of *BT-C-D*, Eqs.(2.35), while for
Icdc = 2, Eqs.(2.71), required by *BT-D-C* are used.

Some of these algorithms, such as *RI-C-D*, *BT-C-D* and *BT-D-C* calculate five matrix models from a standard four matrix model, albeit in a different time-domain, while other algorithms, e.g. *RI-D-C*, calculate a four matrix model from a five matrix model. To help this situation, a set of *L-A-S* algorithms has been developed which greatly facilitates these various time-domain conversions. This is achieved by developing algorithms which convert from a standard four-matrix model in one domain into a corresponding standard four-matrix model in the other domain. All of these algorithms and subroutines use four matrices in $R = \{\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D}\}$ and produce a corresponding "converted" representation $\tilde{R} = \{\tilde{\mathbf{A}},\tilde{\mathbf{B}},\tilde{\mathbf{C}},\tilde{\mathbf{D}}\}$. The syntactical definitions of these algorithms are

$$\textbf{For C-T} \Rightarrow \textbf{D-T conversions:} \tag{2.89}$$

$$\mathbf{A}_c,\ \mathbf{B}_c,\ \mathbf{C}_c,\ \mathbf{D}_c,\ T,\ \epsilon\ (SRCD) \rightarrow \mathbf{A}_d,\ \mathbf{B}_{ds},\ \mathbf{B}_{dr},\ \mathbf{C}_d,\ \mathbf{D}_{ds},\ \mathbf{D}_{dr}$$

$$\mathbf{A}_c,\ \mathbf{B}_c,\ \mathbf{C}_c,\ \mathbf{D}_c,\ T,\ \epsilon\ (BLCD) \rightarrow \mathbf{A}_{db},\ \mathbf{B}_{db},\ \mathbf{C}_{db},\ \mathbf{D}_{db}$$

$$\textbf{and for D-T} \Rightarrow \textbf{C-T conversions:} \tag{2.90}$$

$$\mathbf{A}_d,\ \mathbf{B}_d,\ \mathbf{C}_d,\ \mathbf{D}_d,\ T,\ \epsilon\ (SRDC) \rightarrow \mathbf{A}_c,\ \mathbf{B}_{cs},\ \mathbf{B}_{cr},\ \mathbf{C}_c,\ \mathbf{D}_{ds},\ \mathbf{D}_{cr}$$

$$\mathbf{A}_d,\ \mathbf{B}_d,\ \mathbf{C}_d,\ \mathbf{D}_d,\ T,\ \epsilon\ (BLDC) \rightarrow \mathbf{A}_{cb},\ \mathbf{B}_{cb},\ \mathbf{C}_{cb},\ \mathbf{D}_{cb}$$

As the names of the algorithms in Eq.(2.89) suggest, given a C-T state space representation:

$$R_c = \{\mathbf{A}_c,\mathbf{B}_c,\mathbf{C}_c,\mathbf{D}_c\} \text{ and the sampling interval } T$$

- Algorithm *SRCD* calculates four matrix D-T models for both SI and RI equivalents:

$$R_{ds} = \{\mathbf{A}_{ds},\mathbf{B}_{ds},\mathbf{C}_{ds},\mathbf{D}_{ds}\} \text{ and } R_{dr} = \{\mathbf{A}_{dr},\mathbf{B}_{dr},\mathbf{C}_{dr},\mathbf{D}_{dr}\} \tag{2.91}$$

  where $\mathbf{A}_{ds} = \mathbf{A}_{dr} = \mathbf{A}_d$ and $\mathbf{C}_{ds} = \mathbf{C}_{dr} = \mathbf{C}_d$.
- Algorithm *BLCD* calculates a four matrix D-T model using the bilinear transformation:

$$R_{db} = \{\mathbf{A}_{db},\mathbf{B}_{db},\mathbf{C}_{db},\mathbf{D}_{db}\} \tag{2.92}$$

Both algorithms, in addition to using the "basic" conversion algorithms, i.e. *EATF* and *BT-C-D* (or the subroutine *BCDC.SUB*), also use Algorithm *R5R4* to obtain the four matrix D-T model given by Eqs.(2.91) and (2.92).

Similarly, considering the algorithms in Eq.(2.90), given a D-T state space representation:

$$R_d = \{\mathbf{A}_d,\mathbf{B}_d,\mathbf{C}_d,\mathbf{D}_d\} \text{ and the sampling interval } T$$

- Algorithm *SRDC* calculates four matrix C-T models for both SI and RI equivalents:

$$R_{cs} = \{\mathbf{A}_{cs},\mathbf{B}_{cs},\mathbf{C}_{cs},\mathbf{D}_{cs}\} \text{ and } R_{cr} = \{\mathbf{A}_{cr},\mathbf{B}_{cr},\mathbf{C}_{cr},\mathbf{D}_{cr}\} \tag{2.93}$$

  where $\mathbf{A}_{cs} = \mathbf{A}_{cr} = \mathbf{A}_c$ and $\mathbf{C}_{cs} = \mathbf{C}_{cr} = \mathbf{C}_c$.
- Algorithm *BLCD* calculates a four matrix D-T model using the bilinear transformation:

$$R_{cb} = \{\mathbf{A}_{cb},\mathbf{B}_{cb},\mathbf{C}_{cb},\mathbf{D}_{cb}\} \tag{2.94}$$

In accordance with the developments in Section 2.3, Algorithm SRDC, in addition to using the basic conversion Algorithm *LNM*, first uses Algorithm *R4R5* in order to obtain the five matrix D-T model required by *RI-C-D*. On the other hand, Algorithm *BLDC*, in a similar manner to the previously mentioned *BLCD* and *SRCD*, uses Algorithm *BT-C-D*, i.e. subroutine *BCDC.SUB*, followed by Algorithm *R5R4* in order to obtain the desired four matrix model R_4 given by Eq.(2.94).

In order to facilitate time domain conversions even further, a single "unifying" algorithm, referred to as *CTDT*, has been developed. Its syntax is:

$$\mathbf{A},\ \mathbf{B},\ \mathbf{C},\ \mathbf{D},\ \epsilon,\ Isrb\ (CTDT) \rightarrow \tilde{\mathbf{A}},\ \tilde{\mathbf{B}},\ \tilde{\mathbf{C}},\ \tilde{\mathbf{D}} \qquad (2.95)$$

where, given a state space representation in ONE domain (C-T or D-T)

$$R = \{\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D}\}$$

it calculates the *equivalent* model in THE OTHER domain (D-T or C-T)

$$\tilde{R} = \tilde{\mathbf{A}},\ \tilde{\mathbf{B}},\ \tilde{\mathbf{C}},\ \tilde{\mathbf{D}}$$

The desired conversion is specified by the seventh input argument, i.e. algorithm flag *Isrb* (where "srb" stands for the *step*, *ramp* and *bilinear* transformations). More specifically, for *Isrb* = 1, 2 or 3, Algorithm *CTDT* assumes that the given R is a C-T model and that the representation $\tilde{R}$ is consequently a D-T model corresponding to SI , RI or BT equivalents, respectively.

Conversely, if the algorithm flag *Isrb* = -1 , -2 or -3, the algorithm treats the the given representation R as the D-T model and calculates a C-T model $\tilde{R}$ as an SI , RI or BT equivalent.

For more details about this "unifying" algorithm readers are referred to Chapter 5, where a full description of Algorithm *CTDT* from the user's point of view is given. Listings of all algorithms mentioned in this section, implemented using the *L-A-S* language, are given in Appendix C.

2.6 References

Brogan (1991), Chen (1984), Golub and Van Loan (1991) and Kailath (1980) are a few of the better standard references in the general context of multivariable systems. More specific references are Bingulac and Cooper (1990) for SI and RI discretization techniques, and Cooper and Bingulac (1990), as well as Bingulac and VanLandingham (1992), for background on computing the corresponding "continualized" models. See also Laub (1985) and Moler and Van Loan (1978) for some interesting reading on the topic of discretization.

Bingulac, S. and D. Cooper (1990), "Derivation of discrete- and continuous-time ramp invariant representations," *Electronics Letters*, **26**, 10, 664-666.

Bingulac, S. and H.F. VanLandingham (1991), "Robust algorithms for the transformation of MIMO system between continuous- and discrete-time domains," *Proceedings of the 29th Allerton Conference*, University of Illinois, October 1991, pp.448-454.

Bingulac, S. and H.F. VanLandingham (1992), "A unified approach to the discretization and 'continualization' of MIMO systems," *Control Theory and Advanced Technology (C-TAT), Journal of the Mita Press*, **8**, 3.

Brogan, W.L. (1984), *Modern Control Theory, 3rd Edition*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Chen, C-T. (1984), *Linear System Theory and Design*, Holt, Rinehart and Winston, Inc., New York, NY.

Cooper, D. and S. Bingulac (1990), "Computational improvement in the calculation of the natural log of a square matrix," *Electronics Letters*, **26**, 13, 861-862.

Golub, G. and C.F. Van Loan (1991), *Matrix Computations*, The Johns Hopkins University Press, Baltimore, MD.

Haykin, S.S. (1972), "A unified treatment of recursive digital filtering," *IEEE Trans. on Automatic Control*, February 1972, pp 113-116.

Kailath, T. (1980), *Linear Systems*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Laub, A.J. (1985), "Numerical linear algebra aspects of control design computations," *IEEE Trans. on Automatic Control*, **AC-30**, 2, 97-108.

Moler, C.B., and C.F. Van Loan (1978), "Nineteen dubious ways to compute the exponential of a matrix," *SIAM Review*, **20**, 801-836.

VanLandingham, H.F. (1985), *Introduction to Digital Control Systems*, Macmillan Publishing Co., New York, NY.

2.7 Exercises

2.1 Two (4 × 4) matrices $\mathbf{A}_c$ and $\mathbf{A}_{c1}$ are given below:

$$\mathbf{A}_c = \begin{bmatrix} -1 & 1 & 0 & -1 \\ -1 & -2 & 1 & 0 \\ 1 & 0 & -2 & 1 \\ 0 & 0 & 0 & -2 \end{bmatrix}$$

$$\mathbf{A}_{c1} = \begin{bmatrix} -1 & 1 & 0 & -1 \\ -1 & -2 & 1 & 1 \\ 1 & 0 & -2 & 1 \\ 0 & 0 & 0 & -1 \end{bmatrix}$$

Calculate:

(a) The eigenvalues of $\mathbf{A}_c$ and $\mathbf{A}_{c1}$.
(b) Verify that $\mathbf{A}_{c1}$ is not diagonalizable.
(c) Determine $\mathbf{A}_d = \exp(\mathbf{A}_c T)$ and $\mathbf{A}_{d1} = \exp(\mathbf{A}_{c1} T)$ with a sampling interval of $T = 2$ sec.
(d) Determine $\mathbf{A}_{cnew} = \ln(\mathbf{A}_d)/T$ and $\mathbf{A}_{c1new} = \ln(\mathbf{A}_{d1})/T$
(e) Verify that $\mathbf{A}_c = \mathbf{A}_{cnew}$ and that $\mathbf{A}_{c1} = \mathbf{A}_{c1new}$.
(f) If available, use some other software package to perform the same calculations. Many such packages do not implement functions of a matrix if the matrix is not diagonalizable.

Hints:

- To calculate the eigenvalues of a matrix, use operator EGV.
- To verify if a matrix $\mathbf{A}$ is diagonalizable, check the rank, or null space of the matrix $\mathbf{B} = \mathbf{A} - \lambda_i \mathbf{I}_n$, where λ_i is an eigenvalue of $\mathbf{A}$.
- To calculate the rank, or null space, use operator NRS.
- To calculate the natural log of a matrix, use either (or both) the operator LNM and/or the subroutine LNMj.SUB.
- To calculate the matrix exponential function of a matrix, use either (or both) the operator EATF and/or the subroutine EATj.SUB.
- Verify that subroutines LNMj.SUB and EATj.SUB are not applicable in the case of non-diagonalizable matrices.

A version of an *L-A-S* program which solves this exercise is available in the *L-A-S* subdirectory C:\LAS\DPF\EXER21.DPF.

2.2 Two (4 × 4) matrices $\mathbf{A}_d$ and $\mathbf{A}_{d1}$ are given below:

$$\mathbf{A}_d = \begin{bmatrix} -.9 & .8 & 1 & -.8 \\ -.8 & .1 & 0 & 1.7 \\ -1.4 & .8 & 1.5 & .2 \\ 0 & 0 & 0 & .8 \end{bmatrix}$$

$$\mathbf{A}_{d1} = \begin{bmatrix} -.9 & .8 & 1 & -.8 \\ -.8 & .1 & 0 & 1.4 \\ -1.4 & .8 & 1.5 & .2 \\ 0 & 0 & 0 & .5 \end{bmatrix}$$

Calculate:

(a) —the eigenvalues of $\mathbf{A}_d$ and $\mathbf{A}_{d1}$.
(b) Verify that $\mathbf{A}_{d1}$ is not diagonalizable.
(c) Determine $\mathbf{A}_c = \ln(\mathbf{A}_d)/T$ and $\mathbf{A}_{c1} = \ln(\mathbf{A}_{d1})/T$ with a sampling interval of $T = 2$ sec.
(d) Determine $\mathbf{A}_{dnew} = \exp(\mathbf{A}_c T)$ and $\mathbf{A}_{d1new} = \exp(\mathbf{A}_{c1} T)$.
(e) Verify that $\mathbf{A}_d = \mathbf{A}_{dnew}$ and that $\mathbf{A}_{d1} = \mathbf{A}_{d1new}$.
(f) If available, use some other software package to perform the same calculations.

Hints:

- To calculate the eigenvalues of a matrix, use operator EGV.
- To verify if a matrix $\mathbf{A}$ is diagonalizable, check the rank, or null space of the matrix $\mathbf{B} = \mathbf{A} - \lambda_i \mathbf{I}_n$, where λ_i is an eigenvalue of $\mathbf{A}$.
- To calculate the rank, or null space, use operator NRS.
- To calculate the natural log of a matrix, use either (or both) the operator LNM and/or the subroutine LNMj.SUB.
- To calculate the matrix exponential function of a matrix, use either (or both) the operator EATF and/or the subroutine EATj.SUB.
- Verify that subroutines LNMj.SUB and EATj.SUB are not applicable in the case of non-diagonalizable matrices.

A version of an *L-A-S* program which solves this exercise is available in the *L-A-S* subdirectory C:\LAS\DPF\EXER22.DPF.

2.3 Using the following system state space representation R_d of a D-T MIMO system:

$$\left[\begin{array}{c|c} \mathbf{A}_d & \mathbf{B}_d \\ \hline \mathbf{C}_d & \mathbf{D}_d \end{array}\right] = \left[\begin{array}{cccc|cc} 1.1 & .8 & 1 & .8 & 1 & 0 \\ -.8 & .1 & 0 & .6 & 0 & -1 \\ -.6 & -.8 & -.5 & .2 & -1 & 0 \\ 0 & 0 & 0 & .5 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \end{array}\right]$$

with sampling interval $T = 2$ sec., and the number of samples $N = 41$, calculate:

(a) —the response $\mathbf{y}_d(k)$, $k=[0,N\text{-}1]$, of R_d to zero initial conditions and an input vector $\mathbf{u}(k)$, with $\mathbf{u}(0)=0$. For non-zero values of $\mathbf{u}(k)$, i.e. $\mathbf{u}(k)$, $k=[1,N\text{-}1]$, use pseudo random numbers.

(b) —equivalent C-T models using:
 (1) —a step-invariant assumption — the SI-C-T model R_{cs}
 (2) —a ramp-invariant assumption — the RI-C-T model R_{cr}
 (3) —a bilinear transformation — the BT-C-T model R_{cb}

(c) —the responses $\mathbf{y}_{cs}(t)$, $\mathbf{y}_{cr}(t)$ and $\mathbf{y}_{cb}(t)$ of the obtained SI-C-T, RI-C-T and BT-C-T models, respectively, to zero initial conditions and an input vector $\mathbf{u}(t)$ having at $t = kT$, $k=[0,N\text{-}1]$, the same values $\mathbf{u}(k)$ used in calculating the response $\mathbf{y}_d(k)$ of the given system R_d.

(d) Plot all responses and check the differences between $\mathbf{y}_d(k)$ and the responses of each obtained C-T equivalent model.

(e) Determine which C-T model gives the response which is closest to $\mathbf{y}_d(k)$.

Hints:

- Define the matrices in R_d using either the DMA or the INPM operator.
- Define the scalars T and N using either the DMA or the DSC operator.
- The system order and number of inputs may be extracted from $\mathbf{A}_d$ and $\mathbf{B}_d$ either by the operator CDI or subroutine GETD.SUB.
- The matrix $\mathbf{u}$ containing the samples $\mathbf{u}(k)$ may be defined using operators DPM and SHR.
- Zero initial conditions may be established using the DZM operator.
- The responses of the D-T and C-T models may be calculated using CDSR.SUB.
- Response plotting may be performed using the DIS operator.
- For axes labeling and scaling, operators YLAB, XLAB and YXSC may be used.

- The required C-T models can be calculated in the following two ways:
  - The "easy" way, i.e. using subroutine CTDT.SBR, where the quantities *Isrb* = -1, -2 and -3 should be used for the algorithm flag *Isrb*.
  - The "hard" way, i.e. for
    - —the SI-C-T model use operators LNM and EATF, Eqs.(2.63), (2.65) and (2.9).
    - —the RI-C-T model use the same operators, but Eqs.(2.63), (2.33), (2.67) and (2.68), as well as subroutine R4R5.SUB for converting the given four-matrix D-T model R_d into the required five-matrix model given by Eq.(2.21).
    - —the BT-C-T model use Eqs.(2.70) - (2.72) and subroutine R5R4.SUB for converting the obtained five-matrix C-T model into the required four-matrix model R_{cb}.
- Note that the matrix $\mathbf{A}_d$ has multiple eigenvalues and that it is not diagonalizable. Therefore, subroutines LNMj.SUB and EATj.SUB should not be used!
- The *L-A-S* program EXER23.DPF, residing in the subdirectory C:\LAS\DPF\, contains a possible solution to Exercise 2.3.

2.4 Using the following system state space representation R_c of a C-T MIMO system:

$$\left[\begin{array}{c|c} \mathbf{A}_c & \mathbf{B}_c \\ \hline \mathbf{C}_c & \mathbf{D}_c \end{array}\right] = \left[\begin{array}{cccc|cc} -1 & 1 & 0 & 1 & 1 & 0 \\ 1 & -1 & 1 & 0 & 0 & -1 \\ 1 & -1 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \end{array}\right]$$

with sampling interval T = 2 sec., and the number of samples N = 41, calculate:

(a) —the response $\mathbf{y}_c(k)$, k=[0,N-1], of R_c to zero initial conditions and an input vector $\mathbf{u}(t)$, with $\mathbf{u}(0)$=0. For samples $\mathbf{u}(kT)$, k=[1,N-1], use pseudo random numbers.

(b) —equivalent D-T models using:

(1) —a step-invariant assumption — the SI-D-T model R_{ds}
(2) —a ramp-invariant assumption — the RI-D-T model R_{dr}
(3) —a bilinear transformation — the BT-D-T model R_{db}

(c) —the responses $\mathbf{y}_{ds}(t)$, $\mathbf{y}_{dr}(t)$ and $\mathbf{y}_{db}(t)$ of the obtained SI-D-T, RI-D-T and BT-D-T models, respectively, to zero initial conditions and an input vector $\mathbf{u}(k)$ having the same values as $\mathbf{u}(kT)$, $k=[0,N\text{-}1]$, used in calculating the response $\mathbf{y}_c(t)$ of the given system R_c.

(d) Plot all responses and check the differences between $\mathbf{y}_c(kT)$ and the responses of each obtained D-T equivalent model.

(e) Determine which D-T model gives the response which is closest to $\mathbf{y}_c(kT)$.

Hints:

- Define the matrices in R_c using either the DMA or the INPM operator.
- Define the scalars T and N using either the DMA or the DSC operator.
- The system order and number of inputs may be extracted from $\mathbf{A}_c$ and $\mathbf{B}_c$ either by the operator CDI or subroutine GETD.SUB.
- The matrix $\mathbf{u}$ containing the samples $\mathbf{u}(kT)$ may be defined using operators DPM and SHR.
- Zero initial conditions may be established using the DZM operator.
- The responses of the C-T and D-T models may be calculated using CDSR.SUB.
- Response plotting may be performed using the DIS operator.
- For axes labeling and scaling, operators YLAB, XLAB and YXSC may be used.
- The required D-T models can be calculated in the following two ways:
  - The "easy" way, i.e. using subroutine CTDT.SBR, where for the algorithm flag *Isrb* the quantities *Isrb* = 1, 2 and 3 should be used.
  - The "hard" way, i.e. for
    - □ —the SI-D-T model use the operator EATF and Eq.(2.9).
    - □ —the RI-D-T model use the same operator, but Eqs.(2.32), (2.33) and the subroutine R5R4.SUB for converting the obtained five-matrix D-T model, Eq.(2. 21), into the required four-matrix model R_{dr}.
    - □ —the BT-D-T model use Eqs.(2.35) and subroutine R5R4.SUB for converting the obtained five-matrix D-T model into the required four-matrix model R_{db}.

- Note that the matrix $\mathbf{A}_c$ is singular, has multiple eigenvalues and is not diagonalizable. Therefore, the subroutines EATj.SUB should <u>not</u> be used!

- The *L-A-S* program EXER24.DPF, residing in the subdirectory C:\LAS\DPF\, contains a possible solution to Exercise 2.4.

Chapter 3 System Modeling

In this chapter we will elaborate on the brief introduction of system models given in Chapter 1. In carrying out design problems engineers and analysts frequently need to convert between various system descriptions for insight into the different phases of the design process. The principal linear, time-invariant, dynamic system model types considered in this chapter are:

(1) State space models,
(2) Transfer function matrices, and
(3) Matrix fraction description (MFD) models.

In the first section single-input, single-output (SISO) systems and their canonical representations are considered.

3.1 Canonical Forms for SISO Systems

As an introduction to the special standard forms for representing systems in state space, we will restrict our attention to SISO systems in this section. Later in the chapter the concepts will be extended to MIMO systems. There are three main state space structures that are recognized as "standard:"

- The *controllable* canonical form,
- The *observable* canonical form, and
- The *Jordan* canonical form.

These three forms also have versions which have minor variations, which can arise, for instance, from different labelings of the state variables. The following discussion will be presented from the point of view of C-T systems and the Laplace Transform variable, s; but the reader should keep in mind that exactly the same canonical forms also hold for D-T systems with the corresponding z-Transform notation.

In the following each of the three forms mentioned above will be studied as they pertain to the SISO transfer function:

$$y(s) = g(s)\,u(s) = \frac{b(s)}{a(s)}\,u(s) \tag{3.1}$$

where the numerator and denominator of the transfer function $g(s)$ are polynomials given by

$$a(s) = \sum_{i=0}^{n} a_i\, s^i \quad \text{and} \quad b(s) = \sum_{i=0}^{m} b_i\, s^i \tag{3.2}$$

Of course, it is well known that the transfer function of Eq.(3.1) represents a system described by the differential equation

$$\sum_{i=0}^{n} a_i \frac{d^i}{dt^i} y(t) = \sum_{i=0}^{m} b_i \frac{d^i}{dt^i} u(t) \tag{3.3}$$

For the model in either form to be realizable, the order of the numerator, m, must be less than or equal to the order of the denominator, n. In the subsequent discussions it will be assumed that $a_n = 1$ and $m = n$, i.e. a proper system.

3.1.1 The Controllable Canonical Form

The first method to be presented is a state model based on a natural extension of the *phase variables* of a differential equation, namely,

$$x_1 = y \ , \quad x_2 = \frac{dy}{dt} \ , \quad \dots \ , \quad x_n = \frac{d^{n-1}y}{dt^{n-1}}$$

In order to simplify the notation, a third-order transfer function will be used for the development. However, because of the regularity of the canonical form, the general case will be clear. Consider that the transfer function, $g(s)$, of Eq.(3.1) is given by

$$g(s) = \frac{y(s)}{u(s)} = \frac{b_3 s^3 + b_2 s^2 + b_1 s + b_0}{s^3 + a_2 s^2 + a_1 s + a_0} \tag{3.4}$$

Alternatively, $g(s)$ could be written as

$$g(s) = \frac{y(s)}{u(s)} = \frac{b_3 + b_2 s^{-1} + b_1 s^{-2} + b_0 s^{-3}}{1 + a_2 s^{-1} + a_1 s^{-2} + a_0 s^{-3}} \tag{3.5}$$

From Eq.(3.5) we can solve for $y(s)$ as

$$y(s) = (b_3 + b_2 s^{-1} + b_1 s^{-2} + b_0 s^{-3})\, e(s) \tag{3.6}$$

where

$$e(s) = \frac{u(s)}{1 + a_2 s^{-1} + a_1 s^{-2} + a_0 s^{-3}} \tag{3.7}$$

An equivalent expression for $e(s)$ can be obtained by cross-multiplying in Eq.(3.7) and solving for $e(s)$ in terms of itself and $u(s)$, i.e.

$$e(s) = u(s) - (a_2 s^{-1} + a_1 s^{-2} + a_0 s^{-3})\, e(s) \tag{3.8}$$

The lower portion of Fig. 3.1 illustrates Eq.(3.8) in that the signal e is the sum of the four terms: u, $-a_2s^{-1}e$, $-a_1s^{-2}e$ and $-a_0s^{-3}e$; the last three terms are the "feedback" terms in the diagram. It is easy to see how the diagram of Fig. 3.1 extends to higher order systems by cascading the n integration blocks together for an n^{th} order system.

By labeling the outputs of the integration blocks as *state variables*, as shown in Fig. 3.1, the following equations may be derived:

$$\begin{aligned} \dot{x}_1(t) &= x_2(t) \\ \dot{x}_2(t) &= x_3(t) \\ \dot{x}_3(t) &= -a_0\,x_1(t) - a_1\,x_2(t) - a_2\,x_3(t) + u(t) \end{aligned} \tag{3.9}$$

Finally, from Eq.(3.6) the upper portion of Fig. 3.1 may be drawn; that is, $y(s)$ is a linear combination of the signal $e(s)$ and its integrals. We may refer to $s^{-1}e(s)$ as the integral of $e(s)$. Thus, the output equation is

$$y(t) = (b_0 - a_0b_3)x_1(t) + (b_1 - a_1b_3)x_2(t) + (b_2 - a_2b_3)x_3(t) + b_3u(t) \tag{3.10}$$

When Eqs.(3.9) and (3.10) are put into a vector-matrix form, the following structure of the state space model is obtained.

$$\begin{aligned} \begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \dot{x}_3(t) \end{bmatrix} &= \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -a_0 & -a_1 & -a_2 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u(t) \\ y(t) &= \begin{bmatrix} b_0-a_0b_3 & b_1-a_1b_3 & b_2-a_2b_3 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix} + [\,b_3\,]\,u(t) \end{aligned} \tag{3.11}$$

In general, the *controllable canonical SISO model* is given by

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{A}_c\,\mathbf{x}(t) + \mathbf{b}_c\,u(t) \\ y(t) &= \mathbf{c}_c\,\mathbf{x}(t) + d_c\,u(t) \end{aligned} \tag{3.12}$$

where
$$\begin{aligned} \mathbf{A}_c &= \begin{bmatrix} \mathbf{0} & \mathbf{I}_{n-1} \\ \multicolumn{2}{c}{-\mathbf{a}} \end{bmatrix}, \quad \mathbf{b}_c = \begin{bmatrix} \mathbf{0} \\ 1 \end{bmatrix} \\ \mathbf{c}_c &= [\ c_1 \ \cdots \ c_n\ ], \quad d_c = [\,b_n\,] \end{aligned} \tag{3.13}$$

with $\quad \mathbf{a} = [a_0 \ \cdots \ a_{n-1}]$, and $c_i = b_{i-1} - a_{i-1}b_n$ for $0 \le i \le n-1$ (3.14)

Note that for SISO systems $\mathbf{A}$ is an $(n \times n)$ matrix, $\mathbf{b}$ is an $(n \times 1)$ column, $\mathbf{c}$ is a $(1 \times n)$ row, and d is a scalar.

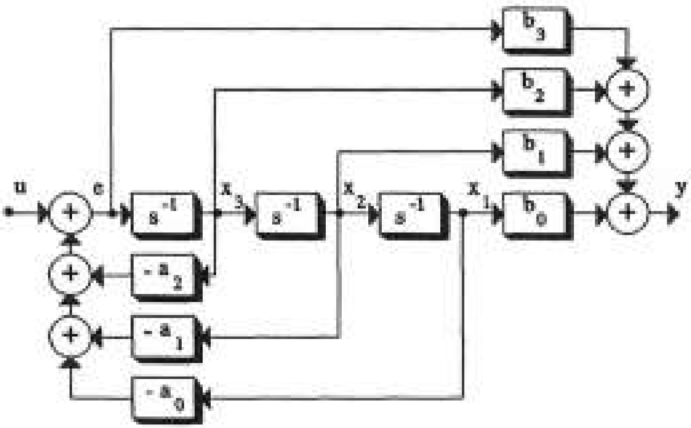


FIGURE 3.1 SISO Feedback (Controllable) Form

As will be made clear later, the canonical form of Eqs.(3.11), represented in Fig. 3.1, will be called the *SISO feedback (controllable)* form. Its main property is that the input vector $\mathbf{b}_c$ has a unit element at the location corresponding to the row of $\mathbf{A}_c$ with non-zero/non-unity elements. A modification of Eqs.(3.11) preferred by some authors is obtained by labeling the states in Fig. 3.1 in reverse order. In this case, with x_1 and x_3 interchanged in Fig. 3.1, the Eqs.(3.11) become

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \dot{x}_3(t) \end{bmatrix} = \begin{bmatrix} -a_2 & -a_1 & -a_0 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} u(t)$$

$$y(t) = \begin{bmatrix} b_2 - a_2 b_3 & b_1 - a_1 b_3 & b_0 - a_0 b_3 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix} + [\, b_3 \,]\, u(t) \tag{3.15}$$

which presents a different looking, but still, a controllable form.

The controllable canonical form of either version provides an extremely useful method of obtaining a set of state equations from a given transfer function. With sufficient practice the reader will be able to skip the intermediary diagram and fill in the state model directly from the transfer function. For instance, in Eqs.(3.11) the upper rows of the state coefficient matrix are formed from a shifted identity matrix, while the last row has a direct correspondence to the denominator of the transfer function. The input matrix is all zeros except for the last entry, which is unity. The output matrix and feedthrough element incorporate the numerator coefficients in a specific manner. Note that the absence of the b_3 (feedthrough) element greatly simplifies the output matrix.

3.1.2 The Observable Canonical Form

In this section a second state space form will be presented based on the same transfer function $g(s)$ in Eq.(3.4) or Eq.(3.5). In a later section the concept of equivalent state space descriptions, i.e. when two state models represent the same system, will be discussed. For now it is sufficient to accept the non-uniqueness of state space models.

As in the previous section the generic third-order system will be used for the development. After cross-multiplying in Eq.(3.5) we isolate y and group terms as follows.

$$y = b_3 u + s^{-1}(b_2 u - a_2 y) + s^{-2}(b_1 u - a_1 y) + s^{-3}(b_0 u - a_0 y) \tag{3.16}$$

where the argument s was omitted for simplification. The diagram for Eq.(3.16) is illustrated in Fig. 3.2. To elaborate, Eq.(3.16) contains four terms for $y(s)$; one, a direct feedthrough from $u(s)$, and three others which are associated with one, two, or three integrations. The reader should be able to follow the contribution of each term to the output signal. For instance, the single-integration term, $s^{-1}(b_2 u - a_2 y)$, is incorporated into the diagram of Fig. 3.2 by feeding the signals $b_2 u(s)$ and $-a_2 y(s)$ into the final integrator block.

As in the previous case of Fig. 3.1 state variables are assigned to the outputs of the integrator elements in some order. With the assignment shown in Fig. 3.2 the resulting *observable canonical form* is given by the following structured equations:

$$\begin{bmatrix} \dot{x}_1(t) \\ \dot{x}_2(t) \\ \dot{x}_3(t) \end{bmatrix} = \begin{bmatrix} 0 & 0 & -a_0 \\ 1 & 0 & -a_1 \\ 0 & 1 & -a_2 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix} + \begin{bmatrix} b_0 - a_0 b_3 \\ b_1 - a_1 b_3 \\ b_2 - a_2 b_3 \end{bmatrix} u(t)$$

$$y(t) = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \end{bmatrix} + [\, b_3 \,] u(t) \tag{3.16}$$

In general, the *observable canonical SISO model* is given by

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{A}_o\, \mathbf{x}(t) + \mathbf{b}_o\, u(t) \\ y(t) &= \mathbf{c}_o\, \mathbf{x}(t) + d_o\, u(t) \end{aligned} \tag{3.17}$$

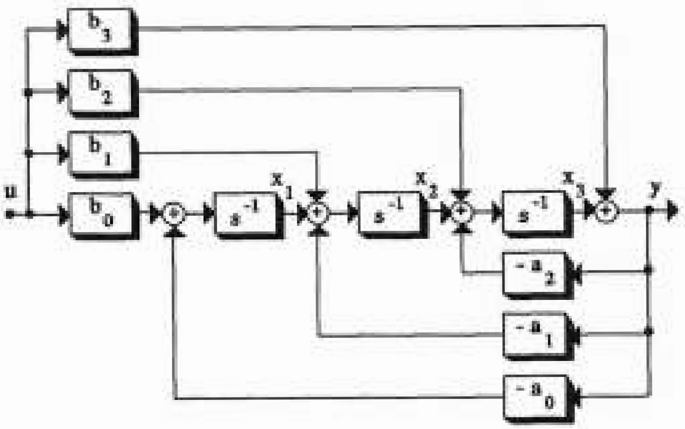


FIGURE 3.2 Observer (Observable) Canonical Form

where

$$\mathbf{A}_o = \begin{bmatrix} \mathbf{0} & -\mathbf{a} \\ \mathbf{I}_{n-1} & \end{bmatrix}, \quad \mathbf{b}_o = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix} \qquad (3.18)$$

$$\mathbf{c}_o = [\ \mathbf{0} \quad 1\ ], \quad d_o = [\,b_n\,]$$

with

$$\mathbf{a} = \begin{bmatrix} a_0 \\ \vdots \\ a_{n-1} \end{bmatrix}, \quad \text{and} \quad \mathbf{b}_i = b_{i-1} - a_{i-1} b_n \quad \text{for} \quad 0 \le i \le n-1 \qquad (3.19)$$

To reiterate, for SISO systems **A** is an $(n \times n)$ matrix, **b** is an $(n \times 1)$ column, **c** is a $(1 \times n)$ row, and d is a scalar. The canonical form Eqs.(3.18), represented in Fig. 3.2, is called the *SISO observer* (*observable*) form. As a dual property to the controllable form Eqs.(3.11), the row vector $\mathbf{c}_o$ has a unit element at the location corresponding to the column of $\mathbf{A}_o$ containing non-zero/non-unity elements.

Again, a variation of Eqs.(3.17) may be obtained by reversing the order of the state variables. Thus, with $\mathbf{x}_1$ and $\mathbf{x}_3$ interchanged in Fig. 3.2, the Eqs.(3.17) become

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} -a_2 & 1 & 0 \\ -a_1 & 0 & 1 \\ -a_0 & 0 & 0 \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} b_2 - a_2 b_3 \\ b_1 - a_1 b_3 \\ b_0 - a_0 b_3 \end{bmatrix} u(t) \tag{3.20}$$
$$y(t) = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} b_3 \end{bmatrix} u(t)$$

which presents another possible observable form.

3.1.3 The Jordan Canonical Form

The third specific form of state space representation of a system to be discussed corresponds to a diagonal, or block diagonal, coefficient matrix, which, as we will see, implies some form of decoupling of the system. This representation is referred to as the *Jordan canonical form* because the resulting coefficient matrix is that of a *Jordan canonical matrix*. Before discussing the more general Jordan form we will consider the simple, but important, case where the transfer function $g(s)$ has distinct poles. In this case the partial fraction expansion of $g(s)$ is

$$g(s) = b_3 + \frac{r_1}{s - \lambda_1} + \frac{r_2}{s - \lambda_2} + \frac{r_3}{s - \lambda_3} \tag{3.21}$$

The diagram for Eq.(3.21) is illustrated in Fig. 3.3. Note the decoupling of the dynamics into first-order blocks. With the state variables as labeled in Fig. 3.3, the corresponding state space representation may be written directly as

$$\dot{\mathbf{x}}(t) = \begin{bmatrix} \lambda_1 & 0 & 0 \\ 0 & \lambda_2 & 0 \\ 0 & 0 & \lambda_3 \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} u(t) \tag{3.22}$$
$$y(t) = \begin{bmatrix} r_1 & r_2 & r_3 \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} b_3 \end{bmatrix} u(t)$$

If the roots are not distinct, for instance, if $\lambda_1 = \lambda_2$, then the partial fraction expansion becomes

$$g(s) = b_3 + \frac{q_1}{(s - \lambda_1)^2} + \frac{r_1}{s - \lambda_1} + \frac{r_3}{s - \lambda_3} \tag{3.23}$$

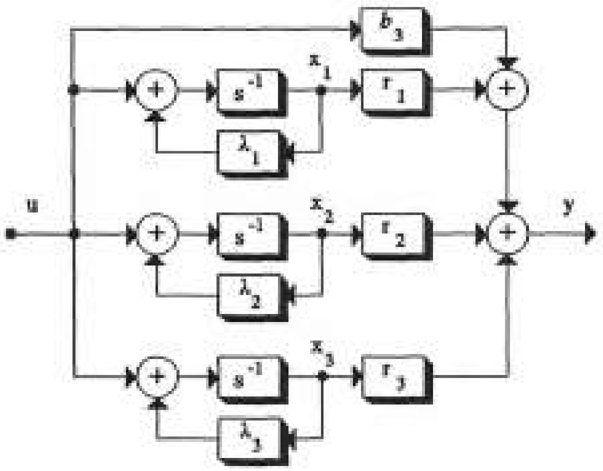


FIGURE 3.3 Block Diagram for a System with Distinct Poles

Figure 3.4 illustrates the diagram corresponding to Eq.(3.23). Note that the dynamics associated with common pole, λ_1, are separated from the distinct pole, λ_5. Thus, the Jordan form model results more generally in a "block diagonal" structure, where each block is associated with one of the distinct poles. The corresponding state equation for Fig. 3.4 is given by

$$\dot{x}(t) = \begin{bmatrix} \lambda_1 & 1 & 0 \\ 0 & \lambda_1 & 0 \\ 0 & 0 & \lambda_5 \end{bmatrix} x(t) + \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix} u(t) \tag{3.24}$$
$$y(t) = \begin{bmatrix} q_1 & r_1 & r_5 \end{bmatrix} x(t) + \begin{bmatrix} b_3 \end{bmatrix} u(t)$$

Equation (3.22) or Eq.(3.24) completes the third state model representing $g(s)$ in Eq.(3.4). Based on previous knowledge, the reader should be able to extend this third-order example to more general systems, including the extension to higher-order systems with repeated poles of degree greater than two. The Jordan form structure is discussed further in Appendices A and B. Obviously, state model representations for a given system are not unique. In the next section the concept of general state model equivalence is discussed. We summarize with some general remarks regarding Jordan form models for SISO systems.

The generalization of Eqs.(3.22) is clearly a diagonal coefficient matrix and column of ones for the input matrix, but when one or more pairs of poles (eigenvalues) are complex conjugates, it is sometimes more convenient to write the

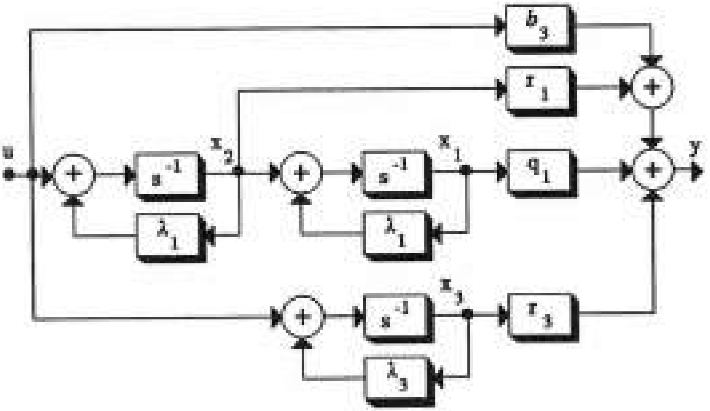


FIGURE 3.4 Block Diagram for System with Multiple Poles

coefficient matrix in a *real number Jordan form*. For example, if $\lambda_1 = \sigma_1 + j\omega_1$ and $\lambda_2 = \sigma_1 - j\omega_1$, where j represents $(-1)^{1/2}$, the real parameters, σ_1 and ω_1, are used instead of the complex values, λ_1 and λ_2, as illustrated in the following:

$$\begin{bmatrix} \lambda_1 & 0 & \cdots \\ 0 & \lambda_2 & \cdots \\ \cdots & & \end{bmatrix} = \begin{bmatrix} \sigma_1 & \omega_1 & \cdots \\ -\omega_1 & \sigma_1 & \cdots \\ \cdots & & \end{bmatrix}$$

A coefficient matrix as shown above corresponds to a partial fraction of second order with complex conjugate roots, i.e.

$$\frac{q_1 s + q_0}{(s - \sigma_1)^2 + \omega_1^2}$$

in place of

$$\frac{r_1}{s - \lambda_1} + \frac{r_2}{s - \lambda_2}$$

where for

$$r_1 = \alpha_1 + j\beta_1\,,\quad r_2 = \bar{r}_1 = \alpha_1 - j\beta_1$$
$$q_1 = 2\alpha_1\,,\quad q_2 = -2(\alpha_1\sigma_1 + \beta_1\omega_1)$$

The generalization of Eqs.(3.24) is more involved. As we know, when a matrix **A** has multiple eigenvalues, the resulting Jordan form matrix may, or may not, be diagonal. It depends on the set of linear independent eigenvectors for **A**, as discussed in Appendix A. The general structure is, however, "block diagonal", meaning that the Jordan form for **A**, say **J**, has r blocks along the diagonal, where r is the number of linearly independent eigenvectors. Thus, if **P** is the *modal matrix*, i.e. the transformation matrix relating the similar matrices **A** and **J**, then

$$\mathbf{J} = \mathbf{P}^{-1}\mathbf{A}\mathbf{P} = \begin{bmatrix} \mathbf{J}_1 & 0 & \cdots & 0 \\ 0 & \mathbf{J}_2 & \cdots & 0 \\ & \cdots & & \\ 0 & 0 & \cdots & \mathbf{J}_r \end{bmatrix}$$

where the "Jordan blocks," $\mathbf{J}_i$, $1 \le i \le r$, are associated with a single eigenvalue and may have dimension up to the multiplicity of the eigenvalue. The "block" associated with a non-repeated eigenvalue is simply the scalar eigenvalue itself. All of the nontrivial blocks have the following form:

$$\mathbf{J}_i = \begin{bmatrix} \lambda & 1 & 0 & \cdots & 0 & 0 \\ 0 & \lambda & 1 & \cdots & 0 & 0 \\ & \cdots & & & & \\ 0 & 0 & 0 & \cdots & \lambda & 1 \\ 0 & 0 & 0 & \cdots & 0 & \lambda \end{bmatrix}$$

where the same eigenvalue is repeated along the main diagonal with ones along the super diagonal.

Appendix B contains a description of a realiable algorithm for calculating the modal matrix of a general non-diagonalizable square matrix.

3.2 Equivalent State Space Models

Since we are familiar from the previous sections with the fact that the choice of state variables for a system is not unique, let us consider the conditions under which two state models represent the same system. Repeating the generic state space representation from Eqs.(1.7),

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t), \quad \mathbf{x}(t_0) \\ \mathbf{y}(t) &= \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t) \end{aligned} \tag{3.25}$$

Although Eqs.(3.25) is a C-T model, we could just as easily work with the D-T model of Eqs.(1.30) as a starting point.

Let us refer to Eqs.(3.25) as system representation S. Then any other representation must be associated with an invertible transformation of state vectors in order to uniquely relate one representation to another. We formalize with the following definition.

Definition 3.1 The C-T state model S given by

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t), \quad \mathbf{x}(t_0)$$

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

where $\mathbf{x}$ is an $(n\times 1)$ vector, $\mathbf{u}$ is an $(m\times 1)$ vector, $\mathbf{y}$ is a $(p\times 1)$ vector and the matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, and $\mathbf{D}$ are constant with compatible dimensions is said to be *equivalent* to the C-T state model Σ given by

$$\dot{\xi}(t) = \mathbf{F}\xi(t) + \mathbf{G}\mathbf{u}(t), \quad \xi(t_0)$$

$$\mathbf{y}(t) = \mathbf{H}\xi(t) + \mathbf{D}\mathbf{u}(t)$$

if and only if $\quad \xi(t_0) = \mathbf{P}^{-1}\mathbf{x}(t_0)$,

$$\mathbf{F} = \mathbf{P}^{-1}\mathbf{A}\mathbf{P}\ , \quad \mathbf{G} = \mathbf{P}^{-1}\mathbf{B}\ , \quad \text{and} \quad \mathbf{H} = \mathbf{C}\mathbf{P} \tag{3.26}$$

It should be clear that Def. 3.1 is derived from the transformation

$$\mathbf{x}(t) = \mathbf{P}\,\xi(t) \tag{3.27}$$

where $\mathbf{P}$ is required to be an $(n \times n)$ nonsingular (invertible) constant matrix.

It is easy to show that the transfer matrices of the two representations S and Σ are equal, as we would expect since they represent the same system. From Eq.(1.38), Def. 1.9, the transfer matrix of S is given by

$$G(s) = \mathbf{C}(s\mathbf{I}-\mathbf{A})^{-1}\mathbf{B} + \mathbf{D} \tag{3.28}$$

Similarly, the transfer matrix of Σ, using the results of Def. 3.1 above is

$$G(s) = (\mathbf{C}\mathbf{P})[s\mathbf{I} - (\mathbf{P}^{-1}\mathbf{A}\mathbf{P})]^{-1}(\mathbf{P}^{-1}\mathbf{B}) + \mathbf{D} \tag{3.29}$$

By introducing $\mathbf{P}^{-1}\mathbf{I}\mathbf{P}$ for the identity matrix $\mathbf{I}$ in Eq.(3.29), and factoring a $\mathbf{P}^{-1}$ to the left and a $\mathbf{P}$ to the right from the bracketed expression,

$$G(s) = (\mathbf{C}\mathbf{P})\left[\mathbf{P}^{-1}(s\mathbf{I} - \mathbf{A})\mathbf{P}\right]^{-1}(\mathbf{P}^{-1}\mathbf{B}) + \mathbf{D} \tag{3.30}$$

Simplifying the expression in brackets,

$$G(s) = (\mathbf{C}\mathbf{P})\,\mathbf{P}^{-1}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{P}\,(\mathbf{P}^{-1}\mathbf{B}) + \mathbf{D} \tag{3.31}$$

which reduces to Eq.(3.28) upon cancelation of the $\mathbf{P}^{-1}\mathbf{P}$ factors.

3.2.1 Transformations between State Models

In Def. 3.1 the representation $S_1 = \{\mathbf{A}_1, \mathbf{B}_1, \mathbf{C}_1, \mathbf{D}_1\}$ is equivalent to representation $S_2 = \{\mathbf{P}^{-1}\mathbf{A}_1\mathbf{P}, \mathbf{P}^{-1}\mathbf{B}_1, \mathbf{C}_1\mathbf{P}, \mathbf{D}_1\} = \{\mathbf{A}_2, \mathbf{B}_2, \mathbf{C}_2, \mathbf{D}_2\}$. Reversing the transformation, i.e. if $\mathbf{T} = \mathbf{P}^{-1}$, then the representations $S_2 = \{\mathbf{A}_2, \mathbf{B}_2, \mathbf{C}_2, \mathbf{D}_2\}$ and $\{\mathbf{T}\mathbf{A}_2\mathbf{T}^{-1}, \mathbf{T}\mathbf{B}_2, \mathbf{C}_2\mathbf{T}^{-1}, \mathbf{D}_2\}$ are equivalent. It is easy to show that this last representation is back to S_1. Since the *similarity*, or *equivalence*, transformation leaves the $\mathbf{D}$ matrix unchanged, it will be convenient in the sequel to represent the algorithm used for this procedure as

$$\mathbf{A}_1\ ,\ \mathbf{B}_1\ ,\ \mathbf{C}_1\ ,\ \mathbf{P}\ \ (STR)\ \rightarrow\ \mathbf{A}_2\ ,\ \mathbf{B}_2\ ,\ \mathbf{C}_2 \tag{3.32}$$

It is interesting to investigate the particular structure of $\mathbf{T}^{-1}\mathbf{A}\mathbf{T}$ for some specific transformation matrices $\mathbf{T}$ or $\mathbf{P}$. Consider the case for $\mathbf{T}$ satisfying

$$\mathbf{t}_{i+1} = \mathbf{A}\,\mathbf{t}_i\ , \quad \text{for} \quad 1 \le i \le n-1 \tag{3.33}$$

where $\mathbf{t}_i$ is the i^{th} column of $\mathbf{T}$. Therefore, with $\mathbf{t} = \mathbf{t}_1$ we may write

$$\mathbf{T} = \begin{bmatrix} \mathbf{t} & \mathbf{A}\mathbf{t} & \cdots & \mathbf{A}^{n-1}\mathbf{t} \end{bmatrix} \tag{3.34}$$

Assuming that the column $\mathbf{t}$ assures the non-singularity of $\mathbf{T}$, it may be easily shown that the structure of the transformed (similar) matrix $\mathbf{A}_s = \mathbf{T}^{-1}\mathbf{A}\mathbf{T}$ is given by

$$\mathbf{A}_s = \mathbf{T}^{-1}\mathbf{A}\mathbf{T} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & -a_0 \\ 1 & 0 & 0 & \cdots & 0 & -a_1 \\ 0 & 1 & 0 & \cdots & 0 & -a_2 \\ & & \cdots & & & \\ 0 & 0 & 0 & \cdots & 1 & -a_{n-1} \end{bmatrix} \tag{3.35}$$

Eq.(3.35) should be verified by considering the equation $\mathbf{T}\,\mathbf{A}_s = \mathbf{A}\,\mathbf{T}$.

In the dual sense, if $\mathbf{A}_s = \mathbf{T}\mathbf{A}\mathbf{T}^{-1}$ where now the *rows* $\mathbf{t}_i$ satisfy

$$\mathbf{T} = \begin{bmatrix} \mathbf{t}_1 \\ \mathbf{t}_2 \\ \cdots \\ \mathbf{t}_n \end{bmatrix} = \begin{bmatrix} \mathbf{t} \\ \mathbf{t}\,\mathbf{A} \\ \cdots \\ \mathbf{t}\,\mathbf{A}^{n-1} \end{bmatrix} \tag{3.36}$$

then $\mathbf{A}_s$ becomes

$$\mathbf{A}_s = \mathbf{TAT}^{-1} = \begin{bmatrix} 0 & 1 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 1 & 0 & \cdots & 0 \\ 0 & 0 & 0 & 1 & \cdots & 0 \\ & \cdots & & & & \\ 0 & 0 & 0 & 0 & \cdots & 1 \\ -a_0 & -a_1 & -a_2 & -a_3 & \cdots & -a_{n-1} \end{bmatrix} \tag{3.37}$$

Again, it is assumed that the row **t** assures that **T** is nonsingular.

The similarity transformation matrices **T** in Eqs.(3.34) and (3.36) may be calculated using algorithms *Qc* and *Qo* from Chapter 1, i.e.

$$\mathbf{A}\,,\,\mathbf{t}\;(Qc) \rightarrow \mathbf{T} \quad \text{and}$$
$$\mathbf{A}\,,\,\mathbf{t}\;(Qo) \rightarrow \mathbf{T}$$

respectively.

3.2.2 Controllability and Observability Forms

It may be deduced that given the SISO system $R = \{\mathbf{A}, \mathbf{b}, \mathbf{c}, d\}$, the similarity transformations

$$\mathbf{A}\,,\,\mathbf{b}\,,\,\mathbf{c}\,,\,\mathbf{Q}_c\;(STR) \rightarrow \mathbf{A}_c\,,\,\mathbf{b}_c\,,\,\mathbf{c}_c \tag{3.38}$$

and

$$\mathbf{A}\,,\,\mathbf{b}\,,\,\mathbf{c}\,,\,\mathbf{Q}_o^{-1}\;(STR) \rightarrow \mathbf{A}_o\,,\,\mathbf{b}_o\,,\,\mathbf{c}_o \tag{3.39}$$

where the (full rank) similarity transformation matrices $\mathbf{Q}_c$ and $\mathbf{Q}_o$ are calculated by

$$\mathbf{A}\,,\,\mathbf{b}\;(Qc) \rightarrow \mathbf{Q}_c \qquad \text{and} \qquad \mathbf{A}\,,\,\mathbf{c}\;(Qo) \rightarrow \mathbf{Q}_o \tag{3.40}$$

will produce the general "controllable" and "observable" models R_c and R_o described in Eqs.(3.41) and (3.42), respectively, for third order systems (and illustrated in Figs. 3.5 and 3.6). For convenience the feedthrough term is assumed to be zero, i.e. $d = 0$.

$$\begin{aligned} \dot{\mathbf{x}}_c(t) &= \begin{bmatrix} 0 & 0 & -a_0 \\ 1 & 0 & -a_1 \\ 0 & 1 & -a_2 \end{bmatrix} \mathbf{x}_c(t) + \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} u(t) \\ y(t) &= \begin{bmatrix} b_0 & b_1 & b_2 \end{bmatrix} \mathbf{x}_c(t) \end{aligned} \tag{3.41}$$

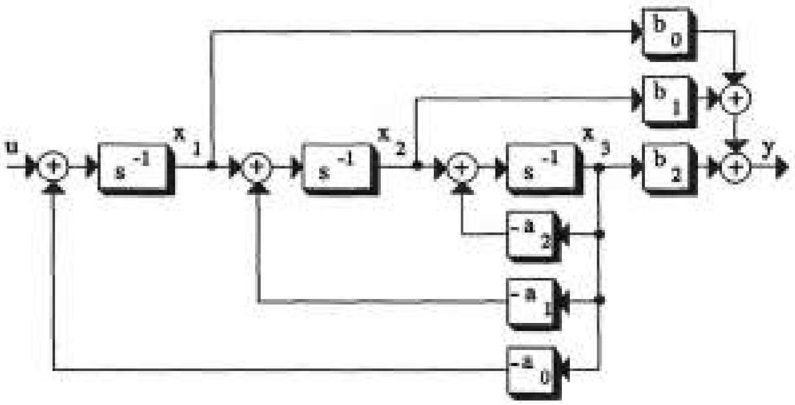


FIGURE 3.5 Controllability (Controllable) Canonical Form

$$\dot{\mathbf{x}}_o(t) = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -a_0 & -a_1 & -a_2 \end{bmatrix} \mathbf{x}_o(t) + \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} u(t) \tag{3.42}$$
$$y(t) = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \mathbf{x}_o(t)$$

Canonical forms of Eqs.(3.41) and (3.42) are called the *controllability* (*controllable*) and the *observability* (*observable*) forms, respectively. They have the important property that their controllability or observability matrix is an $n \times n$ identity matrix. As will be shown later, the *observability form*, because of its useful properties in the MIMO case, is widely used in input/output identification of D-T MIMO state space models.

Any SISO system that is controllable may be put into the forms of Eqs.(3.11), (3.15) or (3.41) since these are all *controllable forms*. Every controllable form is guaranteed to be controllable, i.e. the controllability matrix $\mathbf{Q}_c$ is full rank independent of the system parameters. For example, the controllability matrix of the form (3.41) is the identity matrix as a result of manner in which it was constructed.

Similarly, any observable system may be put into the forms of Eqs.(3.16), (3.20) or (3.42), the *observable forms*. As with the controllable forms, each observable form is observable, i.e. the observability matrix $\mathbf{Q}_o$ is full rank. As previously mentioned, the form given in Eqs.(3.42) has an observability matrix equal to an identity matrix.

If we compare Figs. 3.1 and 3.2, or Figs. 3.5 and 3.6, we notice a certain similarity of structure. In particular, they are *dual systems*. In Chapter 1, Def. 1.8, we briefly touched on the concept of duality. The equivalent block diagram

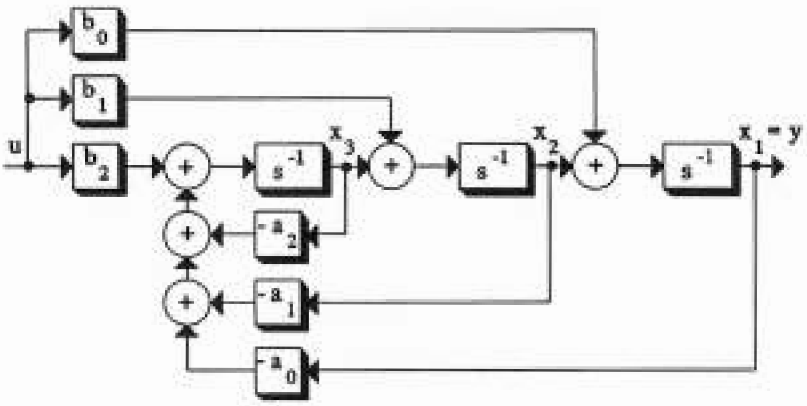


FIGURE 3.6 Observability (Observable) Canonical Form

changes necessary to construct a dual system are: to exchange input and output, reverse the order of labeling the state variables, reverse the signal flow directions and replace tap-off points and summation junctions with summations and tap-offs, respectively. Thus, for instance, Figs. 3.1 and 3.2 are dual diagrams. In the sequel three computational procedures for calculating the representations in Eqs.(3.41) and (3.42) will be presented. However, using the concept of duality, only the transformation to controllable form will be given, with the understanding that transformation of the dual system to controllable form results in the transformation of the original system to observable form. Also at this point in the text, since readers will have obtained some experience using algorithms, we will begin to present the procedures in a slightly more abreviated manner. The reason that more than one procedure is presented is that some ideas of these procedures will be used in the subsequent discussions of canonical forms for MIMO systems.

3.2.3 Transformation to Feedback Controllable Form

Procedure 1:

The problem is to determine the similarity transformation matrix **T** which will transform a given controllable representation $R = \{\mathbf{A}, \mathbf{b}, \mathbf{c}\}$ into the type of representation shown in Eqs.(3.11), i.e.,

$$\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{T}^{-1}\ (STR) \rightarrow \mathbf{A}_c, \mathbf{b}_c, \mathbf{c}_c \qquad (3.43)$$

where, according to Eq.(3.32), $\mathbf{A}_c = \mathbf{TAT}^{-1}$, $\mathbf{b}_c = \mathbf{Tb}$, and $\mathbf{c}_c = \mathbf{cT}^{-1}$.

It is desired that **T** should be of the form given by Eq.(3.36), with the first row of **T** selected to satisfy

$$\mathbf{b}_c = \mathbf{T}\,\mathbf{b} \tag{3.44}$$

where $\mathbf{b}_c$ is specified as in Eq.(3.11). In scalar form Eq.(3.44) becomes

$$\begin{aligned} 0 &= \mathbf{t}\,\mathbf{b} \\ 0 &= \mathbf{t}\,\mathbf{A}\,\mathbf{b} \\ 0 &= \mathbf{t}\,\mathbf{A}^2\,\mathbf{b} \\ &\;\cdots \\ 0 &= \mathbf{t}\,\mathbf{A}^{n-2}\,\mathbf{b} \\ 1 &= \mathbf{t}\,\mathbf{A}^{n-1}\,\mathbf{b} \end{aligned} \tag{3.45}$$

In turn, the Eqs.(3.45) may be collected into the following vector form

$$[0\;\;0\;\;0\;\cdots\;0\;\;1] = \mathbf{t}\left[\mathbf{b}\;\;\mathbf{A}\mathbf{b}\;\;\mathbf{A}^2\mathbf{b}\;\cdots\;\mathbf{A}^{n-1}\mathbf{b}\right] \tag{3.46}$$

Note that the right hand side is simply $\mathbf{t}\mathbf{Q}_c$, where $\mathbf{Q}_c$ is the controllability matrix of the given pair $\{\mathbf{A}, \mathbf{b}\}$. Thus, $\mathbf{t}$ can be calculated from Eq.(3.46) using the inverse of $\mathbf{Q}_c$ since the system is assumed to be controllable.

The following steps summarize *Procedure 1*:

1. Define a state representation $\{\mathbf{A}, \mathbf{b}, \mathbf{c}\}$
2. Set $\mathbf{A}, \mathbf{b}$ (*Qc*) $\Rightarrow \mathbf{Q}_c$
3. Partition $\mathbf{Q}_c^{-1} \Rightarrow \begin{bmatrix}\mathbf{X}\\ \mathbf{t}\end{bmatrix}$, where $\mathbf{t}$ is the last row
4. Set $\mathbf{A}, \mathbf{t}$ (*Qo*) $\Rightarrow \mathbf{T}$
5. Set $\mathbf{A}, \mathbf{b}, \mathbf{c}, \mathbf{T}^{-1}$ (*STR*) $\Rightarrow \mathbf{A}_c, \mathbf{b}_c, \mathbf{c}_c$

Procedure 2:

Again considering Eqs.(3.45), note that the first (n-1) equations can be written in the vector form

$$\mathbf{t}\left[\mathbf{b}\;\;\mathbf{A}\mathbf{b}\;\;\mathbf{A}^2\mathbf{b}\;\cdots\;\mathbf{A}^{n-2}\mathbf{b}\right] = \mathbf{0} \tag{3.47}$$

The interpretation of Eq.(3.47) is that $\mathbf{t}$ is a multiple of the transpose of the null space matrix of $(\mathbf{Q}_{c1})^T$, where $\mathbf{Q}_{c1}$ contains the first (n-1) columns of $\mathbf{Q}_c$, i.e.

$$\mathbf{t} = \mathbf{a}\,\mathbf{N}^T \;\;,\;\; \text{where} \quad \mathbf{Q}_{c1}^T\,\mathbf{N} = \mathbf{0} \tag{3.48}$$

The factor $\mathbf{a}$ in Eq.(3.48) can be determined from the last row of Eqs.(3.45) to satisfy

$$\mathbf{a}\,\mathbf{N}^T\mathbf{A}^{n-1}\mathbf{b} = 1 \tag{3.49}$$

This calculation leads to

$$\mathbf{a} = \frac{1}{\mathbf{N}^T\mathbf{q}_{cn}}$$

where $\mathbf{q}_{cn}$ is the last column from $\mathbf{Q}_c$. The following is a summary of this procedure:

1. Define a state representation $\{\mathbf{A},\ \mathbf{b},\ \mathbf{c}\}$
2. Set $\mathbf{A},\ \mathbf{b}$ $(Qc) \Rightarrow \mathbf{Q}_c$
3. Partition $\mathbf{Q}_c \Rightarrow [\mathbf{Q}_{c1}\ \ \mathbf{q}_{cn}]$, where $\mathbf{q}_{cn}$ is the last column
4. Set $\mathbf{Q}_{c1}{}^T$ $(Null) \Rightarrow \mathbf{N}$, so that $\mathbf{Q}_{c1}{}^T\,\mathbf{N} = \mathbf{0}$
5. Set $1/(\mathbf{N}^T\ \mathbf{q}_{cn}) \Rightarrow \mathbf{a}$
6. Set $\mathbf{N}^T\,\mathbf{a} \Rightarrow \mathbf{t}$
7. Set $\mathbf{A},\ \mathbf{t}$ $(Qo) \Rightarrow \mathbf{T}$
8. Set $\mathbf{A},\ \mathbf{b},\ \mathbf{c},\ \mathbf{T}^{-1}$ $(STR) \Rightarrow \mathbf{A}_c,\ \mathbf{b}_c,\ \mathbf{c}_c$

Procedure 3:

Consider the transfer function matrix $G(s)$ of a strictly proper single input, multi-output (SIMO) system $\{\mathbf{A},\ \mathbf{b},\ \mathbf{C}\}$, where $\{\mathbf{A},\ \mathbf{b}\}$ is equal to $\{\mathbf{A}_c,\ \mathbf{b}_c\}$ given by the structure of Eqs.(3.11), while $\mathbf{C} = \mathbf{I}_n$, i.e.

$$G(s) = (s\mathbf{I} - \mathbf{A}_c)^{-1}\,\mathbf{b}_c = \frac{W_c(s)}{p(s)} \tag{3.50}$$

In this case $m = 1$ and $p = n$, so that $W_c(s)$ is $(n \times 1)$, i.e. an n-dimensional column:

$$W_c(s) = \begin{bmatrix} w_{c1}(s) \\ w_{c2}(s) \\ \vdots \\ w_{cn}(s) \end{bmatrix} \tag{3.51}$$

Due to the special form of $\mathbf{A}_c$ and $\mathbf{b}_c$, it may be easily verified that the polynomials $w_{ci}(s)$ are given by

$$w_{ci}(s) = s^{i-1} \quad , \quad \text{for} \quad 1 \le i \le n \tag{3.52}$$

leading to the matrix $\mathbf{W}_c = \mathbf{I}_n$, see Algorithm *SSTF* in Section 1.3.9. The ith row of $\mathbf{W}_c$ contains all n coefficients of the n-1st order polynomial $w_{ci}(s)$.

Recall that a similarity transformation $\mathbf{P}$ does not change the transfer function. Therefore, for an arbitrary state realization and nonsingular $\mathbf{P}$, the following equation holds

$$\mathbf{c}\,(s\mathbf{I} - \mathbf{A})^{-1}\,\mathbf{b} = \mathbf{c}\,\mathbf{P}\,(s\mathbf{P}^{-1}\,\mathbf{P} - \mathbf{P}^{-1}\,\mathbf{A}\,\mathbf{P})^{-1}\,\mathbf{P}^{-1}\,\mathbf{b} \tag{3.53}$$

If $\mathbf{P} = \mathbf{T}^{-1}$ is selected to satisfy Eq.(3.43), then from Eqs.(3.50), (3.51) and (3.52), Eq.(3.53) can be written as

$$\mathbf{c}\,W(s) = \mathbf{c}\,\mathbf{P}\,W_c(s) \tag{3.54}$$

where $W_c(s)$ is defined in Eq.(3.50) and $W(s)$ is given by

$$W(s) = \mathrm{adj}\,(s\mathbf{I} - \mathbf{A})\,\mathbf{b} \tag{3.55}$$

Using the definition of the matrix $\mathbf{W}$ in the PMF introduced in Chapter 1, Eq.(3.54) may be formally written as

$$\mathbf{c}\,\mathbf{W}\begin{bmatrix} 1 \\ s \\ \vdots \\ s^{n-1} \end{bmatrix} = \mathbf{c}\,\mathbf{P}\,\mathbf{W}_c\begin{bmatrix} 1 \\ s \\ \vdots \\ s^{n-1} \end{bmatrix} \tag{3.56}$$

which, because $\mathbf{W}_c = \mathbf{I}_n$, finally leads to $\mathbf{P} = \mathbf{W}$. The following steps summarize the previous development:

```
1. Define a state representation {A, b, c}
2. Set the number of columns of A ⇒ n
3. Set $0_{n,1}$ ⇒ d
4. Set $I_{n,n}$ ⇒ I
5. Set   A, b, I, d (SSTF) ⇒ a, W
6. Partition W ⇒ [P  z], where P is $(n \times n)$
7. Set   A, b, c, P (STR) ⇒ $A_c$, $b_c$, $c_c$
```

Note that in Step 5, where the *SSTF* algorithm is executed, the characteristic polynomial $\mathbf{a}$ is not used. Only the second output argument in polynomial matrix form (PMF) is required. Since $\mathbf{d}$ in Step 3 is defined as a zero vector, the last $(n+1)$st column in $\mathbf{W}$ contains zeros, and the first n columns contained in $\mathbf{P}$, Step 6, are used as the required similarity transformation matrix.

In conclusion, the following comments are worth mentioning. The *feedback* form of Eqs.(3.11) and *observability* form of Eqs.(3.42) have the same system matrix of the structure Eq.(3.37). In other words, in Eqs.(3.11) $\mathbf{A}_c = \mathbf{TAT}^{-1}$, where $\mathbf{T}$ is the observability matrix of the pair $\{\mathbf{A},\mathbf{t}\}$, given by Eq.(3.36). In spite of the fact that an observability matrix is used, the form Eqs.(3.11) is considered to be a controllable form, since according to Eq.(3.46) the row $\mathbf{t}$ exists only if the pair {**A**,**b**} is controllable. Similarly, in Eqs.(3.42) $\mathbf{A}_o$ exists only if {**A**,**c**} is observable. In the next section we will consider the calculation of SISO canonical forms when the system transfer function is given.

3.2.4 Transformations: $g(s) \Rightarrow$ SISO Canonical Forms

The above procedures, as well as Eqs.(3.38) to (3.40), are applicable in the case when, given an arbitrary SISO representation R = {**A**,**b**,**c**,**d**}, it is required to obtain a controllable or observable form. Note that in this case it is much simpler to obtain the controllability or observability form, Eqs.(3.38) to (3.40), than the feedback or observer form. It frequently occurs, however, that given a transfer function $g(s) = b(s)/a(s)$ of a SISO system, a controllable or observable form is sought. In the sequel four algorithms for calculating:

- Feedback and observer forms, and
- Controllability and observability forms

are given, assuming that a transfer function $g(s) = b(s)/a(s)$ of a SISO system is given. These algorithms will be compared with the previously given algorithms. It will be shown that when $g(s)$ is given, the algorithms for calculating feedback and observer forms are simpler than the procedures for calculating controllability and observability forms. This discussion should also be considered as an "introduction" to the MIMO case, which is more challenging than the SISO case.

Algorithms: Algorithms $g(s)=b(s)/a(s) \Rightarrow$ Four SISO Canonical Forms

The numerator $b(s)$ and the denominator $a(s)$ are defined by:

$$b(s) = \sum_{i=0}^{n} b_i s^i \quad \text{and} \quad a(s) = \sum_{i=0}^{n} a_i s^i\,, \quad \text{with} \quad a_n = 1$$

let:

$$\mathbf{w} = \begin{bmatrix} \mathbf{b} & b_n \end{bmatrix}$$

where

$$\mathbf{b} = \begin{bmatrix} b_0 & b_1 & \cdots & b_{n-1} \end{bmatrix}$$
$$\mathbf{a} = \begin{bmatrix} a_0 & a_1 & \cdots & a_{n-1} \end{bmatrix}$$

while

$$\mathbf{s}_1 = [1 \;\; 0 \;\cdots\; 0 \;\; 0] \quad \text{and} \quad \mathbf{s}_n = [0 \;\; 0 \;\cdots\; 0 \;\; 1]$$

are the first and the nth row of an $(n \times n)$ Identity matrix $\mathbf{I}_n$. Note that

$$\mathbf{I}_n = \left[\begin{array}{c|c} \mathbf{I}_{n-1} & \mathbf{z}^T \\ \hline \mathbf{z} & 1 \end{array}\right]$$

where $\mathbf{z}$ is an appropriate row of zeros. Also, for $a_0 \neq 0$, let $d_1 = b_0/a_0$. If $a_0 \neq 0$, then $d_1 = b(s)/a(s)$ for an arbitrary s different from a root of $a(s)$, i.e. pole of $g(s)$.

The following algorithms may be used for calculating desired state space canonical forms:

1. $g(s) \Rightarrow$ **Feedback form,** $R_c = \{\mathbf{A}_c, \mathbf{b}_c, \mathbf{c}_c, d_c\}$

```
Symbolic form: a, w (RcI) ⇒ A_c, b_c, c_c, d_c

1. Set [ z^T | I_{n-1} ; --- + ---- ; -a ] ⇒ A_c
2. Set s_n^T ⇒ b_c
3. Set b - a b_n ⇒ c_c
4. Set b_n ⇒ d_c
```

$$\left[\begin{array}{c|c} \mathbf{z}^T & \mathbf{I}_{n-1} \\ \hline \multicolumn{2}{c}{-\mathbf{a}} \end{array}\right] \Rightarrow \mathbf{A}_c$$

2. $g(s) \Rightarrow$ **Observer form,** $R_o = \{\mathbf{A}_o, \mathbf{b}_o, \mathbf{c}_o, d_o\}$

```
Symbolic form: a, w (RoI) ⇒ A_o, b_o, c_o, d_o

1. Set [ z | ; ---- + -a^T ; I_{n-1} | ] ⇒ A_o
2. Set b^T - a^T b_n ⇒ b_o
3. Set s_n ⇒ c_o
4. Set b_n ⇒ d_o
```

$$\left[\begin{array}{c|c} \mathbf{z} & \\ \hline & -\mathbf{a}^T \\ \mathbf{I}_{n-1} & \end{array}\right] \Rightarrow \mathbf{A}_o$$

Obviously, due to duality: $\mathbf{A}_o = \mathbf{A}_c^T$, $\mathbf{b}_o = \mathbf{c}_c^T$, and $\mathbf{c}_o = \mathbf{b}_c^T$. Note that when b_n=0, i.e. when $g(s)$ is "strictly" proper, then:

- in R_c: $\mathbf{c}_c = \mathbf{b}$ and $d_c = 0$, while
- in R_o: $\mathbf{b}_o = \mathbf{b}^T$ and $d_o = 0$.

3. **$g(s) \Rightarrow$ Controllability form, $R_c = \{\mathbf{A}_c, \mathbf{b}_c, \mathbf{c}_c, d_c\}$**

```
Symbolic form: $\mathbf{a}, \mathbf{w}$ ($Rc2$) $\Rightarrow \mathbf{A}_c, \mathbf{b}_c, \mathbf{c}_c, d_c$

1. Set $\left[\begin{array}{c|c} \mathbf{z} & \\ \hline \mathbf{I}_{n-1} & \end{array} -\mathbf{a}^T\right] \Rightarrow \mathbf{A}_c$
2. Set $\mathbf{s}_1^T \Rightarrow \mathbf{b}_c$
3. Set $\mathbf{A}_c, \mathbf{s}_n$ ($Qo$) $\Rightarrow \mathbf{Q}_o$ ( $\mathbf{Q}_o$ has $n+1$ rows)
4. Set $\mathbf{w}\,\mathbf{Q}_o \Rightarrow \mathbf{c}_c$
5. Set $\mathbf{c}_c\mathbf{A}_c^{-1}\mathbf{b}_c + d_1 \Rightarrow d_c$
```

4. **$g(s) \Rightarrow$ Observability form, $R_o = \{\mathbf{A}_o, \mathbf{b}_o, \mathbf{c}_o, d_o\}$**

```
Symbolic form: $\mathbf{a}, \mathbf{w}$ ($Ro2$) $\Rightarrow \mathbf{A}_o, \mathbf{b}_o, \mathbf{c}_o, d_o$

1. Set $\left[\begin{array}{c|c} \mathbf{z}^T & \mathbf{I}_{n-1} \\ \hline \multicolumn{2}{c}{-\mathbf{a}} \end{array}\right] \Rightarrow \mathbf{A}_o$
2. Set $\mathbf{A}_o, \mathbf{s}_n^T$ ($Qc$) $\Rightarrow \mathbf{Q}_c$ ( $\mathbf{Q}_c$ has $n+1$ columns)
3. Set $\mathbf{Q}_c\mathbf{w}^T \Rightarrow \mathbf{b}_o$
4. Set $\mathbf{s}_1 \Rightarrow \mathbf{c}_o$
5. Set $\mathbf{c}_o\mathbf{A}_o^{-1}\mathbf{b}_o + d_1 \Rightarrow d_o$
```

Again by duality: $\{\mathbf{A}_o, \mathbf{b}_o, \mathbf{c}_o\} = \{\mathbf{A}_c^T, \mathbf{c}_c^T, \mathbf{b}_c^T\}$.

Since $\mathbf{A}_c$ and $\mathbf{A}_o$ in Algorithms *Rc2* and *Ro2* are of a simple structure, it is relatively easy to verify these algorithms. It is interesting to note that $\mathbf{A}_f$ in the feedback form is equal to $\mathbf{A}_o$ in the observability form, and also $\mathbf{A}_e$ in the observer form is equal to $\mathbf{A}_c$ in the controllability form.

Comparing these algorithms, it may be concluded that when $g(s)$ is given, it is easier to calculate feedback and observer forms than controllability and observability forms. However, recall that when an arbitrary $R = \{\mathbf{A},\mathbf{b},\mathbf{c},d\}$ is given, then it is much easier to obtain controllability and observability forms than feedback and observer forms, since the controllability and observability forms are simply derived by a similarity transformation where the controllability or observability matrix, respectively, is used as a transformation matrix.

This situation might suggest the following "alternate" procedures:

- Instead of $\mathbf{a}, \mathbf{w}$ ($Rc2$) $\Rightarrow \mathbf{A}_c, \mathbf{b}_c, \mathbf{c}_c, d_c$, one may use the sequence:

1. $\mathbf{a}$, $\mathbf{w}$ (*Rc1*) $\Rightarrow$ $\mathbf{A}_c$, $\mathbf{b}_c$, $\mathbf{c}_c$, d_c
2. $\mathbf{A}_c$, $\mathbf{b}_c$ (*Qc*) $\Rightarrow$ $\mathbf{Q}_c$
3. $\mathbf{A}_c$, $\mathbf{b}_c$, $\mathbf{c}_c$, $\mathbf{Q}_c$ (*STR*) $\Rightarrow$ $\mathbf{A}_r$, $\mathbf{b}_r$, $\mathbf{c}_r$

And according to the principle of duality, instead of $\mathbf{a}$, $\mathbf{w}$ (*Ro2*) $\Rightarrow$ $\mathbf{A}_o$, $\mathbf{b}_o$, $\mathbf{c}_o$, d_o , the sequence of Algorithms *Ro1*, *Qo* and *STR* may be used. This is left as an exercise for the reader.

On the other hand, if an arbitrary controllable realization $R = \{\mathbf{A},\mathbf{b},\mathbf{c},d\}$ is given, and the feedback form $R_c = \{\mathbf{A}_c, \mathbf{b}_c, \mathbf{c}_c, d_c\}$ is sought, then, instead of procedures discussed in Section 3.2.3, one may use the following sequence:

1. $\mathbf{A}$, $\mathbf{b}$, $\mathbf{c}$, d (*SSTF*) $\Rightarrow$ $\mathbf{a}$, $\mathbf{W}$
2. $\mathbf{a}$, $\mathbf{W}$ (*Rc1*) $\Rightarrow$ $\mathbf{A}_c$, $\mathbf{b}_c$, $\mathbf{c}_c$, d_c

Again, duality may be applied if the observer form is required. As will be seen in next sections, in the case of MIMO models the things are not as simple.

Examples Consider the 5th order, non-strictly proper transfer function $g(z) = b(z)/a(z)$ where:

- —numerator $b(z)$ coefficients b_i, $i=[0,n]$, $n=5$, are:

  .85 1.62 -4.43 -5.67 1.06 2.14

- —denominator $a(z)$ coefficients a_i, $i=[0,n]$ are:

  .12 .22 -.69 -.70 -1.34 1.00

Feedback Form [Algorithm *Rc1*]

```
|  .00    1.00     .00     .00     .00 |    .00 |
|  .00     .00    1.00     .00     .00 |    .00 |
|  .00     .00     .00    1.00     .00 |    .00 |
|  .00     .00     .00     .00    1.00 |    .00 |
| -.12    -.22     .69     .70    1.34 |   1.00 |
|--------------------------------------+-------|
|  .59    1.15   -2.95   -4.17    3.93 |   2.14 |
```

Observer Form [Algorithm *Ro1*]

```
|  .00     .00     .00     .00    -.12 |    .59 |
| 1.00     .00     .00     .00    -.22 |   1.15 |
|  .00    1.00     .00     .00     .69 |  -2.95 |
|  .00     .00    1.00     .00     .70 |  -4.17 |
|  .00     .00     .00    1.00    1.34 |   3.93 |
|--------------------------------------+-------|
|  .00     .00     .00     .00    1.00 |   2.14 |
```

Controllability Form [Algorithm *Rc2*]

```
|  .00     .00     .00     .00    -.12 |  1.00 |
| 1.00     .00     .00     .00    -.22 |   .00 |
|  .00    1.00     .00     .00     .69 |   .00 |
|  .00     .00    1.00     .00     .70 |   .00 |
|  .00     .00     .00    1.00    1.34 |   .00 |
|--------------------------------------+-------|
| 3.93    1.09    1.26    6.31    9.82 |  2.14 |
```

Observability Form [Algorithm *Ro2*]

```
|  .00    1.00     .00     .00     .00 |  3.93 |
|  .00     .00    1.00     .00     .00 |  1.09 |
|  .00     .00     .00    1.00     .00 |  1.26 |
|  .00     .00     .00     .00    1.00 |  6.31 |
| -.12    -.22     .69     .70    1.34 |  9.82 |
|--------------------------------------+-------|
| 1.00     .00     .00     .00     .00 |  2.14 |
```

In the next section we will begin to extend our modeling techniques to include multiple input, multiple output (MIMO) systems.

3.3 Canonical Forms for MIMO Systems

In order to discuss canonical forms for MIMO systems, it is first necessary to define the concept of *controllability and observability indices*. Assume a given $(n \times n)$ state matrix $\mathbf{A}$, full column rank $(n \times m)$ input matrix $\mathbf{B}$ and full row rank $(p \times n)$ output matrix $\mathbf{C}$ describing a controllable and observable system. Then, the controllability matrix $\mathbf{Q}_c$ has dimensions $(n \times nm)$ and the observability matrix $\mathbf{Q}_o$ has dimensions $(np \times n)$. Since, by assumption the system is controllable as well as observable, there must be n linearly independent columns in $\mathbf{Q}_c$ and n linearly independent rows in $\mathbf{Q}_o$. In each case a nonsingular $n \times n$ transformation matrix may be formed and used to derive the corresponding controllable or observable canonical forms.

Controllable and observable forms to be discussed in this section are MIMO versions (generalizations) of the *SISO controllability* and *SISO observability* forms calculated by Eqs.(3.38) to (3.40) and represented by Eqs.(3.41) to (3.42) and Figs. 3.5 and 3.6. Note that in the SISO case all n columns (rows) from $\mathbf{Q}_c$ ($\mathbf{Q}_o$) are used in the similarity matrices, while in the MIMO case, as we know, there are more than n columns (rows) in $\mathbf{Q}_c$ ($\mathbf{Q}_o$). Consequently, an appropriate selection of linearly independent vectors is required. In the following general discussions, the *controllability* and *observability* forms are treated separately, although there is much similarity due to the principle of duality.

3.3.1 Controllability Forms - General Discussion

A natural way to search for linearly independent columns of $\mathbf{Q}_c$ is to begin from the left, as follows:

$$\mathbf{Q}_c = \left[\mathbf{b}_1\ \mathbf{b}_2\ ...\ \mathbf{b}_m \mid \mathbf{A}\mathbf{b}_1\ ...\ \mathbf{A}\mathbf{b}_m \mid \ldots \mid \mathbf{A}^{n-1}\mathbf{b}_1\ ...\ \mathbf{A}^{n-1}\mathbf{b}_m\right] \quad (3.57)$$

Suppose that in the first q groupings of m columns each we find r_i dependent columns, $0 \le i \le q\text{-}1$. In particular, r_0 dependent columns are found in $\mathbf{B}$; r_1, in $\mathbf{AB}$; etc.. For a full rank $\mathbf{B}$, $r_0 = 0$. As a result of this choice of searching for independent columns, it is easily seen that

$$0 \le r_0 \le r_1 \le \ldots \le r_{\mu-1} < m$$

and

$$r_k = m \quad , \quad \text{for} \quad k \ge \mu$$

where μ is the smallest integer such that

$$\text{rank}\left[\mathbf{B}\quad \mathbf{AB}\ \ ...\ \ \mathbf{A}^{\mu}\mathbf{B}\right] = \text{rank}\left[\mathbf{B}\quad \mathbf{AB}\ \ ...\ \ \mathbf{A}^{\mu+1}\mathbf{B}\right] \qquad (3.58)$$

Thus, at some point there are n linearly independent columns, and all subsequent columns to the right are dependent. Notice that the controllability of the pair $\{\mathbf{A}, \mathbf{B}\}$ can be checked from $[\mathbf{B}\ \ \mathbf{AB}\ \ \mathbf{A}^2\mathbf{B}\ \ldots\ \mathbf{A}^{\mu-1}\mathbf{B}]$, where μ is less than n. The parameter μ is defined to be the *controllability index* for the system with state matrix $\mathbf{A}$ and input matrix $\mathbf{B}$.

Searching by Columns: Since there are many ways, in general, that n linearly independent columns may be chosen from $\mathbf{Q}_c$, let us introduce a convenient graphical device, called a *crate diagram* for "visualizing" the different possibilities. The *crate* is a table consisting of m columns, one for each column of the $\mathbf{B}$ matrix; and up to n rows, one for each power of $\mathbf{A}$ in $\mathbf{Q}_c$. In this manner the $(j,i)^{th}$ cell represents uniquely the column of $\mathbf{Q}_c$ given by $\mathbf{A}^{j-1}\mathbf{b}_i$. Selecting n independent columns of $\mathbf{Q}_c$ corresponds to selecting n cells in the crate. Such a diagram is illustrated in Fig. 3.7 for an m=3 input, n=7 state system. Once the basic representation is understood, we will discuss two fairly natural ways to search the crate for the required linearly independent columns. Remember that each cell represents a vector; thus, e.g. the first "row" of the crate diagram in Fig. 3.7 corresponds to the three columns of the $\mathbf{B}$ matrix (of the assumed 3-input system). First $\mathbf{b}_1$ is selected and a 1 is marked in cell (1,1). Next, continuing with the first column, $\mathbf{Ab}_1$ and $\mathbf{A}^2\mathbf{b}_1$ are considered and found to be independent, so a 1 is marked in cells (2,1) and (3,1), while $\mathbf{A}^3\mathbf{b}_1$ is found to be dependent and a 0 is marked in cell (4,1). Moving to the next column, $\mathbf{b}_2$ is added to the collection of independent vectors. Also $\mathbf{Ab}_2$, $\mathbf{A}^2\mathbf{b}_2$ and $\mathbf{A}^3\mathbf{b}_2$ are added, but not $\mathbf{A}^4\mathbf{b}_2$, since it is found to be dependent on the previously selected columns. At this juncture the required n=7 linearly independent vectors have been selected, and the process is complete. Note that in this selection plan the last column of **B** is not represented, although **B** is assumed to be full rank. This is an example of selecting the independent columns of the transformation matrix **T** for a MIMO system, which is a possible generalization of Eq.(3.38) to the MIMO case. We will refer to this method as *searching by columns*. The reader should note that the results can be widely different with a simple re-ordering of the inputs. There is a tendency to generate a few long "chains" with this method.

$\mathbf{b}_1$	$\mathbf{b}_2$	$\mathbf{b}_3$	
1	1	0	$\mathbf{A}^0$
1	1		$\mathbf{A}^1$
1	1		$\mathbf{A}^2$
0	1		$\mathbf{A}^3$
	0		$\mathbf{A}^4$

FIGURE 3.7 Search-by-Column Example of a Crate Diagram

The resulting state space model obtained by performing a similarity transformation using the collection of independent vectors found is a generalization of the SISO system of Eqs.(3.41). In particular, the crate diagram of Fig. 3.7 indicates that using the similarity transformation

$$\mathbf{T} = \begin{bmatrix} \mathbf{b}_1 & \mathbf{A}\mathbf{b}_1 & \mathbf{A}^2\mathbf{b}_1 & \mathbf{b}_2 & \mathbf{A}\mathbf{b}_2 & \mathbf{A}^2\mathbf{b}_2 & \mathbf{A}^3\mathbf{b}_2 \end{bmatrix} \tag{3.59}$$

the calculations $\qquad \mathbf{A}_c = \mathbf{T}^{-1}\mathbf{A}\mathbf{T}$ and $\mathbf{B}_c = \mathbf{T}^{-1}\mathbf{B}$

implemented by $\qquad \mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{T}\ (STR) \rightarrow \mathbf{A}_c, \mathbf{B}_c, \mathbf{C}_c$

result in the following state space structure:

$$\mathbf{A}_c = \left[\begin{array}{ccc|cccc} 0 & 0 & x & 0 & 0 & 0 & x \\ 1 & 0 & x & 0 & 0 & 0 & x \\ 0 & 1 & x & 0 & 0 & 0 & x \\ \hline 0 & 0 & x & 0 & 0 & 0 & x \\ 0 & 0 & x & 1 & 0 & 0 & x \\ 0 & 0 & x & 0 & 1 & 0 & x \\ 0 & 0 & x & 0 & 0 & 1 & x \end{array}\right], \quad \mathbf{B}_c = \left[\begin{array}{ccc} 1 & 0 & x \\ 0 & 0 & x \\ 0 & 0 & x \\ \hline 0 & 1 & x \\ 0 & 0 & x \\ 0 & 0 & x \\ 0 & 0 & x \end{array}\right] \tag{3.60}$$

where the x's denote possibly non-zero/non-unity values. The $\mathbf{C}$ matrix has no particular form.

Search by Rows: Let us now consider a similar example (of order 7 with 3-inputs) and search the crate by *rows*. Referring to Fig. 3.8, we again begin with a 1 in the (1,1) cell. Since the rank of $\mathbf{B}$ is m, both $\mathbf{b}_2$ and $\mathbf{b}_3$ are linearly independent. In the second row $\mathbf{Ab}_1$ and $\mathbf{Ab}_2$ are found to add to the collection in independent columns, but $\mathbf{Ab}_3$ is not. From the next rows only $\mathbf{A}^2\mathbf{b}_2$ and $\mathbf{A}^3\mathbf{b}_2$ are found to be linearly independent to complete the set. Once a vector $\mathbf{A}^j\mathbf{b}_i$ has been found to be linearly dependent, it is not necessary to check other vectors within the same crate column, $\mathbf{A}^k\mathbf{b}_i$, $k > j$, since they are always dependent on previously selected columns, as will be verified later. With this selection plan there is a tendency to generate shorter chains, and when $\mathbf{B}$ is full rank, all columns of $\mathbf{B}$ are represented in the selected set.

Ordering by Columns of the Crate Diagram: To obtain a controllable form, the vectors of the selected set must be arranged to form a similarity transformation matrix $\mathbf{T}$. Two specific orderings have been used. The first is by "chains" associated with a particular column of $\mathbf{B}$, i.e. by *columns* of the crate diagram (although the *selection* is done by rows). In this case $\mathbf{T}$ becomes

$$\mathbf{T} = \left[\mathbf{b}_1 \;\; \mathbf{A}\mathbf{b}_1 \;\; \mathbf{b}_2 \;\; \mathbf{A}\mathbf{b}_2 \;\; \mathbf{A}^2\mathbf{b}_2 \;\; \mathbf{A}^3\mathbf{b}_2 \;\; \mathbf{b}_3\right] \tag{3.61}$$

$\mathbf{b}_1$	$\mathbf{b}_2$	$\mathbf{b}_3$	
1	1	1	$\mathbf{A}^0$
1	1	0	$\mathbf{A}^1$
0	1		$\mathbf{A}^2$
	1		$\mathbf{A}^3$
	0		$\mathbf{A}^4$

FIGURE 3.8 Search-by-Row Example of a Crate Diagram

Using $\mathbf{A}_c = \mathbf{T}^{-1}\mathbf{AT}$ and $\mathbf{B}_c = \mathbf{T}^{-1}\mathbf{B}$ results in a state space model of the following form:

$$\mathbf{A}_c = \left[\begin{array}{cc|cccc|c} 0 & x & 0 & 0 & 0 & x & x \\ 1 & x & 0 & 0 & 0 & x & x \\ \hline 0 & x & 0 & 0 & 0 & x & x \\ 0 & x & 1 & 0 & 0 & x & x \\ 0 & 0 & 0 & 1 & 0 & x & 0 \\ 0 & 0 & 0 & 0 & 1 & x & 0 \\ \hline 0 & x & 0 & 0 & 0 & x & x \end{array}\right], \quad \mathbf{B}_c = \left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 0 & 0 \\ \hline 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \hline 0 & 0 & 1 \end{array}\right] \tag{3.62}$$

where the x's are again possibly nonzero values. And as in previous case, the **C** matrix has no particular form. The **A** and **B** given in Eq.(3.62) represent another generalization of the SISO system of Eqs.(3.41); the differences lie in the selection of the particular n columns of $\mathbf{Q}_c$ to be used in the similarity transformation.

Ordering by Rows of the Crate Diagram: Another formulation of **T** is more natural since it follows the process of selecting columns. In this case the linearly independent columns are arranged according to the unit elements in the *rows* of the crate diagram, and **T** is obtained as

$$\mathbf{T} = \left[\mathbf{b}_1 \ \ \mathbf{b}_2 \ \ \mathbf{b}_3 \ \ \mathbf{A}\mathbf{b}_1 \ \ \mathbf{A}\mathbf{b}_2 \ \ \mathbf{A}^2\mathbf{b}_2 \ \ \mathbf{A}^3\mathbf{b}_2\right] \tag{3.63}$$

which results in a state space model given by:

$$\mathbf{A}_c = \begin{bmatrix} 0 & 0 & x & x & 0 & 0 & x \\ 0 & 0 & x & x & 0 & 0 & x \\ 0 & 0 & x & x & 0 & 0 & x \\ 1 & 0 & x & x & 0 & 0 & x \\ 0 & 1 & x & x & 0 & 0 & x \\ 0 & 0 & 0 & 0 & 1 & 0 & x \\ 0 & 0 & 0 & 0 & 0 & 1 & x \end{bmatrix}, \quad \mathbf{B}_c = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{3.64}$$

The pair $\{\mathbf{A}_c,\mathbf{B}_c\}$ given in Eq.(3.62) is called a controllable *Luenberger canonical form*, while the pair $\{\mathbf{A}_c,\mathbf{B}_c\}$ in Eq.(3.64) might be called a *modified Luenberger* form, or simply a *controllability* form. The interesting property of Luenberger forms, illustrated by Eqs.(3.60) and (3.62), is that the matrix $\mathbf{A}_c$ is of "block diagonal" structure, having in the main diagonal blocks corresponding to μ_ith order

SISO systems, $i=[1,m]$. Because of this property, at its introduction this form gained instant popularity within the systems/controls community. However, as will be mentioned later, the Luenberger form does not prove to be particularly useful in specific applications.

Since the transformation matrices $\mathbf{T}$ in Eqs.(3.61) and (3.63) contain the same columns, only arranged differently, the controllable forms of Eqs.(3.62) and (3.64) are rather similar. In fact, they have the *same* elements, only arranged differently. Note that, for instance, in Eq.(3.64) the zeros at the end of columns 3 and 4 appear in Eq.(3.62) at locations 5 and 6 in columns 2 and 7. Considering the structure of Eq.(3.64) as being a *more natural* generalization of the SISO case, and, as will be shown later, more convenient for use in various applications, only the structure type of Eq.(3.64) will be used in the sequel. A perhaps stronger justification of the use of the modified form is that it is more natural (and convenient) to form the columns of $\mathbf{T}$ in the order that they are checked for linear dependence (by rows of the crate diagram) than to "rearrange" them into chains, i.e. by columns of the crate diagram.

Controllability Indices: We previously defined the *controllability index* for the pair $\{\mathbf{A},\mathbf{B}\}$ as the smallest integer, μ, such that

$$\text{rank}\begin{bmatrix}\mathbf{B} & \mathbf{AB} & \dots & \mathbf{A}^{\mu-1}\mathbf{B}\end{bmatrix} = n$$

In the previous discussion the word *chain* was used to describe the string of linearly independent vectors generated from a single column of $\mathbf{B}$ by continued multiplication by $\mathbf{A}$. Another way to view the *controllability index* is as the number of vectors in the longest chain. In this context we define the *controllability indices* (plural) as the set of integers $\{\mu_i\}$, $1 \le i \le m$, identifying the lengths of the chains of each column of $\mathbf{B}$. In terms of the crate diagram the controllability indices are the number of 1's in the columns. For instance, in the example of Fig. 3.8 the controllability indices are $\{2, 4, 1\}$. With these definitions one can see that

$$\mu = \max\{\mu_1, \mu_2, \mu_3, \cdots, \mu_m\} \tag{3.65}$$

and that

$$\mu_1 + \mu_2 + \cdots + \mu_m \le n \tag{3.66}$$

The equality holds if the system is controllable. Note that for a given pair $\{\mathbf{A},\mathbf{B}\}$ the set $\{\mu_i\}$, $1 \le i \le m$ is unique. It is noted that once a dependent column is found in a search-from-the-left process on $\mathbf{Q}_c$, then any subsequent column corresponding to that column of $\mathbf{B}$, i.e. any element in that column of the crate diagram, is also dependent on the columns of $\mathbf{Q}_c$ to its left. For example, suppose that

$$\mathbf{A}\mathbf{b}_2 = a_1\mathbf{b}_1 + \cdots + a_m\mathbf{b}_m + a_{m+1}\mathbf{A}\mathbf{b}_1 \tag{3.67}$$

then,
$$A^2 b_2 = a_1 A b_1 + \cdots + a_m A b_m + a_{m+1} A^2 b_1 \tag{3.68}$$

Likewise, $A^j b_2$, $3 \le j \le n\text{-}1$ are linearly dependent on their left-hand-side columns.

It may be shown that the set of controllability indices μ of a pair $\{A,B\}$ is invariant under any similarity transformation. However, under the permutation of columns b_i of the input matrix B, the set μ is not "completely" invariant. To be precise, it may be stated that under an arbitrary permutation of columns b_i the set of controllability indices is invariant "modulo permutation." The conditions under which the values μ_i of the set μ remain invariant are presented in detail in the references at the end of the chapter and will not be pursued here.

To summarize this subsection, any n linearly independent columns of Q_c can be used to generate a "controllability form" state space model. The subsequent discussion, corresponding to "observability form" models, will be brief, calling upon *duality* for many developments.

3.3.2 Observability Forms - General Discussion

In a manner similar to the previous discussion, we will discuss possible variations in constructing state space models which are generalizations of of the *observability* SISO observable form Eqs.(3.42). Beginning with the observability matrix Q_o, we consider the problem of searching for linearly independent rows. Recall that the dimensions of Q_o are $(np \times n)$. Assuming that the system is observable, Q_o must have rank n and, therefore, n linearly independent rows. Following Eq.(3.57), let us display Q_o:

$$Q_o = \left[(c_1)^T \cdots (c_p)^T \mid (c_1 A)^T \cdots (c_p A)^T \mid \cdots \mid (c_1 A^{n-1})^T \cdots (c_p A^{n-1})^T \right]^T$$

It is easily seen that the comments made regarding the columns of Q_c can be made for the rows of Q_o, i.e. using the concept of duality. To summarize, the *observability index* for the pair $\{A,C\}$ is the smallest integer, ν, such that

$$\operatorname{rank}\left[C^T \;\; A^T C^T \;\ldots\; (A^T)^{\nu-1} C^T \right] = n \tag{3.69}$$

Observability indices (plural) are defined as the set of integers $\{\nu_i\}$, $1 \le i \le p$, identifying the lengths of the chains of each row of C. For instance, the rows generated by row i are linearly independent up to (and including) $c_i A^{\nu_i - 1}$. With these definitions one can see that

$$\nu = \max\{\, \nu_1, \nu_2, \nu_3, \cdots, \nu_p \,\}$$
$$\text{and} \quad \nu_2 + \nu_2 + \cdots + \nu_p \le n$$

The equality above holds if the system is observable. To summarize, any n linearly independent rows of $\mathbf{Q}_o$ can be used to generate an "observability form" state space model. And, as was stated earlier, only the version which is dual to the structure given in Eqs.(3.64) will be used in the following developments.

3.3.3 Pseudo-Controllability Indices (PCI)

In generating controllable forms with complete flexibility, it is necessary to investigate all possibilities of obtaining n linearly independent columns from $\mathbf{Q}_c$. To achieve this flexibility, it has been realized that it is *not* necessary in a search-from-the-left process of $\mathbf{Q}_c$ to check *each* column $\mathbf{A}^j\mathbf{b}_i$, $j=[0,n\text{-}1]$, $i=[1,m]$. A particular column may be skipped intentionally, even if it has been found to be linearly independent with respect to the previously selected columns. However, in order to obtain a useful set of n-linearly independent columns from $\mathbf{Q}_c$, if a column $\mathbf{A}^j\mathbf{b}_i$ is skipped, then, in the spirit of Eqs.(3.67) and (3.68), all other columns $\mathbf{A}^{j+h}\mathbf{b}_i$, for $h=[1,2,\ldots]$, should be skipped, regardless of whether they are linearly dependent, or not. It has been verified that under this "selection method," the total number of combinations to check is k, given by

$$k = \binom{n-1}{m-1} = \frac{(n-1)!}{(m-1)!\,(n-m)!} \tag{3.70}$$

We say that a particular selection of n columns is *admissible* if they are linearly independent.

Since, for a given pair $\{\mathbf{A},\mathbf{B}\}$, there are now more sets of linearly independent columns, there are consequently more sets of integers $\{\mu_i\}$ indicating the lengths of chains $\mathbf{A}^j\mathbf{b}_i$, $j=[0,\mu_i\text{-}1]$, for each $\mathbf{b}_i$. These sets are referred to as *admissible sets of pseudo-controllability indices (PCI)*. This same concept of PCI is also called, by some authors, *nice indices*. To formalize the ideas, let us define the following:

Definition 3.1 The *set of individual controllability indices*, $\{\alpha_i\}$, $1 \le i \le m$, is defined by

$$\alpha_i = \text{rank}\left[\mathbf{b}_i \ \ \mathbf{A}\mathbf{b}_i \ \ \cdots \ \ \mathbf{A}^{n-1}\mathbf{b}_i\right] \tag{3.71}$$

where $\mathbf{b}_i$ is the i^{th} column of the matrix $\mathbf{B}$.

For convenience we will use a notation similar to the controllability indices of Eq.(3.65), since the concept of pseudo-controllability indices is a generalization of the notion of "standard" unique controllability indices discussed in the previous section.

Definition 3.2 The *set of pseudo-controllability indices*, $\{\mu_i\}$, $1 \le i \le m$, is any set of numbers satisfying

$$1 \le \mu_i \le n - m + 1 \ , \ \text{and} \sum_{i=1}^{m} \mu_i = n \tag{3.72}$$

Definition 3.3 The set of pseudo-controllability indices, $\{\mu_i\}$, $1 \le i \le m$, is *admissible* if

(3.73)

It has been shown that an element μ_i of an admissible set satisfies:

$$\mu_i \le \alpha_i \tag{3.74}$$

If we make the reasonable assumption that **B** is full rank, i.e. that the columns of **B** are linearly independent, then the *individual controllability indices* are each constrained to be between 1 and n; while each *pseudo-controllability index* of an admissible set $\{\mu_i\}$ is a number between 1 and $(n-m+1)$. If **B** is not full rank, an input transformation may be performed to eliminate the "redundant" input(s).

3.3.4 Pseudo-Observability Indices (POI)

Pseudo-observability indices are used to establish *observability form* state space models. We will use the notation of the set $\{\nu_i\}$ in referring to either the unique set of observability indices or a set of admissible pseudo-observability indices. This is justified by the fact that the unique set of observability indices is always one of the sets of POI. The same is true of the unique set of controllability indices being a member of the PCI. Since this section provides an important background for subsequent chapters, a detailed description is presented. A specific example will help to illustrate the concept.

Consider a system with order $n=7$, m=2 inputs and $p=3$ outputs. We are not interested specifically in the unique set of observability indices, but suppose that the set of unique observability indices is given by

$$\nu = \{ \nu_i \} = \{ 3, 2, 2 \}$$

As we will show soon, the use of this unique set of observability indices does not necessarily lead to the most convenient system representation. Taking into account that the use of admissible sets of pseudo-observability indices offers more flexibility in choosing the appropriate model, in the sequel we will pursue the selection of the most convenient set of (pseudo) observability indices.

Knowing that the system order is 7 and that the number of outputs is 3, there are several possible observable form structures that may be considered. According to Eq.(3.70), the total number of sets of pseudo-observable indices $\{\nu_i\}$ is 15. Specifically, the following combinations are possible:

$$\begin{bmatrix} \nu_1 \\ \nu_2 \\ \nu_3 \end{bmatrix} = \begin{bmatrix} 5 & 4 & 4 & 3 & 3 & 3 & 2 & 2 & 2 & 2 & 1 & 1 & 1 & 1 & 1 \\ 1 & 2 & 1 & 3 & 2 & 1 & 4 & 3 & 2 & 1 & 5 & 4 & 3 & 2 & 1 \\ 1 & 1 & 2 & 1 & 2 & 3 & 1 & 2 & 3 & 4 & 1 & 2 & 3 & 4 & 5 \end{bmatrix}$$

However, according to Eq.(3.70), the number of admissible sets is less than, or equal to 15. To simplify the discussion, we will only consider the following sets of possible POI with the assumption that they are admissible.

Case	1	2	3
Pseudo-Observability Indices	{3,3,1}	{3,1,3}	{1,3,3}

Note that in each case the "observability indices" sum to n=7. We can use a crate diagram to represent each of these three cases. The crate's column entries correspond to rows of $\mathbf{Q}_o$ associated with a particular output. Both here and in subsequent chapters the (reasonable) assumption is made that the outputs, i.e. rows of **C**, are linearly independent. Consequently, the first row of the crate is always selected.

{3,3,1}		
1	1	1
1	1	0
1	1	
0	0	

{3,1,3}		
1	1	1
1	0	1
1		1
0		0

{1,3,3}		
1	1	1
0	1	1
	1	1
	0	0

Crate diagrams are simply a graphical method of visualizing the selection of linearly independent rows from the given observability matrix. For example, with the columns of the crate being associated with a particular row $\mathbf{c}_i$ of $\mathbf{C}$, the center crate above indicates that, among the possible choices of n linearly independent rows from $\mathbf{Q}_o$, the independent elements selected are the rows:

$$\mathbf{c}_1,\ \mathbf{c}_2,\ \mathbf{c}_3,\ \mathbf{c}_1\mathbf{A},\ \mathbf{c}_3\mathbf{A},\ \mathbf{c}_1\mathbf{A}^2,\text{ and }\mathbf{c}_3\mathbf{A}^2.$$

From the crate diagrams several related "selector vectors" are generated:

- By omitting the first row of, say the center diagram, corresponding to the indices {3,1,3}, the vector $\mathbf{v}_i$ is created by selecting the non-blank elements row-wise:

$$\mathbf{v}_i = [\,1 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad 0\,]^T \tag{3.75}$$

- From $\mathbf{v}_i$ the binary complement is formed, and denoted as $\mathbf{v}_d$:

$$\mathbf{v}_d = [\,0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1\,]^T \tag{3.76}$$

- By considering the blank elements to be zeros, $\mathbf{v}_{ii}$ is formed in like manner, but with row 1 included:

$$\mathbf{v}_{ii} = [\,1 \quad 1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0\,]^T \tag{3.77}$$

- Finally, $\mathbf{v}_{id}$ is formed by again including the first row, but now taking the blank elements of the diagram to be unit valued, and finally taking the binary complement, leading to:

$$\mathbf{v}_{id} = [\,0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1\,]^T \tag{3.78}$$

The above selector vectors are uniquely determined by the particular set of pseudo-observability indices, or equivalently, the location of the unity elements in the corresponding crate diagram. As will be shown later, these selector vectors greatly facilitate calculation of the observable forms based on the chosen set of observability indices. In particular, the "selector matrices" given in Eqs.(3.79) below, derived from the associated selector vectors by a corresponding selection of columns from an appropriately dimensioned identity matrix, are actually used in obtaining the observable form.

$$\mathbf{S}_i = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}^T , \quad \mathbf{S}_{li} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}^T \tag{3.79}$$

$$\mathbf{S}_a = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}^T , \quad \mathbf{S}_{ld} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}^T$$

The selection of rows (or columns) of a matrix may be accomplished by a pre- (or post-) multiplication by a corresponding "selector" matrix. Thus, for instance, since $\mathbf{S}_i$ in Eq.(3.79) is a (7×4) selector matrix, the product $\mathbf{S}_i^T\mathbf{M}$, where $\mathbf{M}$ is a (7×7) matrix, results in the "selection" of rows 1, 3, 4 and 5 from $\mathbf{M}$ into the (4×7) product. It will be clear in a later development how useful the selector matrices are in the formulation of various algorithms to be discussed.

To facilitate further discussion, the dependence of the above selector matrices on the set of indices ν will be formally represented by the following algorithm:

$$\nu\ (SMat) \rightarrow \nu_m\, , \mathbf{S}_a\, , \mathbf{S}_i\, , \mathbf{S}_{li}\, , \mathbf{S}_{ld} \quad \text{where} \quad \nu_m = \max\{\, \nu_i \,\}$$

In the subsequent discussion we will relate the crate diagram, selector vectors and selector matrices to the structural properties of a state space observable form R_o = $\{\mathbf{A}_o, \mathbf{B}_o, \mathbf{C}_o, \mathbf{D}_o\}$. It will be shown that for the {3,1,3} example from above matrices $\mathbf{C}_o$ and $\mathbf{A}_o$ have the following structure:

$$\mathbf{A}_o = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ x & x & x & x & x & x & x \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ x & x & x & x & x & x & x \\ x & x & x & x & x & x & x \end{bmatrix} \qquad \mathbf{C}_o = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{3.80}$$

The structure of the pair $\{\mathbf{A}_o, \mathbf{C}_o\}$ is characterized by the following points:

- $\mathbf{C}_o$ consists of the first $p=3$ rows of the $(n\times n)$ identity matrix $\mathbf{I}_n$.
- At locations specified by the unities in the selector vector $\mathbf{v}_l$, the matrix $\mathbf{A}_o$ contains the last n-p = 4 rows of $\mathbf{I}_n$.
- At locations specified by the $p=3$ unities in the selector vector $\mathbf{v}_a$, the matrix $\mathbf{A}_o$ contains rows of elements which are not necessarily of zero or unit value.
- The "observability matrix" $\mathbf{Q}_{oo}$ of the pair $\{\mathbf{A}_o, \mathbf{C}_o\}$, i.e.

$$\mathbf{Q}_{oo} = \left[\, \mathbf{C}_o^T \quad (\mathbf{C}_o\mathbf{A}_o)^T \quad \cdots \quad (\mathbf{C}_o\mathbf{A}_o^{\nu})^T \,\right]^T \tag{3.81}$$

  contains all n rows of $\mathbf{I}_n$ at locations specified by the $n=7$ unities in the selector vector $\mathbf{v}_{ll}$.
- The $p=3$ rows of $\mathbf{A}_o$ containing not necessarily zero or unit elements appear in $\mathbf{Q}_{oo}$ at locations specified by the unities in the selector vector $\mathbf{v}_{ld}$.

The results of Eqs.(3.80) derive from the basic similarity transformation, or change of state,

$$\begin{aligned} \mathbf{A}_o &= \mathbf{TAT}^{-1}\,, & \mathbf{B}_o &= \mathbf{TB} \\ \mathbf{C}_o &= \mathbf{CT}^{-1}\,, & \mathbf{D}_o &= \mathbf{D} \end{aligned} \tag{3.82}$$

where R = {$\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$} is an arbitrary n^{th} order observable state space representation. In order to obtain $\mathbf{A}_o$ and $\mathbf{C}_o$ given by Eqs.(3.80), the transformation matrix $\mathbf{T}$ in Eq.(3.82), corresponding to the pseudo-observability indices {3,1,3}, is given by

$$\mathbf{T} = \left[\, \mathbf{c}_1^T \;\; \mathbf{c}_2^T \;\; \mathbf{c}_3^T \;\; (\mathbf{c}_1\mathbf{A})^T \;\; (\mathbf{c}_3\mathbf{A})^T \;\; (\mathbf{c}_1\mathbf{A}^2)^T \;\; (\mathbf{c}_3\mathbf{A}^2)^T \,\right]^T \tag{3.83}$$

It may be verified that all $n=7$ rows of $\mathbf{T}$ are located in the observability matrix $\mathbf{Q}_o$ of the pair {$\mathbf{A}$, $\mathbf{C}$}, i.e.

$$\mathbf{Q}_o = \left[\, \mathbf{C}^T \quad (\mathbf{CA})^T \quad \cdots \quad (\mathbf{CA}^{\nu})^T \,\right]^T$$

at locations specified by $n=7$ unities in the selector vector $\mathbf{v}_{ll}$, where ν = 3 is the maximum length chain. The algorithmic representation of Eqs.(3.82) and (3.83) is

$$\nu\ (SMat) \rightarrow \nu_m\,,\ \mathbf{S}_a\,,\ \mathbf{S}_l\,,\ \mathbf{S}_{ll}\,,\ \mathbf{S}_{ld}$$

$$\mathbf{A}\,,\ \mathbf{C}\ (Qo) \rightarrow \mathbf{Q}_o$$

$$\mathbf{S}_{ll}^T\mathbf{Q}_o \rightarrow \mathbf{T}$$

$$\mathbf{A}\,,\ \mathbf{B}\,,\ \mathbf{C}\,,\ \mathbf{T}^{-1}\ (STR) \rightarrow \mathbf{A}_o\,,\ \mathbf{B}_o\,,\ \mathbf{C}_o$$

Note that the structure of Eqs.(3.80) is *dual* to the controllable form given in

Eq.(3.64), with the understanding that the sets of indices used in building these forms are different. To emphasize the fact that MIMO controllability and observability forms are not unique, and that they are based on sets of admissible PCI and POI, these forms sometimes will be referred to as pseudo-controllability (PCF) and pseudo-observability forms (POF).

3.3.5 MIMO Feedback and Observer Forms

In discussing feedback and observer forms for SISO systems (Sections 3.1.1 and 3.1.2) it has been mentioned that these forms provide an extremely useful method of obtaining state space equations from a given transfer function. Due to this property, SISO feedback and observer forms gained great popularity. In the case of MIMO systems, however, these forms are not particularly popular. The following discussion will give more insight into this lack of popularity.

These forms are applicable only for solving state feedback pole placement and full and reduced-order observer design problems. However, efficient algorithms have been recently developed which solve these problems directly, using the given state space representation, without the necessity of calculating canonical forms explicitly.

Also, for a given pair $\{\mathbf{A},\mathbf{B}\}$ with unique set of controllability indices $\mu = \{ \mu_i \}$, the feedback form has a unique structure. In other words, there is no flexibility even with the use of pseudo-controllability indices, as compared with the case of controllability and observability forms discussed in Sections 3.3.1 - 3.3.4. Specifically, as will be shown by examples, the structure of the matrix $\mathbf{A}_c$ in the feedback form $\{\mathbf{A}_c,\mathbf{B}_c\}$ is based on the set $\bar{\mu}$ obtained by ordering the set μ of controllability indices of $\{\mathbf{A},\mathbf{B}\}$ in ascending order. Thus, $\mathbf{A}_c$ is of the same structure for pairs $\{\mathbf{A},\mathbf{B}\}$ with different controllability indices provided that the sets of controllability indices of these pairs have the same "ordered" set $\bar{\mu}$. The structure of the matrix $\mathbf{B}_c$, however, reflects actual controllability indices μ of the given pair $\{\mathbf{A},\mathbf{B}\}$.

For these reasons the discussion of these forms will be relatively brief. Also, only the feedback form will be discussed, since the observer form could be obtained by invoking the principle of duality. The main properties of MIMO feedback forms $\{\mathbf{A}_c,\mathbf{B}_c\}$ are as follows:

The matrix $\mathbf{A}_c$ has m rows with possibly non-zero elements. Locations of these rows are determined by unities in the selector vector $\bar{\mathbf{v}}_a$ generated by the set of controllability indices $\bar{\mu}$ obtained by ordering the set of controllability indices μ of the given pair $\{\mathbf{A},\mathbf{B}\}$ in ascending order. Similar to $\mathbf{A}_o$ in Eqs.(3.80), the remaining n-m rows contain the last n-m rows of an $(n \times n)$ identity matrix $\mathbf{I}_n$. Unlike the matrix $\mathbf{C}_o$ in Eqs.(3.80), which is always of the same structure, the $(n \times m)$ matrix $\mathbf{B}_c$ has non-zero elements in the same m rows determined by the m unities in the selector vector $\bar{\mathbf{v}}_a$. Moreover, as will be shown by examples, these

m rows are given by a particular permutation of the rows of the following $(m \times m)$ upper triangular, nonsingular matrix:

$$\bar{\mathbf{B}}_m = \begin{bmatrix} 1 & x & x & \cdots & x \\ & 1 & x & \cdots & x \\ & & \ddots & & \\ & & & 1 & x \\ & & & & 1 \end{bmatrix} \tag{3.84}$$

where x represents a possible non-zero quantity. Specifically, if the actual controllability indices are already ordered in ascending order, then the i^{th} non-zero row of $\mathbf{B}_c$ is equal to the i^{th} row of $\bar{\mathbf{B}}_m$. In the general case, however, the i^{th} non-zero row of $\mathbf{B}_c$ is equal to the i^{th} row of the product:

$$\mathbf{B}_m = \mathbf{T}\,\bar{\mathbf{B}}_m\,, \quad \text{where} \quad \bar{\mu} = \mu\,\mathbf{T} \tag{3.85}$$

where μ and $\bar{\mu}$ are rows containing actual and ordered controllability indices, respectively. For example, if $\mu = [\,2\ 1\ 2\,]$ and $\bar{\mu} = [\,1\ 2\ 2\,]$, then $\mathbf{T}$ and $\mathbf{B}_m$ are:

$$\mathbf{T} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{B}_m = \begin{bmatrix} 0 & 1 & x \\ 1 & x & x \\ 0 & 0 & 1 \end{bmatrix} \tag{3.86}$$

It might be of some interest to mention that in $\bar{\mathbf{B}}_m$, Eq.(3.84), the value of x at locations (i,j), $i < j$, is zero if in the ordered set $\bar{\mu}$, $\bar{\mu}_i < \bar{\mu}_j$. To be specific in the cases of μ given by:

$\{2\ 1\ 2\}$ and $\{2\ 2\ 1\}$, both leading to the set $\bar{\mu} = \{1\ 2\ 2\}$,

matrices $\bar{\mathbf{B}}_m$ are $\begin{bmatrix} 1 & x & 0 \\ & 1 & 0 \\ & & 1 \end{bmatrix}$ and $\begin{bmatrix} 1 & 0 & x \\ & 1 & x \\ & & 1 \end{bmatrix}$ yielding for matrices $\mathbf{T}$ and $\mathbf{B}_m$:

$$\mathbf{T} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{B}_m = \begin{bmatrix} 0 & 1 & 0 \\ 1 & x & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

and

$$\mathbf{T} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}, \quad \mathbf{B}_m = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & x \\ 0 & 1 & x \end{bmatrix}$$

respectively.

The reason for insisting on the structure of $\mathbf{B}_m$ and $\hat{\mathbf{B}}_m$ for different sets μ which are equal modulo permutation, is to stress that the structure of $\mathbf{A}_c$ is the same and that a particular "distribution" of values μ_i in the set μ is reflected in the non-zero rows of matrix $\mathbf{B}_c$, i.e. m rows of $\mathbf{B}_m$.

In the sequel two algorithms for calculating the pair $\{\mathbf{A}_c, \mathbf{B}_c\}$ in the feedback form will be given. It is easy to verify that these two algorithms are MIMO generalizations of SISO algorithms given in Sec. 3.2. We will refer to them as Procedures 1 and 2. The dual version of the Procedure 1, calculating the observer form $\{\mathbf{A}_o, \mathbf{B}_o\}$ for multi-output systems is also given. It is worth mentioning that there is no MIMO generalization of Procedure 3, Eqs.(3.50) - (3.56), based on using the $(n \times n)$ matrix $\mathbf{P}$ containing coefficients w_{ik} of the $(n\text{-}1)^{\text{th}}$ order polynomials $w_i(s)$ in the transfer function matrix (column) $W(s)$ of the following single-input n-output system:

$$W(s) = \mathbf{C}\,\mathrm{adj}(s\mathbf{I} - \mathbf{A})\,\mathbf{b}\ , \quad \text{where} \quad \mathbf{C} = \mathbf{I}_n$$

Procedure 1: Feedback controllable form for multi-input system

1. Define state space representation $\{\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D}\}$
2. Set $\mathbf{A},\mathbf{B}$(*Qc*) $\Rightarrow$ $\mathbf{Q}_c$; Set number of columns in $\mathbf{B} \Rightarrow m$
3. Set $\mathbf{Q}_c$ (*IND*) $\Rightarrow$ μ_c, unique controllability indices
4. Set μ_c (*SMat*) $\Rightarrow$ $\mathbf{S}_a,\mathbf{S}_i,\mathbf{S}_{ij},\mathbf{S}_M$
5. Set $\mathbf{Q}_c\,\mathbf{S}_{ij} \Rightarrow \mathbf{Q}_{cr}$
6. Set $\mathbf{Q}_{cr}^{-1} \Rightarrow \mathbf{Q}_{cri}$
7. Set $\mathbf{S}_a^T\,\mathbf{Q}_{cri} \Rightarrow \mathbf{C}_a$
8. Set $\mathbf{A},\mathbf{C}_a$ (*Qo*) $\Rightarrow$ $\mathbf{Q}_{oa}$
9. Set the first linearly independent rows from $\mathbf{Q}_{oa} \Rightarrow \mathbf{T}_c$
10. Set $\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{T}_c^{-1}$ (*STR*) $\Rightarrow$ $\mathbf{A}_c,\mathbf{B}_c,\mathbf{C}_c$

Procedure 1: Observer observable form for multi-output system

1. Define state space representation $\{\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D}\}$
2. Set $\mathbf{A},\mathbf{C}$(*Qo*) $\Rightarrow$ $\mathbf{Q}_o$; Set number of columns in $\mathbf{C} \Rightarrow p$
3. Set $\mathbf{Q}_o$ (*IND*) $\Rightarrow$ ν_o, unique observability indices
4. Set ν_o (*SMat*) $\Rightarrow$ $\mathbf{S}_a,\mathbf{S}_i,\mathbf{S}_{ij},\mathbf{S}_M$
5. Set $\mathbf{S}_{ij}^T\,\mathbf{Q}_o \Rightarrow \mathbf{Q}_{or}$
6. Set $\mathbf{Q}_{or}^{-1} \Rightarrow \mathbf{Q}_{ori}$

```
 7. Set $\mathbf{Q}_{cri}\ \mathbf{S}_s \Rightarrow \mathbf{B}_a$
 8. Set $\mathbf{A},\mathbf{B}_a$ $(Qc) \Rightarrow \mathbf{Q}_{ca}$
 9. Set the first linearly independent columns from $\mathbf{Q}_{ca} \Rightarrow \mathbf{T}_o$
10. Set $\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{T}_o$ $(STR) \Rightarrow \mathbf{A}_o,\mathbf{B}_o,\mathbf{C}_o$
```

Procedure 2: Feedback controllable form for multi-input system (obtained by calculating null spaces of some columns of $\mathbf{Q}_c$)

```
 1. Define state space representation $\{\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D}\}$
 2. Set $\mathbf{A},\mathbf{B}(Qc) \Rightarrow \mathbf{Q}_c$; Set number of columns in $\mathbf{B} \Rightarrow m$
 3. Set $\mathbf{Q}_c$ $(IND) \Rightarrow \mu_c$, unique controllability indices
 4. Set $\mu_c$ $(SMat) \Rightarrow \mathbf{S}_a,\mathbf{S}_i,\mathbf{S}_b,\mathbf{S}_M$
 5. Set $\mathbf{Q}_c\ \mathbf{S}_s \Rightarrow \mathbf{Q}_{cr}$
 6. Set number of columns in $\mathbf{A} \Rightarrow n$
 7. Set $\mathbf{O}_{0,n} \Rightarrow \mathbf{C}_a$
 8. Set $[\ 1\ \ldots\ 1\ ] \Rightarrow \mathbf{I}_r$; $\mathbf{I}_r = n$ unities
 9. Set $\mathbf{S}_a^T \Rightarrow \mathbf{S}$
10. Set $0 \Rightarrow i$
11. Set $i+1 \Rightarrow i$
12. Set $\mathbf{I}_r$ - $i^{th}$ row of $\mathbf{S} \Rightarrow \mathbf{v}$
13. Set $\mathbf{v}$ $(DSM) \Rightarrow \mathbf{S}_{ai}$
14. Set $\mathbf{Q}_{cr}\ \mathbf{S}_{ai} \Rightarrow \mathbf{M}_i$
15. Set $\mathbf{Q}_{cr}\ \mathbf{v}^T \Rightarrow \mathbf{q}_i$
16. Set null space of $\mathbf{M}_i^T \Rightarrow \mathbf{t}_i$, row $\mathbf{t}_i^T\,\mathbf{M}_i = \mathbf{0}$
17. Set $\mathbf{t}_i^T\ \mathbf{q}_i \Rightarrow \alpha_i$
18. Set $\mathbf{t}_i^T/\alpha_i \Rightarrow \mathbf{c}_{ai}$
19. Set $\begin{bmatrix} \mathbf{C}_a \\ \mathbf{c}_{ai} \end{bmatrix} \Rightarrow \mathbf{C}_a$
20. If $i < m$, go to 11; else, go to 21
21. Set $\mathbf{A},\mathbf{C}_a$ $(Qo) \Rightarrow \mathbf{Q}_{oa}$
22. Set the first linearly independent rows from $\mathbf{Q}_{oa} \Rightarrow \mathbf{T}_c$
23. Set $\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{T}_c^{-1}$ $(STR) \Rightarrow \mathbf{A}_c,\mathbf{B}_c,\mathbf{C}_c$
```

The algorithm for the observer form based on calculating null spaces of some rows from $\mathbf{Q}_o$ is left as an exercise for reader.

In the first 5 steps of the algorithm: Procedure 1, feedback form, the first n linearly independent columns from $\mathbf{Q}_c$ are selected in $\mathbf{Q}_{cr}$ and the selector matrix $\mathbf{S}_a$ has been generated. Then the "auxiliary" $(m \times n)$ output matrix $\mathbf{C}_a$ is calculated by:

$$\mathbf{C}_a = \mathbf{S}_a^T\mathbf{Q}_{cr}^{-1}$$

i.e. the m rows of $\mathbf{C}_a$ are contained in $\mathbf{C}_{cr}^{-1}$ at locations determined by m unities in the selector vector $\mathbf{v}_a$, which generates the selector matrix $\mathbf{S}_a$. Finally, the similarity transformation matrix $\mathbf{T}_c$ used in Step 10 to calculate $R_c = \{\mathbf{A}_c, \mathbf{B}_c, \mathbf{C}_c\}$, obtained by selecting the first n linearly independent rows from the auxiliary observability matrix $\mathbf{Q}_{oa}$ of the pair $\{\mathbf{A},\mathbf{C}_a\}$. It may be shown by inspection that the observability indices of the "auxiliary" pair $\{\mathbf{A},\mathbf{C}_a\}$ are equal to the controllability indices of $\{\mathbf{A},\mathbf{B}\}$ ordered in ascending order, i.e. to $\bar{\boldsymbol{\mu}}$.

In the algorithm of Procedure 2, feedback controllable form, the first 5 steps are exactly the same as the first 5 steps of Procedure 1. By Steps 6 through 20, the auxiliary $(m \times n)$ output matrix $\mathbf{C}_a$ is calculated without explicitly calculating the inverse of $\mathbf{Q}_{cr}$. To visualize how this is done, consider a pair $\{\mathbf{A},\mathbf{B}\}$ with controllability indices $\boldsymbol{\mu} = \{2\ 1\ 2\}$ leading to the selector vector $\mathbf{v}_a$ and selector matrix $\mathbf{S}_a$ given by:

$$\mathbf{v}_a = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 \end{bmatrix}, \quad \mathbf{S}_a = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix}^T$$

In our example the first $n = 5$ linearly independent columns from $\mathbf{Q}_c$ are

$$\mathbf{Q}_{cr} = \begin{bmatrix} \mathbf{b}_1 & \mathbf{b}_2 & \mathbf{b}_3 & \mathbf{Ab}_1 & \mathbf{Ab}_3 \end{bmatrix} \triangleq \begin{bmatrix} x & \mathbf{q}_1 & x & \mathbf{q}_2 & \mathbf{q}_3 \end{bmatrix}$$

For convenience, by $\mathbf{q}_i$, $i=[1,m]$, $m=3$, are denoted columns in $\mathbf{Q}_{cr}$ whose locations correspond to unities in the selector vector $\mathbf{v}_a$. Recall that these columns correspond to the "end-of-chain" columns, i.e. to $\mathbf{b}_2$, $\mathbf{Ab}_1$ and $\mathbf{Ab}_3$. Note that these columns are "associated" with the m columns $\mathbf{b}_i$ of the matrix $\mathbf{B}$ in the order determined by the *ordered* controllability indices $\bar{\mu} = \{1\ 2\ 2\}$, corresponding to actual indices: $\mu_2 = 1$ and $\mu_1 = \mu_3 = 2$. Then, ith row $\mathbf{c}_{ai}$, $i=[1,m]$, of $\mathbf{C}_a$ is calculated by:

$$\mathbf{c}_{ai} = \frac{N(\mathbf{M}_i^T)^T}{N(\mathbf{M}_i^T)^T \mathbf{q}_i} \tag{3.87}$$

where $N(\mathbf{X}) = \mathbf{N}$ represents the null space of $\mathbf{X}$, satisfying $\mathbf{X}\,\mathbf{N} = \mathbf{0}$.

In Eq.(3.87) the $(n \times n\text{-}1)$ matrix $\mathbf{M}_i$ is obtained from $\mathbf{Q}_{cr}$ by eliminating column $\mathbf{q}_i$, $i=[1,m]$. Finally, having calculated rows $\mathbf{c}_{ai}$, i.e. the auxiliary matrix $\mathbf{C}_a$, the last 3 steps of this algorithm are equal to the last 3 steps of the algorithm implementing Procedure 1. It is relatively easy to verify that in the SISO case these two algorithms (Procedures 1 and 2) reduce to the algorithms (Procedure 1 and 2) given in Section 3.2.3.

3.3.6 Modeling Example

To illustrate usefulness and advantages of controllability and observability forms over the feedback and observer forms the following example of 5th order system with $m=2$ inputs and $p=3$ outputs is considered

Given System Representation

```
| .05    .00    .00    .00    .00 | 1.00   1.00 |
| .00    .10    .00    .00    .00 |  .01   1.00 |
| .00    .00    .15    .00    .00 |  .02   1.00 |
| .00    .00    .00    .20    .20 |  .00   1.00 |
| .00    .00    .00   -.20    .20 |  .00   1.00 |     | A | B |
|---------------------------------+-------------  =  |---+---|
| 1.00   .01    .00    .00    .00 |                  | C |
|  .00   .00   1.00    .01    .00 |
| 1.00  1.00   1.00   1.00   1.00 |
```
(a)

The unique controllability and observability indices are

$$\mu = \{3\ 2\} \quad \text{and} \quad \nu = \{2\ 2\ 1\} \tag{b}$$

According to Eq.(3.70) the possible sets of pseudo-controllability and pseudo-observability indices are

$$\mu = \begin{bmatrix} \mu_1 \\ \mu_2 \end{bmatrix} = \begin{bmatrix} 4 & 3 & 2 & 1 \\ 1 & 2 & 3 & 4 \end{bmatrix}, \quad \nu = \begin{bmatrix} \nu_1 \\ \nu_2 \\ \nu_3 \end{bmatrix} = \begin{bmatrix} 3 & 2 & 2 & 1 & 1 & 1 \\ 1 & 2 & 1 & 3 & 2 & 1 \\ 1 & 1 & 2 & 1 & 2 & 3 \end{bmatrix} \tag{c}$$

It has been verified that the first set of controllability indices {4 1} and the first set of observability indices {3 1 1} are not admissible.

Comparing Eqs.(b) and (c), it may be concluded that the second set of PCI is equal to the unique controllability indices. Similarly, the second set of POI is equal to the unique observability indices. The unique feedback and observer forms of Eq.(a) are given in Eqs.(e). As was mentioned previously, the structures of matrices $\mathbf{A}_f$ and $\mathbf{A}_o$ are based on selector vectors $\tilde{\mathbf{v}}_o$ corresponding to the sets

$$\tilde{\mu} = \{2\ 3\} \quad \text{and} \quad \tilde{\nu} = \{1\ 2\ 2\} \tag{d}$$

obtained by ordering the sets μ and ν in (b), respectively.

Feedback (unique controllable) Form

[Unique controllability indices {3 2}; admissibility degree =.12E-07]

```
|    .0  .0       1.0   .0     .0  |  .0        .0 |
|    .0  .0        .0  1.0     .0  |  .0        .0 |
|  -.08  .0        .4   .0     .0  |  .0       1.0 |
|    .0  .0        .0   .0    1.0  |  .0        .0 |
| 112.3  .0    -291.9  .02     .3  | 1.0 -1562.5 |
|----------------------------------+--------------
|-255.4  .015 1563.7  -.25   1.0  |
|  -2.2  .0     32.3   .0     .02 |
|-260.0  .015 1614.4  -.25   1.0  |
```
(e1)

Observer (observable) Form

[Unique observability indices {2 2 1}; admissibility degree =.53E-06]

```
| -.13     .0      .0  -87.9  -76.8 | -475.0 -741.6 |
|  .0      .0      .0     .0     .0 |    -.1    -.1 |
|  .001    .0      .0    -.3    -.2 |   -1.1   -1.7 |
|  .0     1.0      .0     .2     .0 |    1.0    1.0 |
|  .0      .0     1.0    1.1     .7 |     .0    1.0 |
|-----------------------------------+---------------
|  .0      .0      .0    1.0     .0 |
|  .0      .0      .0     .0    1.0 |
| 1.0      .0      .0  470.6  268.6 |
```
(e2)

In Eqs.(e), in addition to the indices from Eq.(b), the *admissibility degree* of the indices are given. The *admissibility degree* of a full rank matrix is defined as the inverse of the condition number of that matrix, i.e. as the ratio of the smallest to the largest *singular value* of the matrix. The reason for using the inverse of the *condition number* is to avoid dealing with infinite numbers when the matrix is not full rank.

Controllability and observability forms obtained using all admissible sets of pseudo-controllability and observability indices are given in Eqs.(f) together with the sets of indices used, as well the admissibility degree of the corresponding similarity transformation matrices used in obtaining these forms.

Controllability (pseudo-controllable) Forms

[Pseudo-controllability indices {1,4}; admissibility degree = .66E-03]

```
|  .051   .0      .0      .0      .003 | 1.0      .0 |
|  .004   .0      .0      .0      .002 |  .0     1.0 |
|  .083  1.0      .0      .0      .024 |  .0      .0 |
| -.356   .0     1.0      .0     -.194 |  .0      .0 |
|  .732   .0      .0     1.0      .648 |  .0      .0 |
|--------------------------------------+--------------
| 1.001  1.01     .051    .002    .001 |
|  .002  1.01     .154    .023    .003 |
| 1.003  5.00     .700    .035    .027 |
```
(f1)

[Pseudo-controllability indices {2 3}; admissibility degree =.53E-03]

```
| .0     .0     .008   .0    -.071 | 1.0    .0  |
| .0     .0     .000   .0     .006 |  .0   1.0  |
|1.0     .0     .213   .0    1.366 |  .0    .0  |
| .0    1.0     .002   .0    -.114 |  .0    .0  |
| .0     .0     .006  1.0     .486 |  .0    .0  |
|----------------------------------+-------------
|1.00   1.01    .050   .051   .002 |
| .02   1.01    .003   .154   .021 |
|1.03   5.00    .054   .700   .035 |
```
(f2)

[Pseudo-controllability indices {3 2}; admissibility degree =.91E-05]

```
| .0     .0     .0  -12.7    .007 | 1.0    .0 |
| .0     .0     .0     -.08  .0   |  .0   1.0 |
|1.0     .0     .0  333.1  -.027  |  .0    .0 |
| .0    1.0     .0     .40   .0   |  .0    .0 |
| .0     .0    1.0-1562.5   .300  |  .0    .0 |
|---------------------------------+-----------
|1.0    1.0     .1      .05  .002 |
| .0    1.0     .0      .15  .0   |
|1.0    5.0     .1      .70  .003 |
```
(f3)

Observability (pseudo-observable) Forms

[Pseudo-observability indices {1 1 3}; admissibility degree = .16E-01]

```
| .049   .000   .001  -.004   .010 | 1.000  1.010 |
| .000   .150  -.001   .011  -.018 |  .020  1.010 |
| .0     .0     .0    1.0     .0   | 1.030  5.000 |
| .0     .0     .0     .0    1.0   |  .054   .700 |
|-.003   .002   .008  -.120   .501 |  .003   .035 |
|----------------------------------+--------------
|1.0     .0     .0     .0     .0   |
| .0    1.0     .0     .0     .0   |
| .0     .0    1.0     .0     .0   |
```
(f4)

[Pseudo-observability indices {1 2 2}; admissibility degree = .30E-03]

```
| .050   .085   .00   -.571   .002 | 1.000  1.010 |
| .0     .0     .0    1.0     .0   |  .020  1.010 |
| .0     .0     .0     .0    1.0   | 1.030  5.000 |
| .000   .017   .00    .038   .002 |  .003   .154 |
| .023  8.491  -.05  -56.723  .612 |  .054   .700 |
|----------------------------------+--------------
|1.0     .0     .0     .0     .0   |
| .0    1.0     .0     .0     .0   |
| .0     .0    1.0     .0     .0   |
```
(f5)

[Pseudo-observability iIndices {2 1 2}; admissibility degree = .17E-03]

```
| .0      .0      .0     1.0      .0    | 1.000  1.010 |
| .087    .149    .001 -1.752    .004   |  .020  1.010 |
| .0      .0      .0      .0     1.0    | 1.030  5.000 |
|-.005    .000    .000    .150    .000  |  .050   .051 |
|-4.906   .043  -.080  99.384    .401   |  .054   .700 |
|---------------------------------------+---------------
| 1.0     .0      .0      .0      .0    |
|  .0    1.0      .0      .0      .0    |
|  .0     .0     1.0      .0      .0    |
```

(f6)

[Pseudo-observability indices {1 3 1}; admissibility degree = .14E-03]

```
| .049    .070    .001    -.60     .879 | 1.000  1.010 |
| .0      .0      .0      1.0      .0   |  .020  1.010 |
|-.051 -6.894     .101 -15.78  413.821  | 1.030  5.000 |
| .0      .0      .0       .0     1.0   |  .003   .154 |
| .000    .012    .000    -.14     .550 |  .000   .023 |
|---------------------------------------+---------------
|1.0      .0      .0       .0      .0   |
| .0     1.0      .0       .0      .0   |
| .0      .0     1.0       .0      .0   |
```

(f7)

[Pseudo-observability indices {2 2 1}; admissibility degree = .12E-03]

```
|     .0       .0      .0      1.0       .0   | 1.000  1.010 |
|     .0       .0      .0       .0      1.0   |  .020  1.010 |
| -23.342  -40.000  -.137  470.581  268.581   | 1.030  5.000 |
|   -.005      .0      .0     .150       .0   |  .050   .051 |
|   -.056   -.080  -.001    1.137     .687    |  .003   .154 |
|---------------------------------------------+---------------
|    1.0       .0      .0       .0       .0   |
|     .0      1.0      .0       .0       .0   |
|     .0       .0     1.0       .0       .0   |
```

(f8)

Comparing the forms in Eqs.(e) and (f), it may be concluded that among all controllable forms, the forms corresponding to the sets of PCI

$$\mu = \{ 1 \ 4 \} \quad \text{and} \quad \mu = \{ 2 \ 3 \}$$

are more "convenient" than the controllability form corresponding to the set of PCI $\mu = \{3 \ 2\}$ which is the "unique" set of controllability indices of the pair $\{\mathbf{A},\mathbf{B}\}$, as well as the unique feedback form. The advantages of these forms are judged on the basis of absolute values of elements of matrices in these forms, which is a direct consequence of the admissibility degree of the transformation matrix used for the similarity transformation. Similarly, among all observable forms, it may be concluded that the observability form based on $\nu = \{1 \ 1 \ 3\}$ has the largest admissibility degree and, consequently, the smallest absolute values of its elements.

The main point of this example is to stress the necessity of checking the admissibility degree of the similarity transformation matrix corresponding to each admissible set of pseudo-controllability or pseudo-observability indices and to use the set leading to the largest admissibility degree. Then, the absolute values of the elements in a state space representation become relatively small, which is computationally desirable. As was the case in this example, the most convenient set of indices is not necessarily equal to the unique set of controllability or observability indices.

3.4 Matrix Fraction Description (MFD)

As introduced in Chapter 1, an alternative representation to either the state space description or the transfer matrix description is the *matrix fraction description (MFD)*. For a C-T MIMO system the MFD model is of the form

$$D(s)\,y(s) = N(s)\,u(s) \tag{3.88}$$

where $y(s)$ is the $(p\times 1)$ system output and $u(s)$ is the $(m\times 1)$ system input. The matrices $D(s) = \{\, d_{ij}(s) \,\}$ and $N(s) = \{\, n_{ij}(s) \,\}$ are *left coprime* $(p\times p)$ and $(p\times m)$ polynomial matrices. The orders of polynomials $d_{ij}(s)$ and $n_{ij}(s)$ satisfy:

$$\begin{aligned} 0 &\le \deg[d_{ij}(s)] \le k \\ 0 &\le \deg[n_{ij}(s)] \le k \end{aligned} \tag{3.89}$$

where $k \le n$, n being the order of the system.

In accordance with the discussion in Chapter 1, polynomials $d_{ij}(s)$ and $n_{ij}(s)$ will be represented by:

$$d_{ij}(s) = \sum_{h=0}^{k} d_{ijh}\, s^{h} \quad \text{and} \quad n_{ij}(s) = \sum_{h=0}^{k} n_{ijh}\, s^{h} \tag{3.90}$$

Similarly, polynomial matrices $D(s)$ and $N(s)$ may be written as

$$D(s) = \sum_{h=0}^{k} \mathbf{D}_h\, s^{h} \quad \text{and} \quad N(s) = \sum_{h=0}^{k} \mathbf{N}_h\, s^{h} \tag{3.91}$$

where

$$\mathbf{D}_h = \begin{bmatrix} d_{11h} & \cdots & d_{1ph} \\ \vdots & \cdots & \vdots \\ d_{p1h} & \cdots & d_{pph} \end{bmatrix} \quad \text{and} \quad \mathbf{N}_h = \begin{bmatrix} n_{11h} & \cdots & n_{1mh} \\ \vdots & \cdots & \vdots \\ n_{p1h} & \cdots & n_{pmh} \end{bmatrix}$$

Two polynomial matrices are *left coprime* if they do not have common terms, or if

$$\mathrm{rank}[\,D(s) \;|\; N(s)\,] = p \quad \text{for all} \quad s$$

In other words, it is assumed that all existing common terms in $D(s)$ and $N(s)$ have been cancelled. In some relevant literature the MFD model is referred to as an *auto-regressive-moving-average (ARMA)* model. As is the case with state space models, the MFD representation is not unique, i.e. there is more than one pair of polynomial matrices $\{D(s), N(s)\}$ that will represent a given system.

One variation of an MFD model is the following model:

$$y(s) = \tilde{N}(s)\,\tilde{D}^{-1}(s)\,u(s) \tag{3.92}$$

which is sometimes expressed as

$$\begin{aligned} y(s) &= \tilde{N}(s)\,v(s) \\ \tilde{D}(s)\,v(s) &= u(s) \end{aligned} \tag{3.93}$$

where $v(s)$ is an auxiliary m dimensional vector.

It is quickly concluded that the MFD model is related to the system transfer matrix, $G(s)$ by

$$G(s) = D^{-1}(s)\,N(s) = \tilde{N}(s)\,\tilde{D}^{-1}(s) \tag{3.94}$$

Similarly, the $(p \times m)$ and $(m \times m)$ matrices $\tilde{N}(s)$ and $\tilde{D}(s)$ are *right coprime* if:

$$rank\left[\,\tilde{N}^T(s) \;|\; \tilde{D}^T(s)\,\right] = m \quad \text{for all} \quad s \tag{3.95}$$

It is worth mentioning that in the case of SISO models, i.e. for $p = m = 1$, matrices $D(s)$ and $N(s)$ become scalar polynomials $d(s)$ and $n(s)$, respectively, and the coprime condition reduces to:

$$\mathrm{rank}[\,d(s) \;|\; n(s)\,] = 1 \quad \text{for all} \quad s \tag{3.96}$$

The condition of Eq.(3.96), in fact, implies that polynomials $d(s)$ and $n(s)$ have no common factors, i.e. there is no value $s = s_0$ for which both $d(s_0)$ and $n(s_0)$ are equal to zero. In other words, for $s = p_i$, $i=[1,n]$, i.e. system poles, $d(p_i) = 0$, but $n(p_i) \neq 0$; i.e. the transfer function $g(s) = n(s)/d(s)$ does not have any pole-zero cancellations. Similarly, if there are no common factors, then for $s = z_i$, i.e. system zeros for which $n(z_i) = 0$, $d(z_i) \neq 0$.

Recall that in the case of SISO systems, it is typically assumed that $d(s)$ is a *monic polynomial*, i.e.

$$d(s) = \sum_{i=1}^{n} d_i s^i \quad \text{where} \quad d_n = 1 \tag{3.97}$$

This type of "normalization," when applied to polynomial matrices requires some additional consideration.

Definition 3.4 The *degree of a polynomial vector* (row or column), $\boldsymbol{a}(s)$:

$$\boldsymbol{a}(s) = \left[\; a_1(s) \;\cdots\; a_p(s) \;\right]$$

is equal to the highest degree of all (polynomial) entries, $a_i(s)$, in the vector. The polynomial vector $\boldsymbol{a}(s)$ is *monic* if its polynomial with highest degree is monic in the sense of Eq.(3.97) and there is only one polynomial with that degree.

Definition 3.5 A *column-reduced polynomial matrix* is defined as follows: For a $p \times p$ polynomial matrix $D(s) = \{\, d_{ij}(s) \,\}$, let the degree of the i^{th} column be n_i. In general,

$$\deg \det\{\, D(s) \,\} = n \le \sum_{i=1}^{p} n_i \tag{3.98}$$

If equality holds in Eq.(3.98), $D(s)$ is considered to be *column reduced.*

Definition 3.6 A $p \times p$ column-reduced polynomial matrix $D(s)$ is said to be *monic* if in each column the polynomial with the highest degree is monic.

Row-reduced and row-reduced monic polynomial matrices are defined in a similar manner. Unless stated differently, the $p \times p$ matrix $D(s)$ of the left coprime pair $\{D(s),\ N(s)\}$ is taken to be both column-reduced and monic. In the corresponding case of the right coprime pair $\{\tilde{N}(s), \tilde{D}(s)\}$ the $m \times m$ matrix $\tilde{D}(s)$ is considered to be both row-reduced and monic.

Since the concept row-(or column-) reduced polynomial matrices is important for our developments, some simple examples will be presented: Consider the 3×3 column-reduced polynomial matrix $D(s)$ with column degrees $\{\, n_i \,\}$ given by:

$$\{\, n_1 , n_2 , n_3 \,\} = \{\, 2 , 2 , 2 \,\} \tag{3.99}$$

$D(s)$ has the general form

$$D(s) = \begin{bmatrix} x + xs + s^2 & x + xs & x + xs \\ x + xs & x + xs + s^2 & x + xs \\ x + xs & x + xs & x + xs + s^2 \end{bmatrix} \tag{3.100}$$

where x represents a possible non-zero value. The matrices in Eqs.(3.91) are:

$$\left[\mathbf{D}_0 \mid \mathbf{D}_1 \mid \mathbf{D}_2\right] = \left[\begin{array}{ccc|ccc|ccc} x & x & x & x & x & x & 1 & 0 & 0 \\ x & x & x & x & x & x & 0 & 1 & 0 \\ x & x & x & x & x & x & 0 & 0 & 1 \end{array}\right] \tag{3.101}$$

In the case for which the column degrees $\{n_i\}$ are not equal, e.g. if

$$\{ n_1, n_2, n_3 \} = \{ 2, 1, 3 \} \tag{3.102}$$

then the corresponding description of Eq.(3.100) becomes:

$$D(s) = \begin{bmatrix} x+xs & x+s & x+xs+xs^2 \\ x+xs+s^2 & x & x+xs+xs^2 \\ x+xs & x & x+xs+xs^2+s^3 \end{bmatrix} \tag{3.103}$$

and the 3×3 matrices in Eqs.(3.91) become:

$$\left[\mathbf{D}_0 \mid \mathbf{D}_1 \mid \mathbf{D}_2 \mid \mathbf{D}_3\right] = \left[\begin{array}{ccc|ccc|ccc|ccc} x & x & x & x & 1 & x & 0 & 0 & x & 0 & 0 & 0 \\ x & x & x & x & 0 & x & 1 & 0 & x & 0 & 0 & 0 \\ x & x & x & x & 0 & x & 0 & 0 & x & 0 & 0 & 1 \end{array}\right] \tag{3.104}$$

Note that in both cases $d(s) = \det \{ D(s) \}$ is an $n=6$th order polynomial with the coefficient associated with s^6 equal to ± 1. Also the $[p \times (k+1)p]$ matrix $\mathbf{D}_r$ defined by:

$$\mathbf{D}_r = \left[\mathbf{D}_0 \mid \mathbf{D}_1 \mid \cdots \mid \mathbf{D}_k\right] \tag{3.105}$$

contains:

- $n=6$ columns with non-zero/non-unity elements,
- $p=3$ columns of the $p \times p$ identity matrix, and
- $kp-n$ columns of zeros.

It may be verified that the locations of the non-zero/non-unity columns and the columns of the identity matrix mentioned above are defined by the unity elements of the selector vectors $\mathbf{v}_g$ and $\mathbf{v}_{id}$, respectively, generated by a set of POI $\{ \nu_i \}$ equal to the column degrees $\{ n_i \}$ of $D(s)$. It is worth mentioning that matrices $D(s)$ given above are completely general, since it is always possible to premultiply both $D(s)$ and $N(s)$ by a $p \times p$ "permutation" matrix to bring $D(s)$ to the above form. The transfer matrix $G(s) = D^{-1}(s)N(s)$ is not altered by this multiplication.

Alternatively, with argument z replacing s, Eqs.(3.88) to (3.92) represent D-T MFD or ARMA models. In particular, using the z-domain description, Eq.(3.88) may be rewritten as

$$(\mathbf{D}_k z^k + \cdots + \mathbf{D}_1 z + \mathbf{D}_0)\, y(z) = (\mathbf{N}_k z^k + \cdots + \mathbf{N}_1 z + \mathbf{N}_0)\, \mathbf{u}(z) \qquad (3.106)$$

where the matrices $\{\mathbf{D}_i\}$, $0 \le i \le k$, and $\{\mathbf{N}_i\}$, $0 \le i \le k$ in Eq.(3.106) have the dimensions $(p \times p)$ and $(p \times m)$, respectively. The time domain equivalent of Eq.(3.106) is

$$\mathbf{D}_k \mathbf{y}(t+k) + \cdots + \mathbf{D}_1 \mathbf{y}(t+1) + \mathbf{D}_0 \mathbf{y}(t) = \mathbf{N}_k \mathbf{u}(t+k) + \cdots + \mathbf{N}_1 \mathbf{u}(t+1) + \mathbf{N}_0 \mathbf{u}(t) \qquad (3.107)$$

where t has been used as the discrete time index, taking on only integer values. More specifically, for the example above, the difference equation corresponding to Eq.(3.101) is

$$\mathbf{y}(t+2) + \mathbf{D}_1\, \mathbf{y}(t+1) + \mathbf{D}_0\, \mathbf{y}(t) = \mathbf{N}_2\, \mathbf{u}(t+2) + \mathbf{N}_1\, \mathbf{u}(t+1) + \mathbf{N}_0\, \mathbf{u}(t) \qquad (3.108)$$

since $\mathbf{D}_2$ is an identity matrix. The vector $y(t)$ has, of course, three components.

The special case for which all column degrees of $D(s)$ are equal is sometimes called the *equi-observable* case, and $D(s)$ is then considered to be a *regular polynomial matrix*. The time domain equivalent in the general case when the column degrees are not necessarily equal is not as simple as for the regular case. For instance, corresponding to the example of Eq.(3.102), we may write

$$\begin{bmatrix} y_2(t+1) \\ y_1(t+2) \\ y_3(t+3) \end{bmatrix} = -\, \mathbf{d}_{23}\, y_3(t+2) - \begin{bmatrix} \mathbf{d}_{11} & \mathbf{d}_{13} \end{bmatrix} \begin{bmatrix} y_1(t+1) \\ y_3(t+1) \end{bmatrix} - \mathbf{D}_0\, \mathbf{y}(t) \qquad (3.109)$$
$$+ \mathbf{N}_3\, \mathbf{u}(t+3) + \mathbf{N}_2\, \mathbf{u}(t+2) + \mathbf{N}_1\, \mathbf{u}(t+1) + \mathbf{N}_0\, \mathbf{u}(t)$$

where $\mathbf{d}_{ij}$ is the j^{th} column of the matrix $\mathbf{D}_i$. In general, the left side of such a description is a p dimensional vector containing samples $y_j(t+n_j)$, $j=[1,p]$, but arranged in ascending order of column degrees n_j. In fact, the left-hand side of (3.109) is

$$[\; y_2(t+n_2) \quad y_1(t+n_1) \quad y_3(t+n_3) \;]^T$$

since in the example, $n_2 \le n_1 \le n_3$.

When a left coprime pair of polynomial matrices $\{ D(s), N(s) \}$, (or with z instead of s for a D-T system), as described previoulsy is available, it is relatively easy to transform a left coprime MFD into an observable representation R_o based on a set of admissible POI corresponding to the column degrees of $D(s)$. And, conversely, given an observable representation R_o, based on a set of POI, it is possible to obtain a corresponding left coprime MFD. Similar statements can be made for the relationships between a right coprime row-reduced MFD and a controllable state space representation. This is the subject of the next chapter where the conversion between the various system models is considered.

3.5 Summary

In this chapter the important concepts of system structure and canonical forms were presented. In Section 3.1 the systems under consideration were restricted to be SISO systems. Controllable, observable and Jordan forms were discussed, particularly with respect to their relation with the corresponding transfer functions. In Section 3.2 the SISO controllable and observable forms were extended to MIMO systems through the use of specific similarity transformations of the system state. In Section 3.3 the discussion of MIMO canonical forms was continued. A very flexible method is given for describing the structure of a MIMO system, using pseudo-controllability and pseudo-observability indices (PCI and POI). The pseudo-controllable and pseudo-observable forms provide a selection of possible system structures from which the "best" one can be chosen. This "best" structure is not always the "unique" controllable or observable canonical form commonly used in the control system community. With this chapter's thorough discussion and exercises on MIMO system structure, the reader will be prepared to study Chapter 4, which presents an entire collection of algorithms for conversion between the various system types. Finally, in Section 3.4 the matrix fraction description (MFD) was presented in detail, tying the concepts of POFs and PCFs to left and right coprime MFD forms.

3.6 References

The topic of system modeling covers a broad area, but we again refer to the basic advanced texts of Kailath (1980), Chen (1984) and Brogan (1991) for well written background reading. Luenberger (1967) is a classical paper on canonical forms for MIMO systems. Ackermann (1985) discusses the various "Frobenius" canonical forms used in this chapter in his Appendix A. For details on the "modified" forms used here, i.e. the use of *pseudo-controllable or observability indices*, see Bingulac and Krtolica (1987), or Gevers and Wertz (1982); also called *nice indices* in Antoulas (1985). A related discussion on the invariance of controllability indices is given in Bingulac and VanLandingham (1992). Kailath (1980) relates state space and matrix fraction descriptions in his Chapter 6.

Ackermann, J. (1985), *Sampled-Data Control Systems*, Springer-Verlag, Berlin.

Antoulas, A.C. (1986), "On recursiveness and related topics in linear systems," *IEEE Trans. on Automatic Control*, **AC-31**, 12, 1121-1135.

Bingulac, S. and R. Krtolica (1987), "On admissibility of pseudo-observability indices," *IEEE Trans. on Automatic Control*, **AC-32**, 920-922.

Bingulac, S. and H.F. VanLandingham (1992), "On the invariance of controllability indices of linear MIMO systems under orderings of the inputs," *Proceedings of the 30th Allerton Conference*, University of Illinois, September 1992, pp 83-87.

Brogan, W.L. (1984), *Modern Control Theory*, Prentice-Hall, Pub. Co., Englewood Cliffs, NJ.

Chen, C-T. (1984), *Linear System Theory and Design*, Holt, Rinehart and Winston, Inc., New York, NY.

Gevers, M. and V. Wertz (1982), "Uniquely identifiable state space and ARMA parametrizations for multivariable linear systems," *Automatica*, **20**, 333-347.

Kailath, T. (1980), *Linear Systems*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Luenberger D.G. (1967), "Canonical forms for linear multivariable systems," *IEEE Trans. on Automatic Control*, **AC-12**, 290-293.

3.7 Exercises

3.1 Given below is a state space representation of a strictly proper SISO system

$$R = \begin{bmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{c} & 0 \end{bmatrix} = \left[\begin{array}{cccc|c} -1 & 1 & 1 & 0 & 1 \\ -1 & -1 & 2 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & -2 & 2 \\ \hdashline 0 & 1 & 0 & 2 & 0 \end{array}\right]$$

Determine:

(a) —the controllability (controllable) form,
(b) —the observability (observable) form,
(c) —the feedback (controllable) forms using Procedures 1 , 2 and 3 described in Section 3.2.3, and
(4) —the observer (observable) form using the principle of duality.

Hints:

- Define the representation R = {**A**,**b**,**c**} using the operator DMA, or INPM.
- Calculate the controllability and observability matrices using the operators Qc and Qo.
- Calculate the required canonical forms using the operator STR.
- Find the null space by using the operator NRS.
- Determine the transfer function, in Procedure 3, using the operator SSTF.
- Matrix partitioning could be done using the operator CTC.

A version of *L-A-S* program performing this exercise is available in the subdirectory C:\LAS\DPF\EXER31.DPF.

3.2 The coefficients w_i and a_i, $i=[0,n]$, $n=4$, of a non-strictly proper SISO transfer fuction $g(s) = w(s)/a(s)$ are given below:

$$\{ w_i \} = [\, 5 \quad 6 \quad -3 \quad -4 \quad -2 \,]$$
$$\{ a_i \} = [\, 2 \quad 6 \quad 7 \quad 4 \quad 1 \,]$$

Determine:

(a) —the feedback (controllable) form $R_c = \{\mathbf{A}_c, \mathbf{b}_c, \mathbf{c}_c, d_c\}$ using Algorithm *Rc1*, Section 3.4.2.
(b) —the observer (observable) form $R_o = \{\mathbf{A}_o, \mathbf{b}_o, \mathbf{c}_o, d_o\}$ using Algorithm *Ro1*, Section 3.4.2.
(c) —the controllability (controllable) form $R_c = \{\mathbf{A}_c, \mathbf{b}_c, \mathbf{c}_c, d_c\}$ using Algorithm *Rc2*, Section 3.4.2.
(d) —the observability (observable) form $R_o = \{\mathbf{A}_o, \mathbf{b}_o, \mathbf{c}_o, d_o\}$ using Algorithm *Ro2* Section 3.4.2.
(e) —the transfer functions of all the obtained state space representations.

Hints:

- Define the coefficients w_i and a_i, $i=[0,n]$, $n=4$, using either operator DMA, or INPM.
- Row partitioning can be performed using operator CTC.
- Extraction of the dimensions of a row/column/matrix can be done using the operators RDI and CDI of the subroutine GETD.SUB.
- Row/column/matrix transposition can be done using the operator T.
- The controllability and observability matrices can be calculated using the operators Qc and Qo, respectively.
- Rows $\mathbf{s}_1$ and $\mathbf{s}_n$ could be defined by partitioning an identity matrix $\mathbf{I}_n$, where the identity matrix is generated by the operator DIM.
- "Shifting" rows/columns/matrices up/down/left/right can be done using the operators SHU, SHD, SHL and SHR.
- The transfer function can be calculated using the operator SSTF.

A version of an *L-A-S* program which solves this exercise is available in the *L-A-S* subdirectory C:\LAS\DPF\EXER32.DPF.

3.3 A 6th order C-T system with $m=3$ inputs and $p=2$ outputs is given below:

$$\left[\begin{array}{c|c} \mathbf{A} & \mathbf{B} \\ \hline \mathbf{C} & \mathbf{D} \end{array}\right] = \left[\begin{array}{cccccc|ccc} 1 & 1 & 0 & 0 & 0 & 0 & .01 & 1 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & -4 & 1 & 0 & 0 \\ 2 & 2 & 0 & 2 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -2 & 1 & 0 & 1 & .01 \\ \hline 1 & .04 & .01 & .02 & .02 & 0 & 0 & 0 & 0 \\ 0 & 3 & 1 & 1 & 3 & 0 & 0 & 1 & 0 \end{array}\right]$$

Determine:

(a) —the unique sets of controllability and observability indices.
(b) Using Eq.(3.70) and Definition 3.2, determine all sets of PCI and POI.
(c) Determine which sets are admissible.
(d) For all admissible sets of PCI and POI determine the corresponding PCF and POF.
(e) Calculate the degrees of admissibility for all admissibile sets.
(f) Determine the particular PCI and POI which correspond to the "best selection," i.e. having the largest admissibility degree.
(g) Are these "best" sets equal to the unique sets of controllability and observability indices?

Hints:

- To define the required arrays, use operator DMA.
- To calculate Q_c and Q_o, use operators Qc and Qo.
- To calculate the unique controllability/observability indices, use either operator RKC/RKR and then subroutine CIND.SUB, or subroutine IND.SUB.
  The calling sequence for IND is:
  `Q,mp,cut,eps(IND,SUB)=Ind.`
  The calling sequence for CIND is: `v(CIND,SUB)=Ind.`
- To calculate selector matrices, use either the operators POI and DSM or the subroutine SMAT.SUB.
  The calling sequence for SMAT is: `Pind(SMAT,SUB)= Inmx,`
  `Sa,Si,Sli,Sld.`

- To calculate the admissibility degree of a particular realization, use either the operator SVD, or the subroutine C#.SUB.
  The calling sequence for C# is: `T(C#,SUB)=C#.`
- For additional hints see those following Exercises 3.1 and 3.2.

A version of an *L-A-S* program which solves this exercise is available in the *L-A-S* subdirectory C:\LAS\DPF\EXER33.DPF.

3.4 A (5×5) matrix **A** and three input matrices $\mathbf{B}_i$, $i=1,2,3$, are given below:

$$\mathbf{A} = \mathrm{diag}\{\ 1,\ 2,\ 3,\ 4,\ 5\ \}$$

$$\mathbf{B}_1 = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 2 & 1 & 1 \\ 0 & 0 & 1 \\ .01 & 0 & 1 \end{bmatrix}, \quad \mathbf{B}_2 = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 2 & 1 & 1 \\ 0 & .01 & 1 \\ 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{B}_3 = \begin{bmatrix} 0 & 1 & 1 \\ 0 & 0 & 1 \\ 2 & 1 & 1 \\ 0 & .01 & 1 \\ .01 & 0 & 1 \end{bmatrix}$$

For the pairs $\{\mathbf{A},\ \mathbf{B}_i\}$, $i=1,2,3$, determine:

(a) —the three sets of unique controllability indices.

(b) —the three pairs, $\{\mathbf{A}_{ci},\ \mathbf{B}_{ci}\}$, of Feedback controllable forms.

(c) —the three pairs, $\{\mathbf{A}_{ci},\ \mathbf{B}_{ci}\}$, of Controllability forms which correspond to the set $\mu = \{\ 1\ 1\ 3\ \}$ of admissible PCI.

Hints:

- To calculate a feedback controllable form, use procedure 1 or 2, Section 3.3.5.
- To calculate a controllability form, use the dual of the procedure given in Section 3.3.4, or use Algorithm *SSRc* discussed in Chapter 4, Section 4.1.2.
- Since the same calculations, but using different input matrices ($\mathbf{B}_i$), should be performed, it is advisable to use an "incompletely" specified operator statement. (See Appendix C, the subsection: "Omitting Input, Output and Operator Fields").
- For other hints see those following Exercises 3.1, 3.2 and 3.3.

A version of an *L-A-S* program which solves this exercise is available in the *L-A-S* subdirectory C:\LAS\DPF\EXER34.DPF.

Remark: Note that even though the matrices $\mathbf{B}_i$ are rather similar, the feedback controllable forms are quite different, while the controllability forms based on the selected set μ are nearly equal.

3.5 A 7th order non-diagonalizable matrix **A** is given below:

$$\mathbf{A} = \begin{bmatrix} 2 & 0 & -1 & 0 & 0 & -2 & -2 \\ 0 & 3 & 2 & 0 & 0 & 2 & 2 \\ 0 & -1 & 1 & 0 & 0 & -1 & -2 \\ 0 & -1 & -1 & 2 & 1 & 0 & -1 \\ 0 & 0 & 0 & -1 & 2 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 3 & 2 \\ 0 & 0 & -1 & 0 & 0 & -2 & 1 \end{bmatrix}$$

Determine:

(a) —the eigenvalues of **A**. Verify that the eigenvalues are $2 \pm j1$ and 2 with multiplicities 2 and 3 respectively,
(b) —the modal matrix **P** which transforms **A** into a block diagonal "real number" Jordan form $\mathbf{A}_J$ satisfying $\mathbf{A}_J = \mathbf{P}^{-1}\,\mathbf{A}\,\mathbf{P}$, and
(c) —the Jordan form $\mathbf{A}_J$.

Hints:

- See Appendix B for more details on Jordan forms.
- Define the matrix **A** using either the operator DMA or INPM.
- To calculate the modal matrix **P**, use the subroutine MODM.SBR. Note that MODM.SBR calls either CHAR.SBR, or CHAC.SBR, for each distinct eigenvalue of **A**.
- Calculate the distinct eigenvalues of **A** using the operators EGV, DMA and DSM.
- To calculate $\mathbf{A}_J$, use the operators -1 and * (inversion and multiplication).

A version of an *L-A-S* program which solves this exercise is available in the *L-A-S* subdirectory C:\LAS\DPF\EXER35.DPF.

Chapter 4 Intermodel Conversion

In the previous chapters several methods of system representation have been discussed. The purpose of this chapter is to present the various algorithms that can be used to convert between the different system models. As illustrated in Fig. 4.1, we consider a *pentagon* of five basic representations. The arrows refer to available algorithms that may be invoked to perform the indicated conversion.

Our approach to presenting this material will be to focus on a particular block of the "pentagon" in Fig. 4.1, e.g. the state space representation, and discuss the algorithms used for converting this model to each of the other four.

4.1 Conversions from a State Space Model

Since we are most familiar with the state space representation, having used this model as our fundamental system description in previous chapters, we will begin by considering the different methods of converting this model to other forms. Even though most of the techniques apply equally well to C-T models, our concentration will be on the conversions of D-T models, as indicated in Fig. 4.1. As is well known, state space representation for a specific system is not unique.

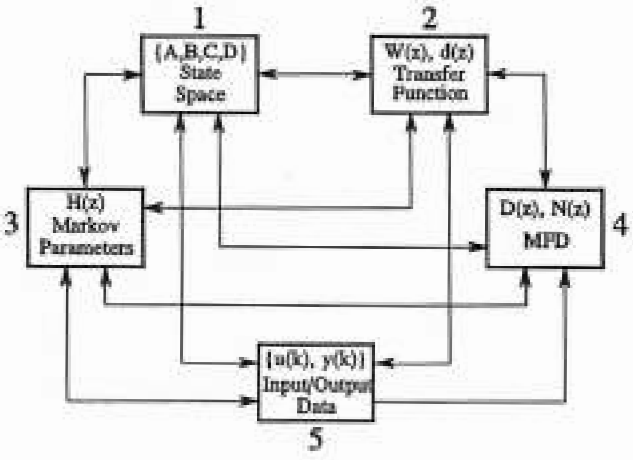


FIGURE 4.1 Algorithms for Intermodel Conversion

In this section several options are discussed. Initially, we assume a general state space representation. And anticipating the need to transform the general state space form into one of the canonical forms, two algorithms are provided: one to convert to an observable form, and a second, to controllable form. We will consider exclusively the controllability and observability forms since they take advantage of flexibility offered by pseudo-controllability and pseudo- observability indices (PCI) and (POI). As was mentioned in Chapter 3, these forms will be sometimes referred to as pseudo-controllable (PCF) and pseudo-observable forms (POF). Another reason is that we found that controllability and observability forms of the structure in Eqs.(3.66) and (3.80) are best suited for all intermodel conversions to be discussed. These forms are better than the feedback and observer forms as well as forms of the Luenberger structure. This is in fact the reason why we in Chapter 3 insisted on these forms and stated that they are more "natural" than other possible canonical forms. The remaining algorithms in this section provide for conversion to transfer function form, ARMA (or MFD) forms, as well as for calculating the Markov parameters, and system responses.

4.1.1 General State Space to Observable Form

This algorithm transforms a general form $R = \{\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}\}$ to an observability (POF) form $R_o = \{\mathbf{A}_o, \mathbf{B}_o, \mathbf{C}_o, \mathbf{D}_o\}$, bases on an admissible set ν of POI as discussed in Chapter 3. To recall the basic steps of the procedure,

$$\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \nu, \epsilon(SSRo) \rightarrow \mathbf{A}_o, \mathbf{B}_o, \mathbf{C}_o, \mathbf{D}_o, \mathrm{C\#}$$

```
1. Set $\nu$ $(SMat) \Rightarrow \nu_m, \mathbf{S}_a, \mathbf{S}_i, \mathbf{S}_b, \mathbf{S}_M$
2. Set $\mathbf{A}, \mathbf{C}$ $(Qo) \Rightarrow \mathbf{Q}_o$  ($\mathbf{Q}_o$ has $\nu_m+1$ blocks of $\mathbf{CA}^i$ of $p$ rows.)
3. Set $\mathbf{S}_b^T\, \mathbf{Q}_o \rightarrow \mathbf{T}_o$
4. Set $\mathbf{T}_o$ $(C\#) \Rightarrow$ C#
5. Set $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{T}_o^{-1}$ $(STR) \rightarrow \mathbf{A}_o, \mathbf{B}_o, \mathbf{C}_o$
6. Set $\mathbf{D} \rightarrow \mathbf{D}_o$
```

The quantity ϵ is a sufficiently small positive number used as "machine zero." The algorithm *C#*, Step 4, determines the "degree of admissibility of the set ν by calculating C#, the ratio of the smallest to the largest singular value of $\mathbf{T}_o$. (C# is the inverse of the "condition number" of $\mathbf{T}_o$.) For more details on MIMO observability forms see Section 3.3.4.

4.1.2 General State Space to Controllable Form

This algorithm transforms a general form $R = \{\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}\}$ to a controllability (PCF) form $R_c = \{\mathbf{A}_c, \mathbf{B}_c, \mathbf{C}_c, \mathbf{D}_c\}$ based on an admissible set μ of PCI.

$$\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}, \boldsymbol{\mu}, \epsilon(SSRc) \rightarrow \mathbf{A}_c, \mathbf{B}_c, \mathbf{C}_c, \mathbf{D}_c, \mathbf{C\#}$$

1. Set μ $(SMat) \Rightarrow \mu_m, \mathbf{S}_a, \mathbf{S}_t, \mathbf{S}_B, \mathbf{S}_M$
2. Set $\mathbf{A}, \mathbf{B}$ $(Qc) \Rightarrow \mathbf{Q}_c$ ($\mathbf{Q}_c$ has μ_m+1 blocks $\mathbf{A}^i\mathbf{B}$ of m columns.)
3. Set $\mathbf{Q}_c$ $\mathbf{S}_B \Rightarrow \mathbf{T}_c$
4. Set $\mathbf{T}_c$ $(C\#) \Rightarrow C\#$
5. Set $\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{T}_c$ $(STR) \Rightarrow \mathbf{A}_c, \mathbf{B}_c, \mathbf{C}_c$
6. Set $\mathbf{D} \Rightarrow \mathbf{D}_c$

Note that *SSRc* is the dual algorithm to *SSRo*. However, due to different sets μ and ν used, representations R_c and R_o obtained by these algorithms are *not* dual to each other in the sense of the definition given in Section 3.2.2. For more details on MIMO controllability forms see Section 3.3.

4.1.3 State Space to Transfer Function

This algorithm transforms a general form $R = \{\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}\}$ to a transfer function matrix $G(s) = \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D}$ using Leverrier's algorithm. The two versions of this conversion are explained in Section 1.3.9. Recall the symbolic notations

$$\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}(LALG) \rightarrow \mathbf{d}, \boldsymbol{W}_r, \boldsymbol{W}$$

and

$$\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}(SSTF) \rightarrow \mathbf{d}, \boldsymbol{W}$$

where $\mathbf{W}$ is a $[pm \times (n+1)]$ matrix in a PMF, see Eq.(G6) in the Glossary.

4.1.4 State Space to Markov Parameters

This algorithm transforms a general form $R = \{\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}\}$ to a set of Markov parameters. The *Markov parameters* for a D-T system may be described as the matrix response sequence of the system, initially at rest, to a collection of *unit pulses* of excitation. Thus, given a state space model

$$\begin{aligned} \mathbf{x}(k+1) &= \mathbf{A}\,\mathbf{x}(k) + \mathbf{B}\,\mathbf{u}(k)\,, \quad \mathbf{x}(0) \\ \mathbf{y}(k) &= \mathbf{C}\,\mathbf{x}(k) + \mathbf{D}\,\mathbf{u}(k) \end{aligned} \tag{4.1}$$

with $\mathbf{x}(0) = 0$, and $\mathbf{u}(k) \rightarrow \delta(k)\mathbf{I}_m$, ($\delta(k) = 1$ if $k = 0$, otherwise $\delta(k) = 0$):

$$\mathbf{H}(k) = \sum_{i=0}^{k-1} \mathbf{C}\mathbf{A}^{k-i-1}\mathbf{B}\,\delta(i) + \mathbf{D}\,\delta(k) \tag{4.2}$$

The first few terms of this *characteristic* sequence are easily calculated to be

$$\mathbf{H}(k) = \{\, \mathbf{D},\ \mathbf{CB},\ \mathbf{CAB},\ \mathbf{CA}^2\mathbf{B},\ \mathbf{CA}^3\mathbf{B},\ \cdots \} \tag{4.3}$$

These matrix elements are the *Markov parameters* of the system in Eq.(4.1). For convenience we will express the Markov parameters $\mathbf{H}_k$ by a polynomial matrix in z^{-1}, i.e.

$$H(z^{-1}) = \sum_{i=0}^{\infty} \mathbf{H}_i z^{-i} \tag{4.4}$$

where $\mathbf{H}_0 = \mathbf{D}$ and $\mathbf{H}_i = \mathbf{C}\,\mathbf{A}^{i-1}\,\mathbf{B}$, for $i=[1,\infty]$.

The reason for using Eq.(4.4) is that it may be shown that the transfer function matrix $G(z)$ may be expressed by:

$$G(z) = H(z^{-1}) \tag{4.5}$$

It may be verified that if all eigenvalues of $\mathbf{A}$ are within unit circle, i.e. if

$$|\lambda(\mathbf{A})| < 1 \tag{4.6}$$

then $\|\mathbf{H}_{M-1}\| << 1$ for a sufficiently large finite $M < \infty$.

If Eq.(4.6) holds, the polynomial matrix $H(z^{-1})$ could formally be represented by:

$$H(z^{-1}) \triangleq \{\, h_{ij}(z^{-1})\,\} = I_p(z^{-1})\,\mathbf{H}_r = \mathbf{H}_c\, I_m^T(z^{-1}) \tag{4.7}$$

where

$$I_p(z^{-1}) = \begin{bmatrix} \mathbf{I}_p & \mathbf{I}_p z^{-1} & \cdots & \mathbf{I}_p z^{-M+1} \end{bmatrix}$$

$$I_m(z^{-1}) = \begin{bmatrix} \mathbf{I}_m & \mathbf{I}_m z^{-1} & \cdots & \mathbf{I}_m z^{-M+1} \end{bmatrix}$$

$$\mathbf{H}_r = \begin{bmatrix} \mathbf{H}_0 & \mathbf{H}_1 & \cdots & \mathbf{H}_{M-1} \end{bmatrix} \quad \text{and} \quad \mathbf{H}_c = \begin{bmatrix} \mathbf{H}_0 \\ \mathbf{H}_1 \\ \vdots \\ \mathbf{H}_{M-1} \end{bmatrix}$$

For more details on this notation, see the Glossary of Symbols.

If Eq.(4.6) does not hold, then instead of using the original representations R, a "time scaling" could be performed, which amounts to dividing $\mathbf{A}$ and $\mathbf{B}$ by a scalar f satisfying:

$$f > \max |\lambda(\mathbf{A})|$$

In other words, a "time scaled" system representation R_f should be used where:

$$R_f = \{ \mathbf{A}/f, \mathbf{B}/f, \mathbf{C}, \mathbf{D} \} \tag{4.8}$$

After performing a desired model conversion, the obtained model should, of course, be "time scaled" back by the same used scalar f.

Using Eq.(4.4) the following algorithm is suggested:

```
Syntax:            A, B, C, D, M (SSH) ⇒ H, h_M

    1. Set A,B (Qc) ⇒ Q_c  (Q_c has M-1 blocks A^i B of m columns.)
    2. Set [ D | C Q_c ] ⇒ H_r
    3. Set ‖H_{M-1}‖ ⇒ h_M
    4. Set H_r, m (PMFr) ⇒ H
```

The matrix **H** is a $[pm \times M]$ matrix in a PMF whose rows contain the first M coefficients of the $(M$-1)st order polynomials $h_{ij}(z^{-1})$ in z^{-1} defined by:

$$H(z^{-1}) = \{ h_{ij}(z^{-1}) \}$$

For more details on the PMF see the Glossary of Symbols. Algorithm *PMFr* is a polynomial "service" algorithm which transforms a $[p \times Mm]$ matrix $\mathbf{H}_r$, Eq.(4.7), into the PMF, i.e. into the $[pm \times M]$ matrix **H** whose rows contain the coefficients of the polynomials $h_{ij}(z^{-1})$ of $H(z^{-1})$.

Using the duality principle, Algorithm *SSH* could also be implemented by:

```
    1. Set A,C (Qo) ⇒ Q_o  (Q_o has M-1 blocks C A^i of p rows.)
    2. Set [ D^T | (Q_o B)^T ]^T ⇒ H_c
    3. Set ‖H_{M-1}‖ ⇒ h_M
    4. Set H_c,p (PMFc) ⇒ H
```

Again, *PMFc* is a "service" algorithm that transforms $\mathbf{H}_c$ into **H** in the PMF. The reason for using Markov parameters as a "system model" is that Markov parameters are a convenient vehicle for intermodel conversions between different system models.

4.1.5 Continuous-Time State Space Response

This algorithm calculates the output response of a general C-T state space model $R = \{\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}\}$, given the initial state vector and samples of the input signals. The calculations are based on the assumption that the input signals are linearly interpolated between samples.

A general algorithm for simulating C-T systems is given in Section 1.3.5.

Syntax: $\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D},\mathbf{x}(0),\mathbf{u},T\ (CDSR) \Rightarrow \mathbf{y}$

4.1.6 Discrete-Time State Space Response

This algorithm calculates the output response of a general D-T state space model $R = \{\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}\}$, given the initial state vector and the sequence values of the input signals. The general algorithm capable of simulating D-T systems is given in Section 1.3.5.

Syntax: $\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D},\mathbf{x}(0),\mathbf{u},0\ (CDSR) \Rightarrow \mathbf{y}$

Note that in order to specify that the response of a D-T model is sought, the 7th input argument should be set to zero or any negative number.

4.1.7 Observable State Space to MFD Model

This algorithm converts an observable form $R_o = \{\mathbf{A}_o, \mathbf{B}_o, \mathbf{C}_o, \mathbf{D}_o\}$ to a left coprime column-reduced MFD, $D(z)^{-1}N(z)$. Of course, this algorithm can equally well be applied to a C-T state model to obtain the corresponding MFD representation. The algorithm furnishes a monic $D(z)$ in the sense of Definition 3.6, given in Chapter 3. Several of the intermodel conversions discussed in this chapter are based on the relationship between the state space model in a POF and a corresponding left coprime MFD. For this reason we will establish this relationship here and then discuss individual algorithms as they arise.

To this end, consider the order-n system with m-inputs and p-outputs:

$$\begin{aligned} \mathbf{x}(t+1) &= \mathbf{A}_o\mathbf{x}(t) + \mathbf{B}_o\mathbf{u}(t) \\ \mathbf{y}(t) &= \mathbf{C}_o\mathbf{x}(t) + \mathbf{D}_o\mathbf{u}(t) \end{aligned} \tag{4.9}$$

where t is used as an integer time index and $R_o = \{\mathbf{A}_o, \mathbf{B}_o, \mathbf{C}_o, \mathbf{D}_o\}$ is in a POF corresponding to a set of admissible POI, $\nu = \{\nu_i\}$. From Eq.(4.9) we may write

$$\begin{bmatrix} \mathbf{y}(t) \\ \mathbf{y}(t+1) \\ \vdots \\ \mathbf{y}(t+r) \end{bmatrix} = \begin{bmatrix} \mathbf{C}_o \\ \mathbf{C}_o\mathbf{A}_o \\ \vdots \\ \mathbf{C}_o\mathbf{A}_o^r \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} \mathbf{D}_o & 0 & \cdots & 0 & 0 \\ \mathbf{C}_o\mathbf{B}_o & \mathbf{D}_o & \cdots & 0 & 0 \\ & & \cdots & & \\ \mathbf{C}_o\mathbf{A}_o^{r-1}\mathbf{B}_o & \cdots & \mathbf{C}_o\mathbf{A}_o\mathbf{B}_o & \mathbf{C}_o\mathbf{B}_o & \mathbf{D}_o \end{bmatrix} \begin{bmatrix} \mathbf{u}(t) \\ \mathbf{u}(t+1) \\ \vdots \\ \mathbf{u}(t+r) \end{bmatrix} \tag{4.10}$$

Now we let $r = \nu_m = \max\{\nu_i\}$. Clearly, Eq.(4.10) holds for any $t = [0, N\text{-}r]$ and can be rewritten as

$$\mathbf{y}_t = \mathbf{Q}_{ao}\mathbf{x}(t) + \mathbf{H}\,\mathbf{u}_t \tag{4.11}$$

where $\mathbf{y}_t$ and $\mathbf{u}_t$ are $(\nu_m+1)p$ and $(\nu_m+1)m$ dimensional columns containing the output and input vectors $\mathbf{y}(t+j)$ and $\mathbf{u}(t+j)$, $j = [0, \nu_m]$. The matrix $\mathbf{Q}_{ao}$ is the observability matrix of the pair $\{\mathbf{A}_o, \mathbf{C}_o\}$, while $\mathbf{H}$ is the $(r+1)p \times (r+1)m$ lower block triangular matrix containing along the main diagonal the $(p\times m)$ blocks $\mathbf{D}_o$. The other nonzero blocks of $\mathbf{H}$ are the $p\times m$ dimensional *Markov parameters*:

$$\mathbf{C}_o\mathbf{A}_o^j\mathbf{B}_o\,, \quad \text{for } j = [0, \nu_m-1] \tag{4.12}$$

Our goal is to eliminate from Eq.(4.10) the $\mathbf{x}(t)$ terms, thereby obtaining an expression which relates the sampled data to the elements in R_o.

Equation (4.10) can be considered to represent $(\nu_m+1)p$ scalar equations in the samples

$$y_{ij} = y_i(t+j) \tag{4.13}$$

i.e. the i^{th} component of the output vector $\mathbf{y}(t+j)$, $i=[1, p]$, $j=[0, \nu_m]$. In Section 3.4.3 it was shown that $\mathbf{Q}_{ao}$ has n rows of the identity matrix $\mathbf{I}_n$ and p rows that correspond to the rows of $\mathbf{A}_o$ with non-zero/non-unity elements. Furthermore, the location of these rows are determined by the selector vectors $\mathbf{v}_li$ and $\mathbf{v}_{ld}$, respectively.

Premultiplying Eq.(4.11) by the selector matrices $\mathbf{S}_{li}^T$ and $\mathbf{S}_{ld}^T$ defined by Eq.(3.79), we obtain, respectively,

$$\mathbf{y}_{1t} = \mathbf{x}(t) + \mathbf{H}_1\,\mathbf{u}_t\,, \quad \text{and} \quad \mathbf{y}_{2t} = \mathbf{A}_r\mathbf{x}(t) + \mathbf{H}_2\mathbf{u}_t \tag{4.14}$$

where

$$\mathbf{y}_{1t} = \mathbf{S}_{li}^T\mathbf{y}_t\,, \quad \mathbf{y}_{2t} = \mathbf{S}_{ld}^T\mathbf{y}_t \quad \text{with} \quad \mathbf{H}_1 = \mathbf{S}_{li}^T\mathbf{H}\,, \quad \mathbf{H}_2 = \mathbf{S}_{ld}^T\mathbf{H}$$

Eliminating $\mathbf{x}(t)$ from Eq.(4.14),

$$\mathbf{y}_{2t} = \begin{bmatrix} (\mathbf{H}_2 - \mathbf{A}_r\mathbf{H}_1) & \mathbf{A}_r \end{bmatrix} \begin{bmatrix} \mathbf{u}_t \\ \mathbf{y}_{1t} \end{bmatrix} \tag{4.15}$$

The matrix $\mathbf{A}_r$ in Eqs.(4.14) and (4.15) is a $(p\times n)$ matrix containing the rows of $\mathbf{A}_o$ with non-zero non-unity elements, whose locations in $\mathbf{A}_o$ are specified by the selector vector $\mathbf{v}_d$. Equation (4.15) may be expressed in a more concise form by

$$\mathbf{y}_{2k} = \left[\ \mathbf{N}_r \quad \mathbf{A}_r\ \right]\mathbf{z}_k \tag{4.16}$$

where $\mathbf{N}_r = \mathbf{H}_2 - \mathbf{A}_r\mathbf{H}_1$ is a $p \times (\nu_m+1)m$ matrix and $\mathbf{z}_k$ is an h-dimensional vector containing $\mathbf{u}_t$ and $\mathbf{y}_{1t}$, where $h = (\nu_m+1)m + n$. Equation (4.16) is referred to as the *identification identity* since it relates input-output data samples arranged into columns $\mathbf{y}_{2t}$ and $\mathbf{z}_t$ to parameters of the state space representation R_{o}, i.e. in the matrices $\mathbf{A}_o$, $\mathbf{B}_o$ and $\mathbf{D}_o$. The *identification identity* is the basis for conversions between input/output data and either state space or MFD models.

For the purpose of Algorithm *RoDN*, to be introduced here, Eq.(4.16) may be rewritten as

$$\mathbf{y}_{2t} - \mathbf{A}_r\mathbf{y}_{1t} = \mathbf{N}_r\mathbf{u}_t \tag{4.17}$$

Note that Eq.(4.17) is a time-domain input/output expression. Applying the z-transform and taking into account the arrangements of the samples $u_i(t+j)$ and $y_i(t+j)$ in the vectors $\mathbf{u}_t$, $\mathbf{y}_{1t}$ and $\mathbf{y}_{2t}$, we obtain:

$$D(z)\,y(z) = N(z)\,u(z) \tag{4.18}$$

which is a left coprime MFD. Since in Eq.(4.17) the p dimensional vector $\mathbf{y}_{2t}$ is multiplied by the identity matrix $\mathbf{I}_p$, it may be concluded that $D(z)$ in Eq.(4.18) is monic. For further details see Section 3.4 and Eq.(3.104). Thus, in order to obtain the $[p \times (\nu_m+1)p]$ matrix $\mathbf{D}_r$, which leads directly to $D(z)$, it is first necessary to obtain the matrix $\mathbf{A}_r$. From the discussion in Section 3.3.4 it is clear that $\mathbf{A}_r$ may be obtained from $\mathbf{A}_o$ in a POF by:

$$\mathbf{S}_a^T\mathbf{A}_o = \mathbf{A}_r \tag{4.19}$$

where $\mathbf{S}_a$ is one of the selector matrices uniquely defined by the particular set of admissible POI $\boldsymbol{\nu}$ and generated by Algorithm *SMat*:

$$\boldsymbol{\nu}\ (SMat) \Rightarrow \nu_m, \mathbf{S}_a, \mathbf{S}_i, \mathbf{S}_{li}, \mathbf{S}_{ld}$$

Then the matrix $\mathbf{D}_r$ becomes

$$\mathbf{S}_{ld}^T - \mathbf{A}_r\mathbf{S}_{li}^T = \mathbf{D}_r \tag{4.20}$$

For more details see Section 3.3.4.

In order to obtain the corresponding $\mathbf{N}(z)$ in the left coprime pair $\{D(z),N(z)\}$, recall that:

$$G(z) = D^{-1}(z)\,N(z)$$

Thus, using $G(z) = W(z)/d(z)$ and $D^{-1}(z) = T(z)/d(z)$, where $T(z) = \operatorname{adj} D(z)$, $d(z) = \det(z\mathbf{I} - \mathbf{A}_o) = \det\{\ D(z)\ \}$ and $W(z) = \mathbf{C}\operatorname{adj}(z\,\mathbf{I} - \mathbf{A}_o)\,\mathbf{B} + d(z)\,\mathbf{D}$, the above equation may be expressed by:

$$W(z) = T(z)\,N(z) \tag{4.21}$$

leading to: $$\sum_{j=0}^{i} \mathbf{T}_{i-j}\mathbf{N}_j = \mathbf{W}_i\ , \quad \text{for} \quad i = [1,n]$$

(with $\mathbf{N}_j = \mathbf{0}$ for $j > k$, where $k = \nu_m$) which may be represented by:

$$\begin{bmatrix} \mathbf{T}_0 & & \\ & \ddots & \\ & & \mathbf{T}_0 \\ \vdots & & \vdots \\ \mathbf{T}_n & \cdots & \mathbf{T}_{n-k} \end{bmatrix} \begin{bmatrix} \mathbf{N}_0 \\ \vdots \\ \mathbf{N}_k \end{bmatrix} = \begin{bmatrix} \mathbf{W}_0 \\ \vdots \\ \mathbf{W}_n \end{bmatrix} \tag{4.22}$$

where $\mathbf{T}_i$, $\mathbf{N}_i$ and $\mathbf{W}_i$ are corresponding real number submatrices in the polynomial matrices $T(z)$, $N(z)$ and $W(z)$.

From the above discussion it is not difficult to conclude that given R_o and the associated set ν, the following algorithm will calculate the corresponding left coprime MFD $\{D(z), N(z)\}$.

Syntax: $\mathbf{A}_o, \mathbf{B}_o, \mathbf{C}_o, \mathbf{D}_o, \nu$ *(RoDN)* $\Rightarrow \mathbf{D}, \mathbf{N}$

```
1. Set $\nu$ $(SMat) \Rightarrow \nu_m, \mathbf{S}_a, \mathbf{S}_i, \mathbf{S}_b, \mathbf{S}_M$
2. Set $\mathbf{S}_a^T \mathbf{A}_o \Rightarrow \mathbf{A}_r$
3. Set $\mathbf{S}_M^T - \mathbf{A}_r\mathbf{S}_b^T \Rightarrow \mathbf{D}_r$
4. Set $\mathbf{D}_r, p$ $(PMFr) \Rightarrow \mathbf{D}$
5. Set adj $D(z) \Rightarrow T(z)$
6. Set $\mathbf{A}_o, \mathbf{B}_o, \mathbf{C}_o, \mathbf{D}_o$ $(SSTF) \Rightarrow \mathbf{d}, \mathbf{W}$
7. From $T(z)$ and $\mathbf{W}$, ( $\mathbf{T}_i$ and $\mathbf{W}_i$ ) build Eq.(4.22) and solve for $\mathbf{N}_c$
8. Set $\mathbf{N}_c, p$ $(PMFc) \Rightarrow \mathbf{N}$
```

As was mentioned earlier, the Algorithm *PMFr* used in Step 4 is a polynomial matrix "service" algorithm which simply transforms $\mathbf{D}_r$ into the PMF, i.e. into the $[p^2 \times (\nu_m+1)]$ matrix $\mathbf{D}$ whose rows contain coefficients of the polynomials $d_{ij}(z)$ of $D(z)$. From Eq.(4.22) it is clear that the submatrices $\mathbf{N}_i$ are in the form of $\mathbf{N}_c$, i.e.

$$\mathbf{N}_c = \begin{bmatrix} \mathbf{N}_0 \\ \vdots \\ \mathbf{N}_k \end{bmatrix}$$

This is why the service algorithm *PMFc* must be used in Step 8.

From the discussion in Section 3.4, as well as from the above algorithm, it may be concluded that the column degrees $\{n_i\}$ of the monic $D(z)$ of the left coprime MFD $\{D(z), N(z)\}$ are equal to the POI used, i.e. $\{n_i\} = \{\nu_i\} = \nu$.

Example:

The purpose of this example is to emphasize the relationships between a given $\mathbf{A}_o$ in a POF and $D_l(z)$ of the corresponding left coprime MFD $\{D_l(z), N_l(z)\}$, where $D_l(z)$ is monic and column-reduced with column degrees $\{n_i\}$ equal to the set of POI $\nu = \{\nu_i\}$ used in building $\mathbf{A}_o$ (in a POF):

$\mathbf{A}_{or}$ (POI = {1,3}):

$$\mathbf{A}_o = \begin{bmatrix} -2.000 & .002 & .003 & .001 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1.000 & -5.001 & -9.001 & -5.000 \end{bmatrix}, \quad \begin{bmatrix} 1 \\ 0 \\ 0 \\ 1 \end{bmatrix}^T = \mathbf{v}_o$$

$$\mathbf{A}_{or} = \begin{bmatrix} -2.000 & .002 & .003 & .001 \\ 1.000 & -5.001 & -9.001 & -5.000 \end{bmatrix}$$

where at the right of $\mathbf{A}_o$ is a selector vector $\mathbf{v}_o$ specifying the non-zero/non-unity rows of $\mathbf{A}_o$ which are used to form the matrix $\mathbf{A}_{or}$.

$\mathbf{D}_{lr}$ (column degrees {1,3}):

$$\mathbf{D}_{lr} = \begin{bmatrix} 2.000 & -.002 & 1 & -.003 & 0 & -.001 & 0 & 0 \\ -1.000 & 5.001 & 0 & 9.001 & 0 & 5.000 & 0 & 1 \end{bmatrix}$$
$$\mathbf{v}_{ll} = [\ 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0\ ]$$

where the selector vector $\mathbf{v}_{ll}$ marks the non-zero/non-unity columns of $\mathbf{D}_{lr}$. When only these columns are selected, $\mathbf{D}_{lc}$ is formed:

$$\mathbf{D}_{lc} = \begin{bmatrix} 2.000 & -.002 & -.003 & -.001 \\ -1.000 & 5.001 & 9.001 & 5.000 \end{bmatrix}$$

Note that $\mathbf{A}_{or} = -\mathbf{D}_{lc}$. The corresponding polynomial matrix $D_l(z)$ can be constructed from $\mathbf{D}_{lr} = [\ \mathbf{D}_0\ \ \mathbf{D}_1\ \ \mathbf{D}_2\ \ \mathbf{D}_3\ ]$ where $\{\mathbf{D}_i\}$ are 2×2 partitions associated with the coefficients of s^i. Thus,

$$D_l(s) = \begin{bmatrix} 2.000 + 1s & -.002 - .003s - .001s^2 \\ -1.000 & 5.001 + 9.001s + 5.000s^2 + 1s^3 \end{bmatrix}$$

The selector vectors corresponding to $\{\nu_i\} = \{n_i\} = \{1,3\}$ are:

$$\begin{aligned}
\mathbf{v}_a &= [\,1 \quad 0 \quad 0 \quad 1\,] \\
\mathbf{v}_i &= [\,0 \quad 1 \quad 1 \quad 0\,] \\
\mathbf{v}_{li} &= [\,1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0\,] \\
\mathbf{v}_{ld} &= [\,0 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0 \quad 1\,]
\end{aligned}$$

To formalize these ideas, we define the following:

Remark 4.1 For a given system the total number of *equivalent left coprime MFDs* with column reduced and monic $D(z)$, having column degrees $\{n_i\}$, is equal to the total number of POFs based on admissible sets of POI $\{\nu_i\}$. Thus, it may be said that there is a one-to-one correspondence between a POF and associated left coprime MFD satisfying:

$$\{\,\nu_i\,\} = \{\,n_i\,\}$$

4.1.8 Controllable State Space to MFD Model

This algorithm converts a controllable form $R_c = \{\mathbf{A}_c, \mathbf{B}_c, \mathbf{C}_c, \mathbf{D}_c\}$ to a right coprime row-reduced MFD, $N(z)D^{-1}(z)$. As with the previous case, this algorithm can be applied to a C-T state model to obtain the corresponding MFD representation. This algorithm is dual to Algorithm *RoDN*. Thus, it may be easily verified that given R_c in a PCF, based on an admissible set of PCI μ, the following algorithm calculates a corresponding right coprime $\{\, N(z), D(z)\,\}$ where $D(z)$ is row-reduced and monic, satisfying:

$$G(z) = \mathbf{C}_c\,(z\mathbf{I} - \mathbf{A}_c)^{-1}\,\mathbf{B}_c + \mathbf{D}_c = N(z)\,D^{-1}(z) \tag{4.23}$$

Syntax: $\mathbf{A}_c, \mathbf{B}_c, \mathbf{C}_c, \mathbf{D}_c, \mu$ (*RcND*) $\Rightarrow$ **N**, **D**

with the basic steps:

1. Set μ (*SMat*) $\Rightarrow \mu_m, \mathbf{S}_a, \mathbf{S}_i, \mathbf{S}_{li}, \mathbf{S}_{ld}$
2. Set $\mathbf{A}_c\mathbf{S}_a \Rightarrow \mathbf{A}_{ac}$
3. Set $\mathbf{S}_{ld} - \mathbf{S}_{li}\mathbf{A}_{ac} \Rightarrow \mathbf{D}_{rc}$

```
4. Set $\mathbf{D}_{rc}, m$ (PMFc) $\Rightarrow$ D
5. Set adj $D(z) \Rightarrow T(z)$
6. Set $\mathbf{A}_r, \mathbf{B}_r, \mathbf{C}_r, \mathbf{D}_r$ (SSTF) $\Rightarrow$ d, W
7. From $T(z)$ and W, ( $\mathbf{T}_i$ and $\mathbf{N}_i$ ) build Eq.(4.25) and solve for $\mathbf{N}_r$
8. Set $\mathbf{N}_r, m$ (PMFr) $\Rightarrow$ N
```

Now, $\mathbf{A}_{cr}$ in Step 2 is an $(n \times m)$ matrix containing m columns from $\mathbf{A}_c$ with non-zero and non-unity elements, see Eq.(3.66), while $\mathbf{D}_{rr}$ is a $[(\mu_m+1)m \times m]$ matrix containing $(m \times m)$ submatrices $\mathbf{D}_i$ of $D(z)$. The structure of $\mathbf{D}_{rr}$ is dual to that of $\mathbf{D}_r$ shown in Eq.(3.104).

Similarly, by duality, instead of Eq.(4.21) we have now

$$W(z) = N(z)\, T(z) \tag{4.24}$$

leading to:

$$\sum_{j=0}^{i} \mathbf{N}_j \mathbf{T}_{i-j} = \mathbf{W}_i\,, \quad \text{for} \quad i = [1,n]$$

(with $\mathbf{N}_j = \mathbf{0}$ for $j > k$) which may be represented by:

$$[\mathbf{N}_0 \cdots \mathbf{N}_k]\begin{bmatrix} \mathbf{T}_0 & & \cdots & \mathbf{T}_n \\ & \ddots & & \vdots \\ & & \mathbf{T}_0 \;\cdots & \mathbf{T}_{n-k} \end{bmatrix} = [\mathbf{W}_0 \quad \cdots \quad \mathbf{W}_n] \tag{4.25}$$

where $k = \mu_m$ while $\mathbf{T}_i$, $\mathbf{N}_i$ and $\mathbf{W}_i$ are corresponding real number submatrices in the polynomial matrices $T(z)$, $N(z)$ and $W(z)$.

From Eq.(4.25) it is clear that submatrices $\mathbf{N}_i$ are in the form of $\mathbf{N}_r$, i.e.

$$\mathbf{N}_r = [\mathbf{N}_0 \cdots \mathbf{N}_k]$$

This is why the service algorithm *PMFr* has to be used in Step 8. It is important to again note that the row degrees $\{n_i\}$ of $D(z)$ equal the PCI, i.e. $\{n_i\} = \{\mu_i\} = \mu$. As in the previous section an example will be used to illustrate these results.

Example:

Consider the relationships between a given $\mathbf{A}_c$ in a PCF and $D_r(z)$ of the corresponding right coprime transfer function $\{N_r(z), D_r(z)\}$, where $D_r(z)$ is monic and row-reduced with row degrees $\{n_i\}$ equal to the set of PCI $\mu = \{\mu_i\}$ used in building $\mathbf{A}_c$ (in a PCF):

$\mathbf{A}_c$ (PCI = {1,2,1}):

$$\mathbf{A}_c = \begin{bmatrix} -.999 & 0 & .000 & 1.000 \\ -.003 & 0 & 1.000 & -5.002 \\ .000 & 0 & -2.000 & .000 \\ -.001 & 1 & .000 & -4.001 \end{bmatrix}$$

$$\mathbf{v}_a = [\ 1 \quad 0 \quad 1 \quad 1\ ]$$

The selector vector $\mathbf{v}_a$ specifies the non-zero/non-unity columns of $\mathbf{A}_c$. These are collected to form $\mathbf{A}_{cc}$:

$$\mathbf{A}_{cc} = \begin{bmatrix} -.999 & .000 & 1.000 \\ -.003 & 1.000 & -5.002 \\ .000 & 2.000 & .000 \\ -.001 & .000 & -4.001 \end{bmatrix}$$

The matrix $\mathbf{D}_{rc}$ containing the submatrices $\mathbf{D}_i$, $i=[0,k]$, of $D(z)$ is:

$$\mathbf{D}_{rc} = \begin{bmatrix} \mathbf{D}_0 \\ \mathbf{D}_1 \\ \mathbf{D}_2 \end{bmatrix} = \begin{bmatrix} .999 & .000 & -1.000 \\ .003 & -1.000 & 5.002 \\ .000 & 2.000 & .000 \\ \hline 1 & 0 & 0 \\ .001 & .000 & 4.001 \\ 0 & 1 & 0 \\ \hline 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \quad \begin{bmatrix} 1 \\ 1 \\ 1 \\ \\ 0 \\ 1 \\ 0 \\ \\ 0 \\ 0 \\ 0 \end{bmatrix}^T = \mathbf{v}_{ll}$$

Note that the selector vector $\mathbf{v}_{ll}$ marks the non-zero/non-unity rows of $\mathbf{D}_{rc}$. When these rows are collected into a matrix $\mathbf{D}_{rr}$, it is clear that $\mathbf{D}_{rr} = -\mathbf{A}_{cc}$. As in the previous section $D_r(s)$ can be formed, this time from the coefficients in $\mathbf{D}_{rc}$. Thus,

$$D_r(s) = \begin{bmatrix} .999 + 1s & .000 & -1.000 \\ .003 + .001s & -1.000 & 5.002 + 4.001s + 1s^2 \\ .000 & 2.000 + 1s & .000 \end{bmatrix}$$

The selector vectors corresponding to $\{\mu_i\} = \{n_i\} = \{1,2,1\}$ are:

$$\begin{aligned}
\mathbf{v}_a &= [\,1 \quad 0 \quad 1 \quad 1\,] \\
\mathbf{v}_i &= [\,0 \quad 1 \quad 0 \quad 0\,] \\
\mathbf{v}_{li} &= [\,1 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 0 \quad 0 \quad 0\,] \\
\mathbf{v}_{ld} &= [\,0 \quad 0 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 0\,]
\end{aligned}$$

As in Remark 4.1 we would like to formalize the ideas for this dual case:

Remark 4.2 For a given system the total number of *equivalent right coprime MFDs* with row reduced and monic $D(z)$, having row degrees $\{n_i\}$, is equal to the total number of PCFs based on admissible sets of PCI $\{\mu_i\}$. Thus, it may be said that there is a one-to-one correspondence between a PCF and associated right coprime MFD satisfying:

$$\{\mu_i\} = \{n_i\}$$

4.2 Conversions from a Transfer Function Matrix

Beginning with a transfer function matrix model, we may want to convert it to a state space form; three useful versions of this conversion will be discussed in this section. In addition, calculations of the system response as well as the Markov parameters are presented. At the end of this section a novel approach to minimal realization will be given.

4.2.1 Transfer Function to State Space Model

This algorithm transforms a transfer function matrix $G(s) = \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D}$ to a specific state space form $R = \{\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}\}$. The conversion into state space is a *minimal realization*, i.e. the state space model is of minimum order. The options include conversion into a *Hessenberg form*, a *Kalman decomposition* or a *Jordan form*, which will be explained in detail. The above mentioned minimal realization procedures require as input arguments a non-minimal (uncontrollable or unobservable or both uncontrollable and unobservable) state space representation $R = \{\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}\}$. Therefore, the model conversion:

$$TF \rightarrow SS$$

will be performed in the following two steps.

1. Conversion from transfer function matrix $G(z) = W(z)/d(z)$ into a non-minimal representation $R = \{\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}\}$, i.e.

$$TF \Rightarrow R$$

2. Minimal realization procedure, i.e. conversion from R into a minimal representation $R_m = \{\mathbf{A}_m, \mathbf{B}_m, \mathbf{C}_m, \mathbf{D}\}$, i.e.

$$R \Rightarrow R_m$$

Note that in R_m the direct through matrix $\mathbf{D}$ is unchanged. Therefore, in our minimal realization procedures we will consider only the strictly proper part of the system, i.e. only matrices $\mathbf{A}$, $\mathbf{B}$ and $\mathbf{C}$. Also, we will assume that the given transfer function $\tilde{G}(z)$ is "strictly" proper, i.e. that polynomial matrix $\tilde{W}(z)$ and polynomial $d(z)$ satisfy:

$$\frac{\tilde{W}(z)}{d(z)} = \mathbf{C}_m(z\mathbf{I} - \mathbf{A}_m)^{-1}\mathbf{B}_m\,, \text{ or } \tilde{W}(z) = \mathbf{C}_m \operatorname{adj}(z\mathbf{I} - \mathbf{A}_m)\mathbf{B}_m \quad (4.26)$$

Of course, the non-strictly proper transfer function matrix $G(z)$ is related to $\tilde{G}(z)$ by: $G(z) = \tilde{G}(z) + \mathbf{D}$. It is worth mentioning at this point that the "extraction" of the strictly proper part $\tilde{G}(z)$, or $\tilde{W}(z)$ from a given non-strictly proper transfer function matrix $G(z) = W(z)/d(z)$ may be performed by a simple procedure symbolically represented by:

$$\mathbf{d},\ \mathbf{W}\ (ExD) \Rightarrow \tilde{\mathbf{W}},\ \mathbf{D}$$

The implementation of this procedure is based on the following equations:

$$W(z) = \sum_{i=0}^{n} \mathbf{W}_i z^i\,,\quad \mathbf{D} = \mathbf{W}_n\,,\quad \tilde{W}(z) = W(z) - \sum_{i=0}^{n} d_i z^i \mathbf{D}$$

Obviously, since $d_n = 1$, all polynomials in $\tilde{W}(z)$ are of up to $(n\text{-}1)^{st}$ order. Thus, the transfer function matrix $\tilde{G}(z) = \tilde{W}(z)/d(z)$ is strictly proper.

The easiest way of performing the conversion $TF \Rightarrow R$ is to build either:

(1) —a controllable, but not necessarily observable, representation $R_1 = \{\mathbf{A}_1, \mathbf{B}_1, \mathbf{C}_1\}$, or

(2) —an observable, but not necessarily controllable, representation $R_2 = \{\mathbf{A}_2, \mathbf{B}_2, \mathbf{C}_2\}$

It may be shown that the following two representations R_1 and R_2 with m inputs and p outputs of orders nm and np, respectively, where n is the order of the characteristic polynomial $d(z)$, each have a transfer function matrix equal to the given strictly proper matrix $\bar{G}(z)=\bar{W}(z)/d(z)$. To facilitate the understanding of the process of building these representations, a generic example of a system with $m = 2$ and $p = 3$ is considered. Generalization to different values of m and p is straightforward.

Conversions from a Transfer Function Matrix to a (Non-Minimal) State Space Representation: $TF \Rightarrow R$

Here we will consider building non-minimal state space representations from a given *strictly proper* transfer function matrix $G(z)=W(z)/d(z)$ of a MIMO system. In the case of an n^{th} order MIMO system with $m=2$ and $p=3$, the $(pm \times n)$ matrix $\mathbf{W}$ in the PMF has a structure

$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_{11} \\ \mathbf{w}_{21} \\ \mathbf{w}_{31} \\ ---- \\ \mathbf{w}_{12} \\ \mathbf{w}_{22} \\ \mathbf{w}_{32} \end{bmatrix}$$

where $\mathbf{w}_{ij}$ is an n-dimensional row containing n coefficients defining the $(n-1)^{st}$ order polynomial $w_{ij}(z)$ in $W(z) = \{ w_{ij}(z) \}$. The first n coefficients d_i, $i=[0,n-1]$, from $d(z)$ are arranged in the n dimensional row $\mathbf{a}$.

A controllable, but not necessarily observable, representation $R_1 = \{\mathbf{A}_1, \mathbf{B}_1, \mathbf{C}_1\}$ is:

$$\mathbf{A}_1 = \text{diag}\{ \mathbf{A}_c \} , \quad \mathbf{B}_1 = \text{diag}\{ \mathbf{b}_c \} \ m \text{ times:}$$

$$\mathbf{A}_1 = \begin{bmatrix} \mathbf{A}_c & \\ & \mathbf{A}_c \end{bmatrix} \quad \mathbf{B}_1 = \begin{bmatrix} \mathbf{b}_c & \\ & \mathbf{b}_c \end{bmatrix} \quad \mathbf{C}_1 = \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix} \qquad (4.27)$$

where $\mathbf{A}_1$ is $nm \times nm$, $\mathbf{B}_1$ is $nm \times m$ and $\mathbf{C}_1$ is $p \times nm$. $\{\mathbf{A}_c, \mathbf{b}_c\}$ in Eq.(4.27) is a controllable pair from the SISO feedback canonical form, Eq.(3.13). Recall that $\mathbf{A}_c$ contains in its last row the row $-\mathbf{a}$ consisting of the coefficients d_i of $d(z)$. All n roots of $d(z)$, or the n eigenvalues of $\mathbf{A}_c$, appear in $\mathbf{A}_1$ as multiple eigenvalues with multiplicity m. The nmth order representation R_1 is controllable, but if $m > 1$, it is unobservable. The number of unobservable modes is equal to $(m-1)n$.

Similarly, an observable, but not necessarily controllable, representation $R_2 = \{\mathbf{A}_2, \mathbf{B}_2, \mathbf{C}_2\}$ has the dual structure:

$$\mathbf{A}_2 = \operatorname{diag}[\,\mathbf{A}_o\,]\ ,\quad \mathbf{C}_2 = \operatorname{diag}[\,\mathbf{c}_o\,]\ p \text{ times:}$$

$$\mathbf{A}_2 = \begin{bmatrix} \mathbf{A}_o & & \\ & \mathbf{A}_o & \\ & & \mathbf{A}_o \end{bmatrix} \quad \mathbf{B}_2 = \begin{bmatrix} w_{11}^T & w_{12}^T \\ w_{21}^T & w_{22}^T \\ w_{31}^T & w_{32}^T \end{bmatrix} \quad \mathbf{C}_2 = \begin{bmatrix} \mathbf{c}_o & & \\ & \mathbf{c}_o & \\ & & \mathbf{c}_o \end{bmatrix} \tag{4.28}$$

where $\mathbf{A}_2$ is $np \times np$, $\mathbf{B}_2$ is $np \times m$ and $\mathbf{C}_2$ is $p \times np$. $\{\mathbf{A}_o, \mathbf{c}_o\}$ in Eq.(4.28) is an observable pair from the SISO observable canonical form, Eq.(3.18). All n roots of $d(z)$, or the n eigenvalues of $\mathbf{A}_o$, appear in $\mathbf{A}_2$ as multiple eigenvalues with multiplicity p. The npth order representation R_2 is observable, but if $p > 1$, it is uncontrollable. The number of uncontrollable modes is equal to $(p-1)n$.

The construction of these representations will be represented by the algorithms:

$$d,\ \mathbf{W}\ (\mathit{TRcn}) \Rightarrow \mathbf{A}_1,\ \mathbf{B}_1,\ \mathbf{C}_1$$

and

$$d,\ \mathbf{W}\ (\mathit{TRon}) \Rightarrow \mathbf{A}_2,\ \mathbf{B}_2,\ \mathbf{C}_2$$

where $R_1 = \{\mathbf{A}_1,\mathbf{B}_1,\mathbf{C}_1\}$ corresponds to R_1, while $R_2 = \{\mathbf{A}_2,\mathbf{B}_2,\mathbf{C}_2\}$ is equal to R_2 given above. Note that when $m < p$, it is more convenient to use Algorithm *TRcn*, since then the order of matrix $\mathbf{A}_1$ in R_1 is smaller than that of $\mathbf{A}_2$ in R_2.

Conversions from a Non-Minimal State Space Representation to a Minimal State Space Representation: $TF \Rightarrow R$

As far as the minimal realization procedures are concerned, it will suffice to mention here that the following algorithms are available:

$$\mathbf{A},\ \mathbf{B},\ \mathbf{C},\ \epsilon\ (MIN) \Rightarrow \mathbf{A}_m,\ \mathbf{B}_m,\ \mathbf{C}_m$$

$$\mathbf{A},\ \mathbf{B},\ \mathbf{C},\ \epsilon\ (KALD) \Rightarrow \mathbf{A}_m,\ \mathbf{B}_m,\ \mathbf{C}_m$$

$$\mathbf{A},\ \mathbf{B},\ \mathbf{C},\ \epsilon\ (JFOR) \Rightarrow \mathbf{A}_m,\ \mathbf{B}_m,\ \mathbf{C}_m$$

i.e., performing the conversion $R \Rightarrow R_m$ using a Hessenberg, Kalman decomposition or a Jordan form approach, respectively. Detailed descriptions of these approaches are given in Appendix B. Minimal representations obtained by these approaches are not necessarily in a canonical form. So, if a specific procedure requires a model in POF or PCF, then, of course, either *SSRc* or *SSRo* should subsequently be used.

Examples

As an illustration of using the above mentioned minimal realization algorithms, consider the following SISO, strictly proper, uncontrollable and unobservable model $R = \{\mathbf{A}, \mathbf{B}, \mathbf{C}\}$.

$$\left[\begin{array}{cccccccc|c}
-.75 & 1.00 & .21 & -.01 & .38 & -.03 & .07 & .37 & .99 \\
-1.08 & -1.24 & -.09 & .03 & .10 & .00 & .28 & .04 & .81 \\
.03 & -.07 & -.98 & .06 & .32 & .23 & .38 & .30 & .82 \\
-.18 & -.10 & -.08 & -2.06 & .07 & .06 & -.04 & -.51 & .79 \\
.16 & .11 & .01 & .19 & -1.89 & .92 & .40 & -.10 & -.04 \\
-.15 & -.28 & -.20 & -.23 & -1.34 & -2.11 & -.41 & -.65 & -.49 \\
-.05 & .09 & .52 & .54 & .23 & .02 & -1.12 & 2.59 & -.09 \\
.06 & -.14 & -.14 & -.46 & -.44 & -.38 & -2.05 & -2.82 & -.27 \\
\hline
1.59 & 1.05 & .28 & 1.54 & 1.57 & 1.32 & .58 & .42 & 0
\end{array}\right]$$

The representation R has been obtained from the "auxiliary" $\tilde{R} = \{\tilde{\mathbf{A}}, \tilde{\mathbf{B}}, \tilde{\mathbf{C}}\}$ where:

$$\left[\begin{array}{cccccccc|c}
-1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
-1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & -2 & 0 & 0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 & -2 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -1 & -2 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -2 & 2 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & -2 & -2 & 0 \\
\hdashline
1 & 1 & 0 & 1 & 1 & 1 & 0 & 0 & 0
\end{array}\right]$$

by the similarity tansformation

$$\tilde{\mathbf{A}},\ \tilde{\mathbf{B}},\ \tilde{\mathbf{C}},\ \mathbf{T}\ (STR) \Rightarrow \mathbf{A},\ \mathbf{B},\ \mathbf{C}$$

where $\mathbf{T}$ was a "random" $(n \times n)$ similarity transformation matrix.

The Jordan form minimal representation R_m has been obtained by first transforming R into the Jordan form $R_J = \{\mathbf{A}_J, \mathbf{B}_J, \mathbf{C}_J\}$. This was done by a similarity transformation:

$$\mathbf{A},\ \mathbf{B},\ \mathbf{C},\ \mathbf{M}\ (STR) \Rightarrow \mathbf{A}_J,\ \mathbf{B}_J,\ \mathbf{C}_J \tag{4.29}$$

where the $(n \times n)$ similarity transformation matrix $\mathbf{M}$, sometimes referred to as the *modal matrix*, contains eigenvectors associated with all n eigenvalues λ_i of $\mathbf{A}$, i.e. the columns of $\mathbf{M} = [\mathbf{m}_1 \ \dots \ \mathbf{m}_n]$ satisfy

$$(\lambda_i \mathbf{I}_n - \mathbf{A})\,\mathbf{m}_i = \mathbf{0}$$

i.e. $\mathbf{m}_i$ is in the null space of $\mathbf{B}_i = \lambda_i\mathbf{I} - \mathbf{A}$.

Using Eq.(4.29), the representation R_J becomes:

$$\left[\begin{array}{cccccccc|c}
-2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-2 & -2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -2 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & -1 & -2 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & -2 & 0 & 0 & 0 & .54 \\
0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 & -.95 \\
0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & -1.20 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & .86 \\
\hdashline
0 & 0 & -.33 & -1.35 & -1.82 & -.80 & -1.02 & 0 & 0
\end{array}\right]$$

Obviously, the first four modes in R_J are uncontrollable, since the corresponding elements in the vector $\mathbf{B}_J$ are equal to zero. Similarly (dually) the modes with indices 1, 2 and 8 are unobservable. In order to extract from R_J the minimal part, i.e. the modes which are both controllable and observable, the following selector vector $\mathbf{v} = \{ v_i \}$ has been generated:

$$\mathbf{v} = [\ 0 \quad 0 \quad 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0\ ]^T$$

The elements v_i are calculated by:

$$v_i = 1 \text{ if } b_{J_i} c_{J_i} > \epsilon, \text{ otherwise } v_i = 0.$$

Finally, the minimal representation R_{Jm} obtained using the Jordan form approach is:

$$\mathbf{A}_{Jm} = \begin{bmatrix} -2.0 & .0 & .0 \\ .0 & -1.0 & 1.0 \\ .0 & -1.0 & -1.0 \end{bmatrix} \qquad \mathbf{B}_{Jm} = \begin{bmatrix} .547 \\ -.953 \\ -1.205 \end{bmatrix}$$
$$\mathbf{C}_{Jm} = [\ 1.829 \quad -.807 \quad -1.021\ ]$$

The sequence of algorithms is:

$$\mathbf{v}\ (DSM) \Rightarrow \mathbf{S}$$
$$\mathbf{S}^T \mathbf{A}_J \mathbf{S} \Rightarrow \mathbf{A}_{Jm}$$
$$\mathbf{S}^T \mathbf{B}_J \Rightarrow \mathbf{B}_{Jm}$$
$$\mathbf{C}_J \mathbf{S} \Rightarrow \mathbf{C}_{Jm}$$

The selector matrix $\mathbf{S}$ has dimension (8×3) containing the 5th, 6th and 7th columns of $\mathbf{I}_8$. A service algorithm, *DSM*, (Define Selector Matrix), is used to generate the matrix $\mathbf{S}$.

The Kalman decomposition procedure generates the minimal representation $R_{Km} = \{\mathbf{A}_{Km}, \mathbf{B}_{Km}, \mathbf{C}_{Km}\}$:

$$\mathbf{A}_{Km} = \begin{bmatrix} -2.01 & .10 & -.11 \\ -.22 & -.92 & 1.28 \\ .09 & -.77 & -1.06 \end{bmatrix} \qquad \mathbf{B}_{Km} = \begin{bmatrix} -1.03 \\ .09 \\ 1.29 \end{bmatrix}$$
$$\mathbf{C}_{Km} = [\ -1.23 \quad .24 \quad 1.33\ ]$$

This was obtained by the following algorithm:

$$\mathbf{A}, \mathbf{B}, \mathbf{C}, \epsilon\ (KalD) \Rightarrow \mathbf{A}_{Km}, \mathbf{B}_{Km}, \mathbf{C}_{Km}$$

As is explained in Appendix B, the Kalman decomposition procedure decomposes the state space into four subspaces, namely:

● c-no	Controllable and unobservable
● c-o	Controllable and observable
● nc-no	Uncontrollable and unobservable and
● nc-o	Uncontrollable and observable

The dimensions of these subspaces in our example are given below:

c-no	c-o	nc-no	nc-o
1	3	2	2

The dimensions as well as the modes belonging to these subspaces may be checked by considering the obtained Jordan form.

The Hessenberg minimal realization $R_{Hm} = \{\mathbf{A}_{Hm}, \mathbf{B}_{Hm}, \mathbf{C}_{Hm}\}$ for our example is:

$$\mathbf{A}_{Hm} = \begin{bmatrix} -1.30 & -.44 & .73 \\ 1.20 & -1.03 & .37 \\ .00 & -.64 & -1.67 \end{bmatrix} \qquad \mathbf{B}_{Hm} = \begin{bmatrix} 1.60 \\ .0 \\ .0 \end{bmatrix}$$

$$\mathbf{C}_{Hm} = [\ 1.88 \quad -.05 \quad -.24\ ]$$

This was obtained by the following algorithm:

$$\mathbf{A},\ \mathbf{B},\ \mathbf{C},\ \varepsilon\ (MIN) \Rightarrow \mathbf{A}_{Hm},\ \mathbf{B}_{Hm},\ \mathbf{C}_{Hm}$$

Of course, all of the above three representationas are controllable and observable and have the same transfer function matrix.

4.2.2 Transfer Function to Markov Parameters

This algorithm calculates the Markov parameters from a transfer function matrix $G(z) = \mathbf{C}(z\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D}$. It is based on the obvious equation:

$$G(z) = \frac{W(z)}{d(z)} = H(z^{-1})$$

which using

$$W(z) = \sum_{i=0}^{n} W_i z^i\ ,\quad d(z) = \sum_{i=0}^{n} d_i z^i\ ,\quad H(z^{-1}) = \sum_{i=0}^{\infty} H_i z^{-i}$$

may be reduced to:

$$\sum_{j=0}^{i} d_{n-j} H_{i-j} = W_{n-i} \quad \text{for} \quad i = [0, n] \quad \text{and}$$

$$\sum_{j=0}^{n} d_{n-j} H_{i-j} = 0 \quad \text{for} \quad i = [n+1, \infty] \tag{4.30}$$

Since, by definition, $d_n = 1$, Eq.(4.30) leads to the following recursive expressions permitting calculation of $\mathbf{H}_j$, $j=[0,M\text{-}1]$, for an arbitrary finite M, given d_i and $\mathbf{W}_i$, $i=[0,n]$:

1. Set $\mathbf{W}_n \Rightarrow \mathbf{H}_0$

2. Set $W_{n-i} - \sum_{j=0}^{i} d_{n-j} H_{i-j} \Rightarrow H_i \quad \text{for} \quad i=[1,n]$

3. Set $-\sum_{j=0}^{n} d_{n-j} H_{i-j} \Rightarrow H_i \quad \text{for} \quad i=[n+1, M-1]$

These recursive expressions are implemented by Algorithm *TFH* having the syntax:

$$\mathbf{d},\ \mathbf{W},\ M\ (TFH) \Rightarrow \mathbf{H},\ h_M$$

Input/Output Arguments:

- d is an $(n+1)$ dimensional row containing the coefficients d_i.
- $\mathbf{W}$ is a $[(pm \times (n+1)]$ matrix in PMF. Rows of $\mathbf{W}$ contain coefficients of the polynomials $w_{ij}(z)$ in $W(z)$.
- M is scalar specifying the number of Markov parameters $\mathbf{H}_i$, $i=[0,M\text{-}1]$, to be calculated.
- $\mathbf{H}$ is a $[pm \times M]$ matrix in PMF. Rows of $\mathbf{H}$ contain the first M coefficients of the polynomials $h_{ij}(z^{-1})$ in $H(z^{-1})$.
- h_M is a scalar equal to $\| \mathbf{H}_{M-1} \|$, where $\mathbf{H}_{M-1}$ is the last Markov parameter calculated.

The reason for calcuating the quantity h_M will be explained later. See the Glossary for the particular matrix norm used. As was mentioned earlier, if all the roots of $d(z)$ are within the unit circle and if the scalar M is sufficiently large so that

$$h_M << 1 \tag{4.31}$$

then, coefficients in the matrix $\tilde{H}(z^{-1})$, representing the truncated polynomial matrix:

$$\tilde{H}(z^{-1}) = \sum_{i=0}^{M-1} H_i z^{-i} = (\,\tilde{h}(z^{-1})\,)$$

with sufficient accuracy satisfy:

$$\tilde{H}(z^{-1}) \approx H(z^{-1}) = G(z)$$

It is worth mentioning that if Eq.(4.31) holds, Eqs.(4.30) may be represented either by:

$$\begin{bmatrix} & & D_0 \\ & D_0 & \\ & & \vdots \\ D_0 & \vdots & D_n \\ & D_n & \\ \vdots & & \\ D_n & & \end{bmatrix}\begin{bmatrix} H_0 \\ \vdots \\ H_{M-1}\end{bmatrix} = \begin{bmatrix}0\\ \vdots\\0\\W_0\\W_1\\ \vdots\\ W_n\end{bmatrix}, \quad D_i = I_p d_i$$

or (4.32)

$$\begin{bmatrix} H_0 & \cdots & H_{M-1}\end{bmatrix}\begin{bmatrix} & & & \tilde{D}_0 & \cdots & \tilde{D}_n \\ & & \tilde{D}_0 & \cdots & \tilde{D}_n & \\ & & \cdot & \cdot & \cdot & \\ & \tilde{D}_0 & \cdots & \tilde{D}_n & & \\ \tilde{D}_0 & \cdots & \tilde{D}_n & & & \end{bmatrix} = \begin{bmatrix} 0 & \cdots & 0 & W_0 & \cdots & W_n\end{bmatrix}$$

$$\text{where} \quad \tilde{D}_i = I_m d_i$$

The forms of Eqs.(4.32) could also be used for calculating H_j, given d_i and W_i.

4.2.3 Continuous-Time Transfer Function Response

This algorithm calculates the zero-state response of a general C-T transfer function matrix $G(s) = W(s)/d(s) = \mathbf{C}(s\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D}$., given the samples of the input signals. The calculations are based on the assumption that the input signals are linearly interpolated between samples. The syntax of the algorithm is:

$$\mathbf{d}, \mathbf{W}, \mathbf{u}, T\ (CDTR) \Rightarrow \mathbf{y}$$

Input/Output Arguments:

- d is an $(n+1)$ dimensional row containing the coefficients d_i, $i=[0,n]$ of $d(s)$.
- **W** is a $[pm \times (n+1)]$ matrix in the PMF. The rows of **W** contain the coefficients w_{ijk} of polynomials $w_{ij}(s)$ in $W(s)$.
- u is an $(m \times N)$ matrix containing N samples of the m-dimensional input vector $\mathbf{u}(t)$.
- T is total simulation time in seconds.

- y is a $(p \times N)$ matrix containing N samples of the C-T system response.

The calculation is performed using a 4th order Runge-Kutta method. Since no prediction or correction is made, for sufficient accuracy the number of samples N should satisfy:

$$N > 5T|\lambda_{max}|$$

where λ_{max} is the maximum root of the denominator $d(s)$. The multi-input case is treated by summing the responses of m single-input multi-output subsystems. It is fair to say that better accuracy in simulating C-T systems is offered by *CDSR* discussed in 4.1.5. which calculates the response of systems defined in state space.

4.2.4 Discrete-Time Transfer Function Response

This algorithm calculates the zero-state response of a general D-T transfer function matrix $G(z) = W(z)/d(z) = \mathbf{C}(z\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D}$, given the samples of the input signals. The syntax of the algorithm is:

$$\mathbf{d}, \mathbf{W}, \mathbf{u}, 0\ (CDTR) \Rightarrow \mathbf{y}$$

Input/Output Arguments:

- **d** is an $(n+1)$ dimensional row containing the coefficients d_i, $i=[0,n]$ of $d(z)$.
- **W** is a $[pm \times (n+1)]$ matrix in the PMF. The rows of **W** contain the coefficients w_{ijk} of polynomials $w_{ij}(z)$ in $W(z)$.
- **u** is an $(m \times N)$ matrix containing N samples of the m-dimensional input vector $\mathbf{u}(t)$.
- **y** is a $(p \times N)$ matrix containing N samples of the D-T system response.

The multi-imput case is treated by summing the responses of m single-input multi-output subsystems. The execution time for the calculation of D-T system responses using this algorithm is slightly longer than that of the Algorithm *CDSR*, Section 4.1.6, although the accuracy in the case of D-T systems is the same.

Note that, formally, the same algorithm is used in simulating C-T and D-T systems. The only difference is in the fourth input argument. If the fourth argument is zero or negative, then the arrays **d** and **W** are interpreted by the algorithm to describe a D-T system transfer function matrix. Otherwise, as is the case in simulating C-T systems, the fourth argument contains the total simulation time interval.

4.2.5 Transfer Function to Left Coprime MFD

This algorithm calculates a left coprime MFD $\{D(z), N(z)\}$ from a transfer function $G(z) = W(z)/d(z)$. It is based on:

$$G(z) = \frac{W(z)}{d(z)} = D^{-1}(z)N(z)$$

which may be rewritten as

$$N(z)\tilde{D}(z) - D(z)W(z) = \mathbf{0}\,, \quad \text{where} \quad \tilde{\mathbf{D}}(z) = \mathbf{I}_m d(z) \tag{4.33}$$

Using

$$D(z) = \sum_{i=0}^{k} D_i z^i\,, \quad N(z) = \sum_{i=0}^{k} N_i z^i\,, \quad d(z) = \sum_{i=0}^{n} d_i z^i\,, \quad W(z) = \sum_{i=0}^{n} W_i z^i$$

Eq.(4.33) may be represented by

$$\left[\, N_0 \cdots N_k \mid D_0 \cdots D_k \,\right] \left[\begin{array}{cccccc} \tilde{D}_0 & \tilde{D}_1 & \cdots & \tilde{D}_n & & \\ & \ddots & \ddots & & \ddots & \\ & & \tilde{D}_0 & \tilde{D}_1 & \cdots & \tilde{D}_n \\ \hline -W_0 & -W_1 & \cdots & -W_n & & \\ & \ddots & \ddots & & \ddots & \\ & & -W_0 & -W_1 & \cdots & -W_n \end{array}\right] = \mathbf{0}$$

which, for short, will be expressed by:

$$[\, \mathbf{N}_r \mid \mathbf{D}_r \,]\mathbf{T}_k = \mathbf{0} \tag{4.34}$$

$\mathbf{N}_r$ and $\mathbf{D}_r$ in Eq.(4.34) are $p \times (k+1)m$ and $p \times (k+1)p$ matrices, respectively, containing N_i and D_i, while $\mathbf{T}_k$ is a $[(k+1)(p+m) \times m(n+k+1)]$ matrix consisting of known submatrices $\tilde{D}_i$ and W_i.

Since in this algorithm the matrix $\mathbf{T}_k$ is given, the unknown matrices $\mathbf{N}_r$ and $\mathbf{D}_r$, defining $D(z)$ and $N(z)$ are to be determined from the Null space of $\mathbf{T}_k^T$. Also, the integer k must be determined. Recall that we are looking for a monic $D(z)$ whose matrix $\mathbf{D}_r$ has the properties discussed in Section 3.4, i.e. it has:

- n columns with non-zero, non-unity elements
- p columns of the identity matrix $\mathbf{I}_p$
- kp-n columns of zeros

Similarly, as is the case in some other algorithms to be discussed later, it may be shown that for a sufficiently large k the row rank of $\mathbf{T}_k$ is given by:

$$\text{rank}\,[\ \mathbf{T}_k\ ] = (k+1)m + n$$

leading to

$$n = \text{rank}\,[\ \mathbf{T}_k\ ] - (k+1)m \tag{4.35}$$

The integer n in Eq.(4.35) is the order of the polynomial given by $\det\{\ D(z)\ \}$, where, of course, $D(z)$ is a monic column-reduced polynomial matrix to be determined. It is worth mentioning that if the given $W(z)$ and $d(z)$ are obtained from $\mathbf{C}(z\mathbf{I}-\mathbf{A})^{-1}\mathbf{B} + \mathbf{D} = W(z)/d(z)$, where $R = \{\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D}\}$ is a minimal state space representation of order n, then:

$$d(z) = \det(z\mathbf{I} - \mathbf{A}) = \det\{\ D(z)\ \}$$

Thus, in this algorithm it is first necessary to build the matrix $\mathbf{T}_k$ and then to determine the smallest integer k satisfying Eq.(4.35). Note that the integer k defines the maximum value of the column degrees $\{\ n_i\ \}$, $k = \max\ \{\ n_i\ \}$, by which $D(z)$ is to be represented. From the structure of the matrix $\mathbf{T}_k$ it may be concluded that since $\mathbf{D}_n = \mathbf{I}_m d_n$, with $d_n = 1$, the first $(k+1)m$ rows in $\mathbf{T}_k$ are linearly independent and that among the remaining $(k+1)p$ rows of $\mathbf{T}_k$ there are only n additional linearly independent rows. Recall that in discussing Algorithm *RoDN*, Section 4.1.7, it was established that a set of column degrees of $D(z)$ is equal to a corresponding set of admissible POI. Thus, it may be concluded here that the total number of sets of n linearly independent rows from the "lower" part of $\mathbf{T}_k$ is equal to the total number of admissible sets of POI, i.e. to the *total number* of "admissible" sets of column degrees by which a monic column reduced $D(z)$ may be represented. Thus, from the structure of the matrix $\mathbf{D}_r$, Eq.(3.104), it follows that any selection of n rows from the above mentioned $(k+1)p$ rows of $\mathbf{T}_k$, made in accordance with the selector vector $\mathbf{v}_{ij}$, generated by an admissible set of POI, would yield n rows which are linearly independent with respect to the first $(k+1)m$ rows in $\mathbf{T}_k$. Let this selection be represented by:

$$\tilde{\mathbf{S}}_{ij}^T \mathbf{T}_k = \mathbf{H}_1 \tag{4.36}$$

where the selector matrix $\tilde{\mathbf{S}}_{ij}$ selects into $\mathbf{H}_1$ all $(k+1)m+n$ linearly independent rows. The matrix $\tilde{\mathbf{S}}_{ij}$ may be interpreted as the selector matrix generated by an auxiliary selector vector $\tilde{\mathbf{v}}_{ij}$ given by:

$$\tilde{\mathbf{v}}_{ij} = [\ \underbrace{1\ \cdots\ 1}_{(k+1)m}\ |\ \underbrace{\mathbf{v}_{ij}}_{(k+1)p}\ ] \tag{4.37}$$

i.e. obtained by concatenating a row vector containing $(k+1)m$ unities and the vector $\mathbf{v}_{li}$ generated by an admissible set of POI, i. e. a corresponding set $\{ n_i \}$. In order to determine the matrix $\mathbf{N}_r$ and the non-zero non-unity columns in $\mathbf{D}_r$, Eq.(4.34), denoted in Section 4.1.7, Eq.(4.17), by $-\mathbf{A}_r$, a selector matrix $\bar{\mathbf{S}}_{ld}$ should be generated using another auxiliary vector $\bar{\mathbf{v}}_{ld}$ defined by:

$$\bar{\mathbf{v}}_{ld} = [\, \underbrace{0 \cdots 0}_{(k+1)m} \mid \underbrace{\mathbf{v}_{ld}}_{(k+1)p} \,] \tag{4.38}$$

Recall that the vectors $\mathbf{v}_{li}$ and $\mathbf{v}_{ld}$ used in Eqs.(4.37) and (4.38) have n and p unities, respectively, while all their other elements are equal to zero. These vectors are generated by the set $\{ n_i \}$. Then, using:

$$\bar{\mathbf{S}}_{ld}^T \mathbf{T}_k = \mathbf{H}_2$$

the desired matrices $\mathbf{N}_r$ and $-\mathbf{A}_r$ may be calculated by solving the following system of linear algebraic equations:

$$[\, \mathbf{N}_r \mid -\mathbf{A}_r \,]\mathbf{H}_1 = -\mathbf{H}_2 \tag{4.39}$$

Having determined matrices $\mathbf{N}_r$ and $-\mathbf{A}_r$, the desired polynomial matrices $D(z)$ and $N(z)$ may be obtained using Eq.(4.20) given in Algorithm *RoDN*, i.e.

$$\mathbf{S}_{ld}^T - \mathbf{A}_r \mathbf{S}_{li}^T \Rightarrow \mathbf{D}_r$$

and, finally, $\mathbf{N}_r$, m *(PMFr)* $\Rightarrow N(z)$ and $\mathbf{D}_r$, p *(PMFr)* $\Rightarrow D(z)$.

The algorithm may be considerably simplified if we know in advance the system order n, i.e. an "admissible" set of column degrees $\{ n_i \}$. Then, it is sufficient to determine k, build the matrix $\mathbf{T}_k$ and proceed with Eq.(4.36). Details are given in the algorithm formulation.

Thus, the following algorithm, permitting calculation of a left coprime MFD $\{ D(z), N(z) \}$ with $D(z)$ monic and column-reduced, given a transfer function matrix $G(z)$, i.e. a $(p \times m)$ polynomial matrix $W(z)$ and a characteristic polynomial $d(z)$, may be suggested.

Syntax: $\mathbf{d}, \mathbf{W}, \varepsilon, \mathbf{n}_d$ *(TFDN)* $\Rightarrow \mathbf{D}, \mathbf{N}, \mathbf{n}$, *C#*

Input/Output Arguments:

- d is an $(n+1)$ dimensional row containing the coefficients of $d(z)$.
- W is a $[pm \times n+1]$ matrix in the PMF. The rows of $\mathbf{W}$ contain the coefficients w_{ijk} of the polynomials $w_{ij}(z)$ in $H(z)$.
- ε is a sufficiently small positive number used in rank calculations.
- $\mathbf{n}_d = \{ n_i \}$ is the set of "desired" column degrees by which $D(z)$

is to be represented. If $\mathbf{n}_d$ is not known, any scalar, e.g. ϵ, may be used as the fourth argument.

- $\mathbf{D}$ is a $[p^2 \times (k+1)]$ matrix in the PMF. The rows of $\mathbf{D}$ contain the coefficients d_{ijk} of the polynomials $d_{ij}(z)$ in $D(z)$.
- $\mathbf{N}$ is a $[pm \times (k+1)]$ matrix in the PMF. The rows of $\mathbf{N}$ contain the coefficients n_{ijk} of the polynomials $n_{ij}(z)$ in $N(z)$.
- $\mathbf{n}$ is the set of column degrees of $D(z)$.
- *C#* is the degree of admissibility of the set of column degrees of $D(z)$.

Algorithm:

```
1.  Set $\mathbf{W}$ (Alt) $\Rightarrow \mathbf{W}_c, \mathbf{W}_r$, $\mathbf{I}_m d_i \Rightarrow \hat{\mathbf{D}}_i$
2.  If $\mathbf{n}_d$ is specified, determine $k$, build $\mathbf{T}_k$, set $\mathbf{n}_d \Rightarrow \{ n_i \}$, and go to
    9; else, go to 3
3.  Set $0 \Rightarrow k$ and $0 \Rightarrow \mathbf{n}_o$
4.  Set $k+1 \Rightarrow k$
5.  Build $\mathbf{T}_k$, in Eq.(4.34), and set rank($\mathbf{T}_k$) - $km \Rightarrow n$
6.  If $n = n_o$ go to 7; else, set $n \Rightarrow n_o$ and go to 4
7.  From $\mathbf{T}_k$ determine the unique column degrees, i.e. $\mathbf{T}_k$ (IND) $\Rightarrow$
    $\{n_i\}$
8.  Define an appropriate set of column degrees $\{n_i\}$
9.  Set $\{n_i\}$ (SMat) $\Rightarrow n_m, \mathbf{S}_w, \mathbf{S}_l, \mathbf{S}_{ll}, \mathbf{S}_{ld}$
10. Using $\mathbf{S}_{ll}, \mathbf{S}_{ld}$ and $k$, define $\hat{\mathbf{S}}_{ll}$ and $\hat{\mathbf{S}}_{ld}$, Eqs.(4.37)-(4.38)
11. Set $\hat{\mathbf{S}}_{ll}^T \mathbf{T}_k \Rightarrow \mathbf{H}_1$ and $\hat{\mathbf{S}}_{ld}^T \mathbf{T}_k \Rightarrow \mathbf{H}_2$
12. Calculate the admissibility degree of $\mathbf{H}_1$, i.e. $\mathbf{H}_1$ (C#) $\Rightarrow$ C#
13. If C# is "too small," go to 8; else, go to 14
14. Solve $\mathbf{X} \mathbf{H}_1 = -\mathbf{H}_2$ for $\mathbf{X}$, where $\mathbf{X} = [\, \mathbf{N}_r \mid -\mathbf{A}_r \,]$
15. Set $\mathbf{S}_{ld}^T - \mathbf{A}_r \mathbf{S}_{ll}^T \Rightarrow \mathbf{D}_r$
16. Set $\mathbf{D}_r, p$ (PMFr) $\Rightarrow \mathbf{D}$ and $\mathbf{N}_r, m$ (PMFr) $\Rightarrow \mathbf{N}$
```

The polynomial matrix "service" algorithm *Alt* used in Step 1 rearranges elements in the PMF $\mathbf{W}$ into the "alternate" forms $\mathbf{W}_c$ and $\mathbf{W}_r$, given by Eq.(4.7). The Algorithm *IND*, in Step 7, determines the unique observability indices, i.e. column degrees of $D(z)$, by detecting the first linearly independent rows in $\mathbf{T}_k$. Note that in this case by detecting the first $(k+1)m + n$ linearly independent rows in $\mathbf{T}_k$, Algorithm *IND* first determines the auxiliary selector vector $\hat{\mathbf{v}}_{ll}$, Eq.(4.37), which is later partitioned into $(k+1)m$ unities and the selector vector $\mathbf{v}_{ll}$ which leads directly to the unique column degrees (or observability indices ν_{ci}). The admissibility degree algorithm (*C#*) in Step 12 defines *C#* as the ratio of the smallest to the largest singular value of $\mathbf{H}_1^T$.

4.2.6 Transfer Function to Right Coprime MFD

Although this algorithm is dual to *TFDN*, it will be briefly stated here. The algorithm is based on:

$$G(z) = \frac{W(z)}{d(z)} = N(z)\,D^{-1}(z)$$

which may be expressed as

$$\tilde{D}(z)N(z) - W(z)D(z) = \mathbf{0}\ , \quad \text{where} \quad \tilde{\mathbf{D}}(z) = \mathbf{I}_p d(z) \tag{4.40}$$

Eq.(4.40) may also be represented by:

$$\left[\begin{array}{ccc|ccc}
\tilde{\mathbf{D}}_0 & & & -\mathbf{W}_0 & & \\
\tilde{\mathbf{D}}_1 & \tilde{\mathbf{D}}_0 & & -\mathbf{W}_1 & -\mathbf{W}_0 & \\
\vdots & \ddots & & \vdots & \ddots & \\
\tilde{\mathbf{D}}_n & & \tilde{\mathbf{D}}_0 & -\mathbf{W}_n & & -\mathbf{W}_0 \\
 & \tilde{\mathbf{D}}_n & \tilde{\mathbf{D}}_1 & & -\mathbf{W}_n & -\mathbf{W}_1 \\
 & \ddots & \vdots & & \ddots & \vdots \\
 & & \tilde{\mathbf{D}}_n & & & -\mathbf{W}_n
\end{array}\right]
\left[\begin{array}{c} N_0 \\ \vdots \\ N_k \\ --- \\ D_0 \\ \vdots \\ D_k \end{array}\right] = \mathbf{0} \tag{4.41}$$

which, for short, will be expressed by:

$$\mathbf{T}_k \left[\begin{array}{c} \mathbf{N}_c \\ --- \\ \mathbf{D}_c \end{array}\right] = \mathbf{0} \tag{4.42}$$

The relationships between the symbols in Eqs.(4.41) and (4.42) should be clear.

Therefore, according to the duality principle, the first linearly independent columns of $\mathbf{T}_k$ give information about the set of row degrees $\{ n_i \}$ which are equal to the unique controllability indices μ_u of the corresponding state space representation. Consequently, by postmultiplying $\mathbf{T}_k$ by a selector matrix $\tilde{\mathbf{S}}_{li}$, corresponding to an auxiliary selector vector $\tilde{\mathbf{v}}_{li}$, defined in the dual sense by Eq.(4.37), a full column rank matrix is obtained, i.e.:

$$\mathbf{T}_k \tilde{\mathbf{S}}_{li} = \mathbf{H}_1$$

Similarly, by postmultiplying $\mathbf{T}_k$ by the selector matrix $\tilde{\mathbf{S}}_{ld}$, dual to Eq.(4.38), m columns are selected from $\mathbf{T}_k$ which are linearly dependent on the columns of $\mathbf{H}_1$. In other words:

$$\mathbf{H}_1 \begin{bmatrix} \mathbf{N}_c \\ ---- \\ -\mathbf{A}_{cc} \end{bmatrix} = -\mathbf{H}_2 \,, \quad \text{where} \quad \mathbf{H}_2 = \mathbf{T}_k \hat{\mathbf{S}}_{ld} \tag{4.43}$$

From Section 4.1.8 it may be concluded that all n rows of the matrix $-\mathbf{A}_{cc}$ in Eq.(4.43) correspond to n non-zero non-unity rows in $\mathbf{D}_c$, and also that the m columns of $\mathbf{A}_{cc}$ correspond to the m non-zero non-unity columns in the system matrix $\mathbf{A}_c$ in a PCF R_c, based on the admissible set of PCI $\mu = \{ \mu_i \}$ which are, in turn, equal to the row degrees $\{ n_i \}$ of the matrix $D(z)$ that we are looking for in this algorithm.

Therefore, having $-\mathbf{A}_{cc}$ from Eq.(4.43), the matrix $\mathbf{D}_c$ may be calculated by:

$$\mathbf{S}_{ld} - \mathbf{S}_{ll} \mathbf{A}_{cc} \Rightarrow \mathbf{D}_c$$

Finally, the desired polynomial matrices $D(z)$ and $N(z)$ may be obtained from $\mathbf{D}_c$ and $\mathbf{N}_c$ using the service algorithm *PMFc*.

Thus, the following algorithm, permitting calculation of a right coprime MFD $\{ N(z), D(z) \}$ with $D(z)$ monic and row-reduced, given a transfer function matrix $G(z)$, i.e. a $(p \times m)$ polynomial matrix $W(z)$ and a characteristic polynomial $d(z)$, may be suggested.

Syntax: $\mathbf{d}, \mathbf{W}, \epsilon, \mathbf{n}_d$ *(TFND)* $\Rightarrow \mathbf{N}, \mathbf{D}, \mathbf{n}, C\#$

Input/Output Arguments:

- $\mathbf{d}$ is an $(n+1)$ dimensional row containing the coefficients of $d(z)$.
- $\mathbf{W}$ is a $[pm \times n+1]$ matrix in the PMF. The rows of $\mathbf{W}$ contain the coefficients w_{ijk} of the polynomials $w_{ij}(z)$ in $H(z)$.
- ϵ is a sufficiently small positive number used in rank calculations
- $\mathbf{n}_d = \{ n_i \}$ is the set of "desired" column degrees by which $D(z)$ is to be represented. If $\mathbf{n}_d$ is not known, any scalar, e.g. ϵ, may be used as the fourth argument.
- $\mathbf{N}$ is a $[pm \times (k+1)]$ matrix in the PMF. The rows of $\mathbf{N}$ contain the coefficients n_{ijk} of the polynomials $n_{ij}(z)$ in $N(z)$.
- $\mathbf{D}$ is a $[m^2 \times (k+1)]$ matrix in the PMF. The rows of $\mathbf{D}$ contain the coefficients d_{ijk} of the polynomials $d_{ij}(z)$ in $D(z)$.
- $\mathbf{n}$ is the set of row degrees of $D(z)$.
- $C\#$ is the degree of admissibility of the set of row degrees of $D(z)$.

Algorithm:

1. Set $\mathbf{W}$ *(All)* $\Rightarrow \mathbf{W}_c, \mathbf{W}_r$, $\mathbf{I}_p d_i \Rightarrow \tilde{\mathbf{D}}_i$
2. If $\mathbf{n}_d$ is specified, determine k, build T_k, set $\mathbf{n}_d \Rightarrow \{ n_i \}$, and go to 9; else, go to 3
3. Set $0 \Rightarrow k$ and $0 \Rightarrow n_o$

```
4.  Set $k+1 \Rightarrow k$
5.  Build $\mathbf{T}_k$, Eq.(4.42), and set rank( $\mathbf{T}_k$ ) - $kp \Rightarrow n$
6.  If $n = n_o$ go to 7; else, set $n \Rightarrow n_o$ and go to 4
7.  From $\mathbf{T}_k$ determine the unique row degrees, i.e. $\mathbf{T}_k$ (IND) $\Rightarrow \{n_i\}$
8.  Define an appropriate set of row degrees $\{n_i\}$
9.  Set $\{n_i\}$ (SMat) $\Rightarrow n_m$, $\mathbf{S}_a$, $\mathbf{S}_l$, $\mathbf{S}_{li}$, $\mathbf{S}_{ld}$
10. Using $\mathbf{S}_{li}$, $\mathbf{S}_{ld}$ and $k$, define $\bar{\mathbf{S}}_{li}$ and $\bar{\mathbf{S}}_{ld}$, Eqs.(4.37)-(4.38)
11. Set $\mathbf{T}_k\bar{\mathbf{S}}_{li} \Rightarrow \mathbf{H}_1$ and $\mathbf{T}_k\bar{\mathbf{S}}_{ld} \Rightarrow \mathbf{H}_2$
12. Calculate the degree of admissibility of $\mathbf{H}_1$, i.e. $\mathbf{H}_1$ (C#) $\Rightarrow$ C#
13. If C# is "too small," go to 8; else, go to 14
14. Solve $\mathbf{H}_1\mathbf{X} = -\mathbf{H}_2$ for $\mathbf{X}$, where $\mathbf{X} = [\ \mathbf{N}_c^T \mid -\mathbf{A}_{oc}^T\ ]^T$
15. Set $\mathbf{S}_{ld} - \mathbf{S}_{li}\mathbf{A}_{oc} \Rightarrow \mathbf{D}_c$
16. Set $\mathbf{D}_c$,m (PMFc) $\Rightarrow$ $\mathbf{D}$ and $\mathbf{N}_c$,p (PMFc) $\Rightarrow$ $\mathbf{N}$
```

The service algorithms *Alt*, *IND* and *C#* were explained in the previous section.

4.2.7 Transfer Function to State Space Forms

In this section the Algorithms *TFRo* and *TFRc* will be formulated. Since they are obtained by slight modifications of Algorithms *TFDN* and *TFND*, respectively, the algorithms will be given directly. The necessary modifications will be discussed following the formal algorithm presentation.

It is worthwhile to compare these algorithms with the "classical" minimal realization procedures, discussed in Section 4.2.1. The main advantage of the *TFRo* and *TFRc* algorithms is that they do *not* require a non-minimal state space representation; but, instead, directly use the given $W(z)$ and $d(z)$, which considerably simplifies the computational aspect. Taking into account the sizes of the matrices involved, it may be concluded that *TFRo* should be used when $m < p$. In this sense, Algorithms *TFRo* and *TFRc* may be considered as "novel" approaches to minimal realization of MIMO systems.

Syntax: $\mathbf{d}, \mathbf{W}, \epsilon, \nu_d$ (*TFRo*) $\Rightarrow R_o = \mathbf{A}_o, \mathbf{B}_o, \mathbf{C}_o, \mathbf{D}_o, \nu$, *C#*

Input/Output Arguments:

- $\mathbf{d}$ is an $(n+1)$ dimensional row containing the coefficients of $d(z)$
- $\mathbf{W}$ is a $[pm \times n+1]$ matrix in the PMF. The rows of $\mathbf{W}$ contain the coefficients w_{ijk} of the polynomials $w_{ij}(z)$ in $H(z)$
- ϵ is a sufficiently small positive number used in rank calculations
- $\nu_d = \{ \nu_i \}$ is an admissible set of POI. If ν_d is not known, any scalar, e.g. ϵ, may be used as the fourth argument
- $R_o = \{\mathbf{A}_o, \mathbf{B}_o, \mathbf{C}_o, \mathbf{D}_o\}$, state space model in a POF

- ν is an admissible set of POI corresponding to R_o
- $C\#$ is the degree of admissibility of the set ν

Algorithm:

```
1.  Set $\mathbf{W}$ $(Alt) \Rightarrow \mathbf{W}_e, \mathbf{W}_r$,  $\mathbf{I}_m d_i \Rightarrow \hat{\mathbf{D}}_i$
2.  If $\mathbf{n}_d$ is specified, determine $k$, build $\mathbf{T}_k$, set $\mathbf{n}_d \Rightarrow \{ n_i \}$, and go to
    9; else, go to 3
3.  Set $0 \Rightarrow k$ and $0 \Rightarrow n_o$
4.  Set $k+1 \Rightarrow k$
5.  Build $\mathbf{T}_k$, Eq.(4.34), and set rank( $\mathbf{T}_k$ ) - $km \Rightarrow n$
6.  If $n = n_o$ go to 7; else, set $n \Rightarrow n_o$ and go to 4
7.  From $\mathbf{T}_k$ find the unique column degrees, i.e. $\mathbf{T}_k$ $(IND) \Rightarrow \{\nu_i\}$
8.  Define an appropriate set of row degrees $\{\nu_i\}$
9.  Set $\{\nu_i\}$ $(SMat) \Rightarrow \nu_m, \mathbf{S}_o, \mathbf{S}_l, \mathbf{S}_{ll}, \mathbf{S}_{ld}$
10. Using $\mathbf{S}_{ll}$, $\mathbf{S}_{ld}$ and $k$, define $\bar{\mathbf{S}}_{ll}$ and $\bar{\mathbf{S}}_{ld}$, Eqs.(4.37)-(4.38)
11. Set $\bar{\mathbf{S}}_{ll}^T \mathbf{T}_k \Rightarrow \mathbf{H}_1$ and $\bar{\mathbf{S}}_{ld}^T \mathbf{T}_k \Rightarrow \mathbf{H}_2$
12. Calculate the degree of admissibility of $\mathbf{H}_1$, i.e. $\mathbf{H}_1$ $(C\#) \Rightarrow C\#$
13. If $C\#$ is "too small," go to 8; else, go to 14
14. Solve  $\mathbf{X}\,\mathbf{H}_1 = -\mathbf{H}_2$ for $\mathbf{X}$, where $\mathbf{X} = [\ \mathbf{N}_r \mid -\mathbf{A}_r\ ]$
15. Partition $\mathbf{I}_n \Rightarrow \begin{bmatrix} \mathbf{C}_o \\ \mathbf{A}_2 \end{bmatrix}$ ($\mathbf{C}_o$ has $p$ rows.)
16. Set $\mathbf{S}_l\mathbf{A}_2 + \mathbf{S}_o\mathbf{A}_r \Rightarrow \mathbf{A}_o$
17. Set $\mathbf{N}_r$, $m$ $(R2C) \Rightarrow \mathbf{N}_c$
18. Set $\mathbf{A}_o$, $\mathbf{S}_o$ $(Qc) \Rightarrow \mathbf{Q}_c$ , $\mathbf{Q}_c$ has $k+1$ blocks $\{\mathbf{A}_o^i\mathbf{S}_o\}$ of $p$ columns
19. Set $\mathbf{Q}_c\mathbf{N}_c \Rightarrow \mathbf{B}_o$
20. Partition $\mathbf{W}_r \Rightarrow [\ \mathbf{Y} \mid \mathbf{D}_o\ ]$, $\mathbf{D}_o$ contains the last $m$ columns of $\mathbf{W}_r$
```

The service algorithm *R2C* in Step 17 rearranges alternate form $\mathbf{N}_r$ into $\mathbf{N}_c$. For details see Eq.(4.7). Matrices $\mathbf{A}_{ro}$ and $\mathbf{N}_{ro}$ used in Step 20 contain the first p and the first m columns from $\mathbf{A}_r$ and $\mathbf{N}_r$, respectively.

Comparing Algorithms *TFDN* and *TFRo*, it may be concluded that the first 14 steps are exactly the same. Only the last 6 steps in *TFRo* differ from the last 2 steps in *TFDN*. This is a consequence of Remark 4.1 given in Section 4.1.7. As may be noted, these 6 last steps in *TFRo* actually calculate the matrices in $R_o = \{\mathbf{A}_o, \mathbf{B}_o, \mathbf{C}_o, \mathbf{D}_o\}$. In the sequel a brief explanation of these steps will be given. Some of these steps may be obvious, while others may be verified by direct (straightforward) matrix calculation. It should be noted that these expressions will be used in several algorithms to be discussed later.

Step 15 defines the matrix $\mathbf{C}_o = [\ \mathbf{I}_p \mid \mathbf{0}\ ]$, consisting of the first p rows of an $(n \times n)$ identity matrix $\mathbf{I}_n$, as well as the auxiliary $[(n\text{-}p) \times n]$ matrix $\mathbf{A}_2$ containing the last n-p rows from $\mathbf{I}_n$. Obviously:

$$\begin{bmatrix} \mathbf{C}_o \\ \mathbf{A}_2 \end{bmatrix} = \mathbf{I}_n \tag{4.44}$$

Step 16 defines $\mathbf{A}_o$ as:

$$\mathbf{A}_o = \mathbf{S}_l \mathbf{A}_2 + \mathbf{S}_a \mathbf{A}_r \,, \quad \text{where} \quad \mathbf{A}_r = -\mathbf{D}_r \mathbf{S}_{ll} \tag{4.45}$$

which may be verified by considering the structure of the auxiliary matrix $\mathbf{A}_2$ and selector matrices $\mathbf{S}_l$, $\mathbf{S}_a$ and $\mathbf{S}_{ll}$ discussed in Section 3.3.4, Eq.(3.79).

Steps 17, 18 and 19 define the matrix $\mathbf{B}_o$ by:

$$\mathbf{B}_o = \left[\, \mathbf{S}_a \;|\; \mathbf{A}_o \mathbf{S}_a \;|\; \cdots \;|\; \mathbf{A}_o^k \mathbf{S}_a \,\right] \begin{bmatrix} \mathbf{N}_0 \\ \mathbf{N}_1 \\ \vdots \\ \mathbf{N}_k \end{bmatrix}, \quad \text{with} \quad k = \max\{\, \nu_i \,\}$$

which is, in fact, a straightforward MIMO generalization of the procedure used in calculating the observability state space form of a SISO system given a transfer function $g(s) = b(s)/a(s)$, described in Procedure 4 in Section 3.2.4.

Finally, the matrix $\mathbf{D}_o$ in Step 20 is calculated by:

$$\mathbf{D}_o = \mathbf{C}_o \mathbf{A}_o^{-1} \mathbf{B}_o - \mathbf{A}_{ro}^{-1} \mathbf{N}_o \tag{4.46}$$

which is a direct consequence of:

$$\mathbf{D}_o = D^{-1}(z)\, N(z) - \mathbf{C}_o (z\mathbf{I} - \mathbf{A}_o)^{-1} \mathbf{B}_o \tag{4.47}$$

and the assumption that the matrices $\mathbf{A}_o$ and $\mathbf{A}_{ro}$, the first p columns from $\mathbf{A}_r$, are nonsingular, which is almost always the case in the case of discrete systems.

If it happens that $\mathbf{A}_o$ is singular, i.e. that the system has at least one pole at the origin, leading to a singular $\mathbf{A}_{ro}$ as well, then $\mathbf{D}_o$ may be calculated by evaluating $\mathbf{D}_o$ using Eq.(4.47) for an arbitrary z not equal to a system pole. This calculation, for matrices $\mathbf{D}_o$ (and $\mathbf{D}_c$), is performed by Algorithms *GeDo* (and *GeDc*); both algorithms have *L-A-S* implementations listed in Appendix C.

Algorithm TFRc:

Syntax: $\mathbf{d}, \mathbf{W}, \epsilon, \mu_d$ *(TFRc)* $\Rightarrow \mathbf{A}_c, \mathbf{B}_c, \mathbf{C}_c, \mathbf{D}_c, \mu,$ *C#*

Input/Output Arguments:

- $\mathbf{d}$ is an $(n+1)$ dimensional row containing the coefficients of $d(z)$
- $\mathbf{W}$ is a $[pm \times n+1]$ matrix in the PMF. The rows of $\mathbf{W}$ contain the coefficients w_{ijk} of the polynomials $w_{ij}(z)$ in $H(z)$
- ϵ is a sufficiently small positive number used in rank calculations
- $\mu_d = \{ \mu_i \}$ is an admissible set of PCI. If μ_d is not known, any scalar, e.g. ϵ, may be used as the fourth argument
- $R_c = \{\mathbf{A}_c, \mathbf{B}_c, \mathbf{C}_c, \mathbf{D}_c\}$, state space model in a PCF
- μ is an admissible set of PCI corresponding to R_c
- *C#* is the degree of admissibility of the set μ

Algorithm:

```
1.  Set $\mathbf{W}$ (Alt) $\Rightarrow \mathbf{W}_c, \mathbf{W}_r$, $\mathbf{I}_p d_l \Rightarrow \hat{\mathbf{D}}_l$
2.  If $\mu_d$ is specified, determine $k$, build $\mathbf{T}_k$, set $\mu_d \Rightarrow \{ \mu_i \}$, and go to
    9; else, go to 3
3.  Set $0 \Rightarrow k$ and $0 \Rightarrow n_o$
4.  Set $k+1 \Rightarrow k$
5.  Build $\mathbf{T}_k$, Eq.(4.42), and set rank( $\mathbf{T}_k$ ) - $kp \Rightarrow n$
6.  If $n = n_o$ go to 7; else, set $n \Rightarrow n_o$ and go to 4
7.  From $\mathbf{T}_k$ determine the unique set of CI, i.e. $\mathbf{T}_k$ (IND) $\Rightarrow \{\mu_i\}$
8.  Define an admissible set of PCI $\{\mu_i\}$
9.  Set $\{\mu_i\}$ (SMat) $\Rightarrow \mu_{ur}, \mathbf{S}_u, \mathbf{S}_l, \mathbf{S}_{li}, \mathbf{S}_{ld}$
10. Using $\mathbf{S}_{li}$, $\mathbf{S}_{ld}$ and $k$, define $\hat{\mathbf{S}}_{li}$ and $\hat{\mathbf{S}}_{ld}$, Eqs.(4.37)-(4.38)
11. Set $\mathbf{T}_k\hat{\mathbf{S}}_{li} \Rightarrow \mathbf{H}_1$ and $\mathbf{T}_k\hat{\mathbf{S}}_{ld} \Rightarrow \mathbf{H}_2$
12. Calculate the degree of admissibility of $\mathbf{H}_1$, i.e. $\mathbf{H}_1$ (C#) $\Rightarrow$ C#
13. If C# is "too small," go to 8; else, go to 14
14. Solve $\mathbf{H}_1\mathbf{X} = -\mathbf{H}_2$ for $\mathbf{X}$, where $\mathbf{X} = [\ \mathbf{N}_c^T \mid -\mathbf{A}_{ur}^T\ ]^T$
15. Partition $\mathbf{I}_n \Rightarrow [\ \mathbf{B}_c \mid \mathbf{A}_2\ ]$ ($\mathbf{B}_c$ has $m$ columns.)
16. Set $\mathbf{A}_2\mathbf{S}_l^T + \mathbf{A}_{ur}\mathbf{S}_u^T \Rightarrow \mathbf{A}_c$
17. Set $\mathbf{N}_c$, $p$ (C2R) $\Rightarrow \mathbf{N}_r$
18. Set $\mathbf{A}_c$, $\mathbf{S}_u^T$ (Qo) $\Rightarrow \mathbf{Q}_o$ ($\mathbf{Q}_o$ has $k+1$ blocks $\{\mathbf{S}_u^T\mathbf{A}_c^i\}$ of $m$ rows.)
19. Set $\mathbf{N}_r\mathbf{Q}_o \Rightarrow \mathbf{C}_c$
20. Partition $\mathbf{W}_r \Rightarrow \begin{bmatrix} \mathbf{Y} \\ \cdots \\ \mathbf{D}_c \end{bmatrix}$, $\mathbf{D}_c$ contains the last $p$ rows of $\mathbf{W}_r$
```

Again, the first 14 steps in Algorithms *TFND* and *TFRc* are exactly the same. Using the principle of duality, the last 6 steps of *TFRc* may be easily verified. For convenience and reference the equations defining the matrices in $R_c = \{\mathbf{A}_c, \mathbf{B}_c, \mathbf{C}_c, \mathbf{D}_c\}$ will be given in the sequel:

$$\mathbf{I}_n = [\,\mathbf{B}_c \mid \mathbf{A}_2\,], \quad \text{or} \quad \mathbf{B}_c = \begin{bmatrix} \mathbf{I}_m \\ 0 \end{bmatrix}, \quad \mathbf{A}_2 = \begin{bmatrix} 0 \\ \mathbf{I}_{n-m} \end{bmatrix}$$

$$\mathbf{A}_c = \mathbf{A}_2 \mathbf{S}_i^T + \mathbf{A}_{cc} \mathbf{S}_a^T, \quad \text{where} \quad \mathbf{A}_{cc} = -\mathbf{S}_{ii}^T \mathbf{D}_c$$

$$\mathbf{C}_c = [\,\mathbf{N}_0 \mid \mathbf{N}_1 \mid \cdots \mathbf{N}_k\,] \begin{bmatrix} \mathbf{S}_a^T \\ \mathbf{S}_a^T \mathbf{A}_c \\ \vdots \\ \mathbf{S}_a^T \mathbf{A}_c^k \end{bmatrix}, \quad k = \max\{\mu_i\} \tag{4.48}$$

$$\mathbf{D}_c = \mathbf{C}_c \mathbf{A}_c^{-1} \mathbf{B}_c - \mathbf{N}_a \mathbf{A}_{cca}^{-1}$$

where now $\mathbf{A}_{cca}$ consists of the first m rows from the $(n \times m)$ matrix $\mathbf{A}_{cc}$. Equations (4.48) are dual to Eqs.(4.44) - (4.47). For more details see Algorithm *RcND*, Remark 4.2 and the included example in Section 4.1.8.

4.3 Conversions from Markov Parameters

Given a set of Markov parameters, there are algorithms for calculating equivalent state space models (observable or controllable forms) as well as equivalent ARMA (MFD) models (left or right coprime forms). If it is desired to convert to a transfer function, it is recommended to use the state space representation or either left or right MFD as an intermediate stage, although there is an algorithm for direct conversion to a transfer function matrix.

There are several procedures for calculating state space models from a given set of Markov parameters. Some of them, known variations of the *Ho-Kalman algorithm*, *eigensystem realization algorithm (ERA)*, etc. are mentioned in the end-of-chapter references. Here, we will describe alternate procedures which take advantage of the flexibility offered by PCI and POI and, consequently, give state space representations in either PCF or POF. In addition, the procedures to be discussed here are less computationally "intensive" and more compatible with other intermodel conversions discussed in this chapter.

4.3.1 Markov Parameters to Observable State Form

This algorithm calculates an observable form state space model $R_o = \{\mathbf{A}_o, \mathbf{B}_o,$

$\mathbf{C}_o$, $\mathbf{D}_o\}$ in a POF, based on an admissible set of POI from a corresponding set of Markov parameters.

To derive the algorithm *HRo*, consider a state space model $R_o = \{\mathbf{A}_o, \mathbf{B}_o, \mathbf{C}_o, \mathbf{D}_o\}$ in a POF, based on an admissible set of POI $\nu = \{\nu_i\}$. Then it is relatively easy to verify that the following relation holds:

$$\mathbf{Q}_o\mathbf{Q}_c = \begin{bmatrix} \mathbf{H}_1 & \mathbf{H}_2 & \cdots & \mathbf{H}_k \\ \mathbf{H}_2 & \mathbf{H}_3 & \cdots & \mathbf{H}_{k+1} \\ \vdots & & \cdots & \vdots \\ \mathbf{H}_k & \mathbf{H}_{k+1} & \cdots & \mathbf{H}_{2k+1} \end{bmatrix} \triangleq \mathbf{H}[k] \tag{4.49}$$

where $\mathbf{Q}_o$ and $\mathbf{Q}_c$ are the observability and controllability matrices of the pairs $\{\mathbf{A}_o, \mathbf{C}_o\}$ and $\{\mathbf{A}_o, \mathbf{B}_o\}$, respectivly, while $\mathbf{H}_i$, $i=[1,2k+1]$ are Markov parameters given by (4.3). The matrix $\mathbf{H}[k]$, Eq.(4.49) is referred to as the *Hankel matrix*. For simplicity of notation, let $k = \nu_{max} = \max\{\nu_i\} = \mu_{max} = \max\{\mu_j\}$. Since Markov parameters do not depend on the particular similarity transformation matrix used, they may also be expressed by:

$$\mathbf{H}_i = \mathbf{C}_o\mathbf{A}_o^{i-1}\mathbf{B}_o$$

Assuming that R_o is minimal, i.e. both controllable and observable, then

$$\text{rank}(\mathbf{Q}_c) = \text{rank}(\mathbf{Q}_o) = n \tag{4.50}$$

leading to:

$$\text{rank}(\mathbf{H}[k]) = n \tag{4.51}$$

From the definition of the unique observability indices ν_o, Sec. 3.3.2, it may be concluded that the first linearly independent rows of $\mathbf{H}[k]$ in Eq.(4.49) determine these indices in exactly the same manner as these indices are determined from the rows of the observability matrix $\mathbf{Q}_o$.

From Eq.(4.49) it may be concluded that:

$$\mathbf{Q}_o\mathbf{A}_o\mathbf{Q}_c = \begin{bmatrix} \mathbf{H}_2 & \mathbf{H}_3 & \cdots & \mathbf{H}_{k+1} \\ \mathbf{H}_3 & \mathbf{H}_4 & \cdots & \mathbf{H}_{k+2} \\ \vdots & & \cdots & \vdots \\ \mathbf{H}_{k+1} & \mathbf{H}_{k+2} & \cdots & \mathbf{H}_{2k+2} \end{bmatrix} \triangleq \tilde{\mathbf{H}}[k] \tag{4.52}$$

Now, recall that in Chapter 3 for R_o in a POF, it was stated that:

- $\mathbf{Q}_o$ has n rows equal to all n rows of the identity matrix $\mathbf{I}_n$. The locations of these rows correspond to the locations of unities in the selector vector $\mathbf{v}_k$.

Thus, using the selector matrix, $\mathbf{S}_k$, defined in Eq.(3.79), it may be concluded that:

$$\mathbf{S}_{ii}^{T}\mathbf{Q}_{o} = \mathbf{I}_{n} \tag{4.53}$$

Thus, premultipying $\mathbf{H}[k]$ and $\tilde{\mathbf{H}}[k]$ in Eqs.(4.49) and (4.52), respectively, with $\mathbf{S}_{ii}^{T}$, and using Eq.(4.53) yields:

$$\begin{gathered}\mathbf{A}_{o}\hat{\mathbf{H}}_{1} = \hat{\mathbf{H}}_{2} \quad \text{and} \quad \mathbf{Q}_{c} = \hat{\mathbf{H}}_{1} \\ \text{where} \quad \hat{\mathbf{H}}_{1} = \mathbf{S}_{ii}^{T}\mathbf{H}[k] \quad \text{and} \quad \hat{\mathbf{H}}_{2} = \mathbf{S}_{ii}^{T}\tilde{\mathbf{H}}[k]\end{gathered} \tag{4.54}$$

which, since $\hat{\mathbf{H}}_{1}$ is a full row rank matrix, may be used for calculating $\mathbf{A}_{o}$. Also, $\mathbf{B}_{c}$ may be obtained by taking the first m columns from $\hat{\mathbf{H}}_{1}$. Finally, it is known that:

$$\mathbf{C}_{o} = [\, \mathbf{I}_{p} \quad \mathbf{0} \,] \quad \text{and} \quad \mathbf{D}_{o} = \mathbf{H}_{0}$$

Thus, from Eqs.(4.49) - (4.54) the following algorithm may be formulated:

Syntax: $\mathbf{H}, \epsilon, \nu_{d}$ *(HRo)* $\Rightarrow \mathbf{A}_{o}, \mathbf{B}_{o}, \mathbf{C}_{o}, \mathbf{D}_{o}, \nu, C\#$

Input/Output Arguments:

- $\mathbf{H}$ is a $(pm \times M)$ matrix in PMF. The rows of $\mathbf{H}$ contain the first M coefficients of the polynomials $h_{ij}(z^{-1})$ in $H(z^{-1})$.
- ϵ is a sufficiently small positive number used in rank calculations.
- $\nu_{d} = \{ \nu_{i} \}$ is an admissible set of POI. If ν_{d} is not known, any scalar, e.g. ϵ, may be used as the third argument.
- $R_{o} = \{\mathbf{A}_{o}, \mathbf{B}_{o}, \mathbf{C}_{o}, \mathbf{D}_{o}\}$, a state space representation in a POF.
- $\nu = \{ \nu_{i} \}$, a set of admissible POI corresponding to R_{o}.
- $C\#$ is the degree of admissibility of the set ν.

Algorithm:

```
1.  Set H (Alt) ⇒ $\mathbf{H}_c, \mathbf{H}_r$
2.  If $\nu_d$ is specified, determine $k$, build $\mathbf{H}[k]$, Eq.(4.49), set $\nu_d = \{\nu_d\}$,
    and go to 9; else, go to 3
3.  Set $0 \Rightarrow k$ and $0 \Rightarrow n_o$
4.  Set $k+1 \Rightarrow k$
5.  Build $\mathbf{H}[k]$, Eq.(4.49), and set rank( $\mathbf{H}[k]$ ) $\Rightarrow n$
6.  If $n = n_o$ go to 6; else, set $n \Rightarrow n_o$ and go to 4
7.  From $\mathbf{H}[k]$ determine the unique observability indices $\nu_u$, i.e. $\mathbf{H}[k]$
    (IND) $\Rightarrow \nu_u$
8.  Define an appropriate admissible set of POI $\nu$
9.  Set $\nu$ (SMat) $\Rightarrow \nu_m, \mathbf{S}_a, \mathbf{S}_i, \mathbf{S}_{ii}, \mathbf{S}_{ai}$
10. Partition $\mathbf{H}[k] \Rightarrow [\, \mathbf{H}[k] \mid \mathbf{X} \,]$, $\mathbf{X}$ has $m$ columns
```

```
11. Parition $\mathbf{H}[k] \Rightarrow \begin{bmatrix} \mathbf{X} \\ \hat{\mathbf{H}}[k] \end{bmatrix}$ and $\mathbf{H}[k] \Rightarrow \begin{bmatrix} \hat{\mathbf{H}}[k] \\ \mathbf{Y} \end{bmatrix}$ ; X and Y have $p$ rows
12. Set $\mathbf{S}_a^T\mathbf{H}[k] \Rightarrow \hat{\mathbf{H}}_1$ and $\mathbf{S}_a^T\hat{\mathbf{H}}[k] \Rightarrow \hat{\mathbf{H}}_2$
13. Calculate the degree of admissibility of $\hat{\mathbf{H}}_1$, i.e. $\hat{\mathbf{H}}_1$ $(C\#) \Rightarrow C\#$
14. If $C\#$ is "too small," go to 8; else, go to 15
15. Solve $\mathbf{A}_o\hat{\mathbf{H}}_1 = \hat{\mathbf{H}}_2$ for $\mathbf{A}_o$
16. Partition $\hat{\mathbf{H}}_1 \Rightarrow [\ \mathbf{B}_o \mid \mathbf{X}\ ]$, $\mathbf{B}_o$ has $m$ columns
17. Set $[\ \mathbf{I}_p \mid \mathbf{0}\ ] \Rightarrow \mathbf{C}_o$
18. Set $\mathbf{H}_0 \Rightarrow \mathbf{D}_o$
```

Note that in Step 7 the loop "counter" k corresponds to $\nu_m + 1$. That is why it was necessary in Steps 10 and 11 to extract $\mathbf{H}[k]$ and $\hat{\mathbf{H}}[k]$ as defined by Eqs.(4.49) and (4.52) for $k = \nu_m$. As was mentioned earlier, the "service" algorithm *Alt* used in Step 1 rearranges elements in $\mathbf{H}$ into the "alternate" forms $\mathbf{H}_c$ and $\mathbf{H}_o$, given by Eq.(4.7). The algorithm *IND*, in Step 7, determines the unique observability indices ν_i of R_o by detecting the first n linearly independent rows in $\mathbf{H}[k]$. The algorithm $C\#$, in Step 13 defines $C\#$ as the ratio of the smallest to the largest singular value of $\hat{\mathbf{H}}_1$.

4.3.2 Markov Parameters to Controllable State Form

This algorithm calculates a controllable form $R_c = \{\mathbf{A}_c, \mathbf{B}_c, \mathbf{C}_c, \mathbf{D}_c\}$ from a corresponding set of Markov parameters. To derive the algorithm *HRc*, consider a state space model $R_c = \{\mathbf{A}_c, \mathbf{B}_c, \mathbf{C}_c, \mathbf{D}_c\}$ in a PCF, based on an admissible set of PCI $\mu = \{\ \mu_i\ \}$. Then, considering Eqs.(4.49) - (4.52), and applying the principle of duality, it may be stated for the controllability matrix $\mathbf{Q}_c$ of the pair $\{\mathbf{A}_c, \mathbf{B}_c\}$ in a PCF that:

- $\mathbf{Q}_c$ has n columns equal to all n columns of the identity matrix $\mathbf{I}_n$. The locations of these columns correspond to the locations of unities in the selector vector $\mathbf{v}_U$ based on μ.
- $\mathbf{Q}_c$ has m columns equal to all m columns of $\mathbf{A}_c$ containing non-zero non-unity elements. Locations of these columns are determined by the selector vector $\mathbf{v}_{\bar{U}}$. The corresponding locations of these "parameter" columns in $\mathbf{A}_c$ are specified by the selector vector $\mathbf{v}_a$.

See Section 3.3.4 for more details in the dual sense. Thus, using the selector matrix, $\mathbf{S}_U$, defined in Eq.(3.79), it may be concluded that:

$$\mathbf{Q}_c\mathbf{S}_U = \mathbf{I}_n \tag{4.55}$$

Thus, postmultipying $\mathbf{H}[k]$ and $\hat{\mathbf{H}}[k]$ in Eqs.(4.49) and (4.52), respectively, with $\mathbf{S}_{li}$, and using Eq.(4.55), we obtain:

$$\begin{gathered} \hat{\mathbf{H}}_1 \mathbf{A}_c = \hat{\mathbf{H}}_2 \quad \text{and} \quad \mathbf{Q}_s = \hat{\mathbf{H}}_1 \\ \text{where} \quad \hat{\mathbf{H}}_1 = \mathbf{H}[k]\,\mathbf{S}_{li} \quad \text{and} \quad \hat{\mathbf{H}}_2 = \hat{\mathbf{H}}[k]\,\mathbf{S}_{li} \end{gathered} \tag{4.56}$$

which, since $\hat{\mathbf{H}}_1$ is a full column rank matrix, may be used for calculating $\mathbf{A}_c$. Also, $\mathbf{C}_c$ may be obtained by taking the first p rows from $\hat{\mathbf{H}}_1$. Finally, it is known that:

$$\mathbf{B}_c = \begin{bmatrix} \mathbf{I}_m \\ \mathbf{0} \end{bmatrix} \quad \text{and} \quad \mathbf{D}_c = \mathbf{H}_0$$

Thus, the following algorithm may be formulated:

Syntax: $\mathbf{H}, \epsilon, \mu_d$ (*HRc*) $\Rightarrow \mathbf{A}_c, \mathbf{B}_c, \mathbf{C}_c, \mathbf{D}_c, \mu, C\#$

Input/Output Arguments:

- $\mathbf{H}$ is a $(pm \times M)$ matrix in PMF. The rows of $\mathbf{H}$ contain the first M coefficients of the polynomials $h_{ij}(z^{-1})$ in $H(z^{-1})$.
- ϵ is a sufficiently small positive number used in rank calculations.
- $\mu_d = \{\ \mu_i\ \}$ is an admissible set of PCI. If μ_d is not known, any scalar, e.g. ϵ, may be used as the third argument.
- $R_c = \{\mathbf{A}_c, \mathbf{B}_c, \mathbf{C}_c, \mathbf{D}_c\}$, a state space representation in a PCF.
- $\mu = \{\ \mu_i\ \}$, a set of admissible PCI corresponding to R_c.
- $C\#$ is the degree of admissibility of the set μ.

Algorithm:

1. Set $\mathbf{H}$ (*Alt*) $\Rightarrow \mathbf{H}_c, \mathbf{H}_r$
2. If μ_d is specified, determine k, build $\mathbf{H}[k]$, Eq.(4.49), set $\mu_d = \{\mu_i\}$, and go to 9; else, go to 3
3. Set $0 \Rightarrow k$ and $0 \Rightarrow n_o$
4. Set $k+1 \Rightarrow k$
5. Build $\mathbf{H}[k]$, Eq.(4.49), and set rank($\mathbf{H}[k]$) $\Rightarrow n$
6. If $n = n_o$ go to 6; else, set $n \Rightarrow n_o$ and go to 4
7. From $\mathbf{H}[k]$ determine the unique controllability indices μ_u, i.e. $\mathbf{H}[k]$ (*IND*) $\Rightarrow \mu_u$
8. Define an appropriate admissible set of PCI μ
9. Set μ (*SMat*) $\Rightarrow \mu_m, \mathbf{S}_a, \mathbf{S}_i, \mathbf{S}_{li}, \mathbf{S}_{ld}$
10. Partition $\mathbf{H}[k] \Rightarrow [\ \mathbf{H}[k]\ |\ \mathbf{X}\ ]$, $\mathbf{X}$ has m columns

```
11.  Partition $\mathbf{H}[k] \Rightarrow \begin{bmatrix} \mathbf{X} \\ \hat{\mathbf{H}}[k] \end{bmatrix}$ and $\mathbf{H}[k] \Rightarrow \begin{bmatrix} \hat{\mathbf{H}}[k] \\ \mathbf{Y} \end{bmatrix}$ ; $\mathbf{X}$ and $\mathbf{Y}$ have $p$
     rows
12.  Set $\mathbf{H}[k]\mathbf{S}_k \Rightarrow \hat{\mathbf{H}}_1$ and $\hat{\mathbf{H}}[k]\mathbf{S}_k \Rightarrow \hat{\mathbf{H}}_2$
13.  Calculate the degree of admissibility of $\hat{\mathbf{H}}_1$, i.e. $\hat{\mathbf{H}}_1$ $(C\#) \Rightarrow C\#$
14.  If $C\#$ is "too small," go to 8; else, go to 15
15.  Solve $\hat{\mathbf{H}}_1\mathbf{A}_c = \hat{\mathbf{H}}_2$ for $\mathbf{A}_c$
16.  Partition $\mathbf{H}_1 \Rightarrow \begin{bmatrix} \mathbf{C}_c \\ \mathbf{X} \end{bmatrix}$, $\mathbf{C}_c$ has $p$ rows
17.  Set $\begin{bmatrix} \mathbf{I}_m \\ \mathbf{0} \end{bmatrix} \Rightarrow \mathbf{B}_c$
18.  Set $\mathbf{H}_0 \Rightarrow \mathbf{D}_c$
```

For more details see Algorithm *HRo* in Section 4.3.1.

4.3.3 Markov Parameters to Left Coprime MFD

This algorithm calculates a left coprime column-reduced ARMA (MFD) model, $D(z)^{-1}N(z)$, from a corresponding set of Markov parameters. Of course, this algorithm can equally well be applied to a C-T state model to obtain the corresponding MFD representation. As in some previously discussed algorithms, this algorithm is based on:

$$G(z) = H(z^{-1}) = D^{-1}(z)N(z)$$

which using

$$D(z) = \sum_{i=0}^{k} \mathbf{D}_i z^i, \quad N(z) = \sum_{i=0}^{k} \mathbf{N}_i z^i, \quad H(z^{-1}) = \sum_{i=0}^{\infty} \mathbf{H}_i z^{-i} \tag{4.57}$$

may be reduced to:

$$\begin{gathered} \sum_{j=0}^{i} \mathbf{D}_{k-j}\mathbf{H}_{i-j} = \mathbf{N}_{k-i} \quad \text{for} \quad i = [0,k] \quad \text{and} \\ \sum_{j=0}^{k} \mathbf{D}_{k-j}\mathbf{H}_{i-j} = \mathbf{0} \quad \text{for} \quad i = [k+1,\infty] \end{gathered} \tag{4.58}$$

The important differences between Eqs.(4.58) and (4.30), see Section 4.2.2, are:

- $\mathbf{D}_k$ in Eq.(4.58) is not necessarily a full rank matrix, while $\mathbf{I}_p d_n$ in Eq.(4.32) is.
- The integer k in Eq.(4.58) is less than the system order n used in Eq.(4.32).

To be specific, the value of $k = \max\{ n_i \}$, n_i being the column degrees of $D(z)$, satisfies:

$$k \leq n-p+1 \tag{4.59}$$

Equation (4.58) may also be represented by:

$$[\mathbf{D}_0 \;\; \mathbf{D}_1 \;\cdots\; \mathbf{D}_k] \begin{bmatrix} \mathbf{H}_{k+1} & \mathbf{H}_k & \cdots & \mathbf{H}_1 \\ \mathbf{H}_{k+2} & \mathbf{H}_{k+1} & \cdots & \mathbf{H}_2 \\ \cdots & & \ddots & \\ \mathbf{H}_{2k} & \mathbf{H}_{2k-1} & \cdots & \mathbf{H}_k \\ \mathbf{H}_{2k+1} & \mathbf{H}_{2k} & \cdots & \mathbf{H}_{k+1} \end{bmatrix} = \mathbf{0}$$

and

$$[\mathbf{D}_0 \;\; \mathbf{D}_1 \;\cdots\; \mathbf{D}_k] \begin{bmatrix} \mathbf{H}_0 & & & \\ \mathbf{H}_1 & \mathbf{H}_0 & & \\ & \cdots & \ddots & \\ \mathbf{H}_k & \cdots & \mathbf{H}_1 & \mathbf{H}_0 \end{bmatrix} = [\mathbf{N}_0 \;\; \mathbf{N}_1 \;\cdots\; \mathbf{N}_k] \tag{4.60}$$

which, for short, will be expressed by:

$$\mathbf{D}_r \mathbf{T}_k = \mathbf{0} \quad \text{and} \quad \mathbf{D}_r \mathbf{R}_k = \mathbf{N}_r \tag{4.61}$$

respectively. The relationships between symbols in Eqs.(4.60) and (4.61) should be clear.

It is interesting to note that the matrix $\mathbf{T}_k$ in Eqs.(4.60) and (4.61) contains the same Markov parameters $\mathbf{H}_i$, $i = [1,2k+1]$, as the matrix $\mathbf{H}[k]$ in Eq.(4.49), only arranged differently. Note that postmultiplying $\mathbf{T}_k$ by the $[(k+1)m \times (k+1)m]$ permutation matrix $\mathbf{P}_m$, given by:

$$\mathbf{P}_m = \begin{bmatrix} & & & \mathbf{I}_m \\ & & \cdot & \\ & \mathbf{I}_m & & \\ \cdot & & & \\ \mathbf{I}_m & & & \end{bmatrix} \tag{4.62}$$

gives $\mathbf{T}_k\mathbf{P}_m = \mathbf{H}[k]$. Therefore, as was stated in Section 4.3.1, the first linearly independent rows of $\mathbf{T}_k$ give the information about the unique observability indices ν_m of the corresponding state space representation. Consequently, by premultiplying $\mathbf{T}_k$ by a selector matrix $\mathbf{S}_{li}^T$, corresponding to a set of admissible POI, a full row rank matrix is obtained, i. e.:

$$\mathbf{S}_{li}^T\mathbf{T}_k = \mathbf{T}_1$$

Similarly, by premultiplying $\mathbf{T}_k$ by the selector matrix $\mathbf{S}_{ld}^T$, p rows are selected from $\mathbf{T}_k$ which are linearly dependent on the rows of $\mathbf{T}_1$. In other words, Eq.(4.61) yields:

$$-\mathbf{A}_r\mathbf{T}_1 + \mathbf{T}_2 = \mathbf{0}$$

leading to:

$$\mathbf{A}_r\mathbf{T}_1 = \mathbf{T}_2\ , \quad \text{where} \quad \mathbf{T}_2 = \mathbf{S}_{ld}^T\mathbf{T}_k \tag{4.63}$$

The $[p \times n]$ matrix $\mathbf{A}_r$ defines the above mentioned linear dependence.

On the other hand, note that we are looking for a monic column-reduced matrix $D(z)$, whose structure was exemplified in Section 3.4, Eqs.(3.100) - (3.105). In discussing this generic example it was stated that the matrix $\mathbf{D}_r$, appearing in Eqs.(4.60) and (4.61), contains:

- n columns with non-zero and non-unity elements
- p columns of the identity matrix $\mathbf{I}_p$
- kp-n columns of zeros

Combining all what was stated above and with the help of Remark 4.1 and Eqs.(4.17) - (4.20), Section 4.1.7, derived discussing Algorithm *RoDN*, it is not difficult to conclude that all n columns of the matrix $\mathbf{A}_r$ in Eq.(4.17) correspond to n non-zero non-unity columns in $\mathbf{D}_r$ multiplied by -1, and also that the p rows of $\mathbf{A}_r$ correspond to p non-zero non-unity rows in the matrix $\mathbf{A}_o$ in a POF, based on the admissible set of POI $\nu = \{\ \nu_i\ \}$ which is equal to the set of column degrees $\{\ n_i\ \}$ of the desired matrix $D(z)$.

Having $\mathbf{A}_r$ from Eq.(4.63), matrices $\mathbf{D}_r$ and $\mathbf{N}_r$ may be calculated by:

$$\mathbf{S}_{ld}^T - \mathbf{A}_r\mathbf{S}_{li}^T = \mathbf{D}_r$$
$$\mathbf{D}_r\mathbf{R}_k = \mathbf{N}_r$$

Finally, desired polynomial matrices $D(z)$ and $N(z)$ may be obtained from $\mathbf{D}_r$ and $\mathbf{N}_r$ using the service algorithm *PMFr*.

Thus, the following algorithm, permitting calculation of a left coprime MFD $\{\ D(z), N(z)\ \}$ with $D(z)$ monic and column reduced, given the Markov parameters $\mathbf{H}_i$, $i = [0, 2k+1]$, is suggested.

The total number of Markov parameters required is equal to $2k+2$, with:

$$k = \max\{ n_i \}$$

where $\{ n_i \}$ is a selected set of column degrees with which a desired $D(z)$ is to be represented.

Syntax: $\mathbf{H}, \epsilon, \mathbf{n}_d$ (*HDN*) $\Rightarrow$ $\mathbf{D}, \mathbf{N}, \mathbf{n}$, *C#*

Input/Output Arguments:

- $\mathbf{H}$ is a $(pm \times M)$ matrix in PMF. The rows of $\mathbf{H}$ contain the first M coefficients of the polynomials $h_{ij}(z^{-1})$ in $H(z^{-1})$.
- ϵ is a sufficiently small positive number used in rank calculations.
- $\mathbf{n}_d = \{ n_i \}$ is an admissible set of column degrees. If $\mathbf{n}_d$ is not known, any scalar, e.g. ϵ, may be used as the third argument.
- $\mathbf{D}$ is a $[p^2 \times (k+1)]$ matrix in the PMF. The rows of $\mathbf{D}$ contain the coefficients d_{ijk} of the polynomials $d_{ij}(z)$ in $D(z)$.
- $\mathbf{N}$ is a $[pm \times (k+1)]$ matrix in the PMF. The rows of $\mathbf{N}$ contain the coefficients n_{ijk} of the polynomials $n_{ij}(z)$ in $N(z)$.
- $\mathbf{n}$ is the set of column degrees of $D(z)$.
- *C#* is the degree of admissibility of the set of column degrees.

Algorithm:

1. Set $\mathbf{H}$ (*Alt*) $\Rightarrow$ $\mathbf{H}_o, \mathbf{H}_r$
2. If $\mathbf{n}_d$ is specified, determine k, build $\mathbf{T}_k$, Eq.(4.61), set $\mathbf{n}_d = \{n_i\}$, and go to 9; else, go to 3
3. Set $0 \Rightarrow k$ and $0 \Rightarrow n_o$
4. Set $k+1 \Rightarrow k$
5. Build $\mathbf{T}_k$, Eq.(4.61), and set rank($\mathbf{T}_k$) $\Rightarrow n$
6. If $n = n_o$ go to 7; else, set $n \Rightarrow n_o$ and go to 4
7. From $\mathbf{T}_k$ determine the unique column degrees, i.e. $\mathbf{T}_k$ (*IND*) $\Rightarrow$ $\{n_i\}$
8. Define a desired set of column degrees $\{n_i\}$
9. Set $\{n_i\}$ (*SMat*) $\Rightarrow$ $\mathbf{n}_{sr}, \mathbf{S}_o, \mathbf{S}_l, \mathbf{S}_{lt}, \mathbf{S}_M$
10. Set $\mathbf{S}_{lt}^T\mathbf{T}_k \Rightarrow \mathbf{T}_1$ and $\mathbf{S}_M^T\mathbf{T}_k \Rightarrow \mathbf{T}_2$
11. Calculate the degree of admissibility of $\mathbf{T}_1$, i.e. $\mathbf{T}_1$ (*C#*) $\Rightarrow$ *C#*
12. If *C#* is "too small," go to 8; else, go to 13
13. Solve $\mathbf{A}_r\mathbf{T}_1 = \mathbf{T}_2$ for $\mathbf{A}_r$
14. Set $\mathbf{S}_M^T - \mathbf{A}_r\mathbf{S}_{lt}^T \Rightarrow \mathbf{D}_r$
15. Set $\mathbf{D}_r\mathbf{R}_k \Rightarrow \mathbf{N}_r$
16. Set $\mathbf{D}_r$ (*PMFr*) $\Rightarrow$ $\mathbf{D}$ and $\mathbf{N}_r$ (*PMFr*) $\Rightarrow$ $\mathbf{N}$

The service algorithms *Alt*, *IND* and *C#* are explained under Algorithm *TFDN*, Section 4.2.5.

4.3.4 Markov Parameters to Right Coprime MFD

This algorithm calculates a right coprime row-reduced ARMA (MFD) model, $N(z)D^{-1}(z)$, from a corresponding set of Markov parameters. As with the previous case, this algorithm can be applied to a C-T state model to obtain the corresponding MFD representation. Although this algorithm is dual to *HDN*, it will be briefly stated here. The algorithm is based on:

$$G(z) = H(z^{-1}) = N(z)D^{-1}(z) \tag{4.64}$$

which using Eq.(4.57) may be reduces to

$$\begin{aligned} \sum_{j=0}^{i} \mathbf{H}_{i-j}\mathbf{D}_{k-j} &= \mathbf{N}_{k-i} \quad \text{for} \quad i=[0,k] \quad \text{and} \\ \sum_{j=0}^{k} \mathbf{H}_{i-j}\mathbf{D}_{k-j} &= \mathbf{0} \quad \text{for} \quad i=[k+1,\infty] \end{aligned} \tag{4.65}$$

The comments stated after Eq.(4.58) apply here in the dual sense. Equation (4.65) may also be represented by:

$$\begin{bmatrix} \mathbf{H}_{k+1} & \mathbf{H}_{k+2} & \cdots & \mathbf{H}_{2k+1} \\ \mathbf{H}_{k} & \mathbf{H}_{k+1} & \cdots & \mathbf{H}_{2k} \\ \cdots & \ddots & & \\ \mathbf{H}_{2} & \mathbf{H}_{3} & \cdots & \mathbf{H}_{k+2} \\ \mathbf{H}_{1} & \mathbf{H}_{2} & \cdots & \mathbf{H}_{k+1} \end{bmatrix} \begin{bmatrix} \mathbf{D}_0 \\ \\ \vdots \\ \\ \mathbf{D}_k \end{bmatrix} = \mathbf{0}$$

and (4.66)

$$\begin{bmatrix} \mathbf{H}_0 & \mathbf{H}_1 & \cdots & \mathbf{H}_k \\ & \mathbf{H}_0 & & \vdots \\ & & \ddots & \mathbf{H}_1 \\ & & & \mathbf{H}_0 \end{bmatrix} \begin{bmatrix} \mathbf{D}_0 \\ \mathbf{D}_1 \\ \vdots \\ \mathbf{D}_k \end{bmatrix} = \begin{bmatrix} \mathbf{N}_0 \\ \mathbf{N}_1 \\ \vdots \\ \mathbf{N}_k \end{bmatrix}$$

which, for short, will be expressed by:

$$\mathbf{T}_k\mathbf{D}_c = \mathbf{0} \quad \text{and} \quad \mathbf{R}_k\mathbf{D}_c = \mathbf{N}_c \tag{4.67}$$

respectively. The relationships between symbols in Eqs.(4.66) and (4.67) should be clear.

It is interesting to note that the matrix $\mathbf{T}_k$ in Eqs.(4.66) and (4.67) contains the same Markov parameters $\mathbf{H}_i$, $i = [1,2k+1]$, as the matrix $\mathbf{H}[k]$ in Eq.(4.49), only arranged differently. Note that premultiplying $\mathbf{T}_k$ by the $[(k+1)p \times (k+1)p]$ permutation matrix $\mathbf{P}_p$, given by:

$$\mathbf{P}_p = \begin{bmatrix} & & & \mathbf{I}_p \\ & & \cdot^{\cdot^{\cdot}} & \\ & \mathbf{I}_p & & \\ \cdot & & & \\ \mathbf{I}_p & & & \end{bmatrix} \tag{4.68}$$

gives $\mathbf{P}_p\mathbf{T}_k = \mathbf{H}[k]$. Therefore, according to duality, the first linearly independent columns of $\mathbf{T}_k$ give the information about the unique controllability indices μ_u of the corresponding state space representation. Consequently, by postmultiplying $\mathbf{T}_k$ by a selector matrix $\mathbf{S}_{li}$, corresponding to a set of admissible PCI, a full column rank matrix is obtained, i.e.:

$$\mathbf{T}_k \mathbf{S}_{li} = \mathbf{T}_1$$

Similarly, by postmultiplying $\mathbf{T}_k$ by the selector matrix $\mathbf{S}_{ld}$, m rows are selected from $\mathbf{T}_k$ which are linearly dependent on the columns of $\mathbf{T}_1$. In other words:

$$\mathbf{T}_1 \mathbf{A}_{cc} = \mathbf{T}_2 \,, \quad \text{where} \quad \mathbf{T}_2 = \mathbf{T}_k \mathbf{S}_{ld} \tag{4.69}$$

The $[n \times m]$ matrix $\mathbf{A}_{cc}$ defines the above mentioned linear dependence.

On the other hand, note that we are looking for a monic row-reduced matrix $D(z)$, whose structure in the dual sense was exemplified in Section 3.4, Eqs.(3.100) to (3.105). Thus, the $[(k+1)m \times m]$ matrix $\mathbf{D}_c$, appearing in Eqs.(4.66) and (4.67), contains:

- n rows with non-zero and non-unity elements
- m rows of the identity matrix $\mathbf{I}_m$
- km-n rows of zeros

Thus, it may be concluded that all n rows of the matrix $\mathbf{A}_{cc}$ in Eq.(4.69) correspond to n non-zero non-unity rows in $\mathbf{D}_c$ multiplied by -1, and also that the m columns of $\mathbf{A}_{cc}$ correspond to m non-zero non-unity rows in the matrix $\mathbf{A}_c$ in a PCF, based on the admissible set of PCI $\mu = \{\ \mu_i\ \}$ which is equal to the set of row degrees $\{n_i\}$ of the desired matrix $D(z)$.

Having $\mathbf{A}_{cc}$ from Eq.(4.69), matrices $\mathbf{D}_c$ and $\mathbf{N}_c$ may be calculated by:

$$\mathbf{S}_{ld} - \mathbf{S}_{li}\mathbf{A}_{cc} \rightarrow \mathbf{D}_c$$

$$\mathbf{R}_k \mathbf{D}_c \rightarrow \mathbf{N}_c$$

Finally, desired polynomial matrices $D(z)$ and $N(z)$ may be obtained from $\mathbf{D}_c$ and

$\mathbf{N}_c$ using the service algorithm *PMFc*. Thus, the following algorithm, permitting calculation of a right coprime MFD $\{ N(z), D(z) \}$ with $D(z)$ monic and row reduced, given the Markov parameters $\mathbf{H}_i$, $i = [0, 2k+1]$, $k = \max\{ n_i \}$, is suggested.

Syntax: $\mathbf{H}, \epsilon, \mathbf{n}_d$ *(HND)* $\Rightarrow \mathbf{N}, \mathbf{D}, \mathbf{n}, C\#$

Input/Output Arguments:

- $\mathbf{H}$ is a $(pm \times M)$ matrix in PMF. The rows of $\mathbf{H}$ contain the first M coefficients of the polynomials $h_{ij}(z^{-1})$ in $H(z^{-1})$.
- ϵ is a sufficiently small positive number used in rank calculations.
- $\mathbf{n}_d = \{ n_i \}$ is an admissible set of row degrees. If $\mathbf{n}_d$ is not known, any scalar, e.g. ϵ, may be used as the third argument.
- $\mathbf{N}$ is a $[pm \times (k+1)]$ matrix in the PMF. The rows of $\mathbf{N}$ contain the coefficients n_{ijk} of the polynomials $n_{ij}(z)$ in $N(z)$.
- $\mathbf{D}$ is a $[m^2 \times (k+1)]$ matrix in the PMF. The rows of $\mathbf{D}$ contain the coefficients d_{ijk} of the polynomials $d_{ij}(z)$ in $D(z)$.
- $\mathbf{n}$ is the set of row degrees of $D(z)$.
- $C\#$ is the degree of admissibility of the set of row degrees.

Algorithm:

```
1.  Set H (Alt) ⇒ $\mathbf{H}_c$, $\mathbf{H}_r$
2.  If $\mathbf{n}_d$ is specified, determine $k$, build $\mathbf{T}_k$, Eq.(4.67), set $\mathbf{n}_d = \{n_i\}$,
    and go to 9; else, go to 3
3.  Set $0 \Rightarrow k$ and $0 \Rightarrow n_o$
4.  Set $k+1 \Rightarrow k$
5.  Build $\mathbf{T}_k$, Eq.(4.67), and set rank( $\mathbf{T}_k$ ) $\Rightarrow n$
6.  If $n = n_o$ go to 7; else, set $n \Rightarrow n_o$ and go to 4
7.  From $\mathbf{T}_k$ determine the unique row degrees, i.e. $\mathbf{T}_k$ (IND) $\Rightarrow \{n_i\}$
8.  Define a desired set of row degrees $\{n_i\}$
9.  Set $\{n_i\}$ (SMat) $\Rightarrow n_m$, $\mathbf{S}_a$, $\mathbf{S}_i$, $\mathbf{S}_b$, $\mathbf{S}_{bl}$
10. Set $\mathbf{T}_k\mathbf{S}_a \Rightarrow \mathbf{T}_1$ and $\mathbf{T}_k\mathbf{S}_{bl} \Rightarrow \mathbf{T}_2$
11. Calculate the degree of admissibility of $\mathbf{T}_1$, i.e. $\mathbf{T}_1$ (C#) $\Rightarrow$ C#
12. If C# is "too small," go to 8; else, go to 13
13. Solve $\mathbf{T}_1\mathbf{A}_c = \mathbf{T}_2$ for $\mathbf{A}_c$
14. Set $\mathbf{S}_{bl} - \mathbf{S}_b\mathbf{A}_c \Rightarrow \mathbf{D}_c$
15. Set $\mathbf{R}_k\mathbf{D}_c \Rightarrow \mathbf{N}_c$
16. Set $\mathbf{D}_c$, $m$ (PMFc) $\Rightarrow$ D and $\mathbf{N}_c$, $p$ (PMFc) $\Rightarrow$ N
```

The service algorithms *Alt*, *IND* and *C#* were explained earlier under Algorithm *TFDN* in Section 4.2.5.

4.3.5 Markov Parameters to Transfer Function

In this section we present a method for calculating the transfer function matrix $G(z) = W(z)/d(z)$ from given Markov parameters $\mathbf{H}_i$. Since this algorithm is the "inverse" of *TFH*, Section 4.2.2, it is based on the same expressions Eq.(4.30), i.e.:

$$G(z) = \frac{W(z)}{d(z)} = H(z^{-1})$$

which using

$$W(z) = \sum_{i=0}^{n} W_i z^i \ , \quad d(z) = \sum_{i=0}^{n} d_i z^i \ , \quad H(z^{-1}) = \sum_{i=0}^{\infty} H_i z^{-i}$$

may be reduced to:

$$\sum_{j=0}^{i} d_{n-j} H_{i-j} = W_{n-i} \quad \text{for} \quad i=[0,n]$$

and (4.70)

$$\sum_{j=0}^{n} d_{n-j} H_{i-j} = 0 \quad \text{for} \quad i=[n+1,\infty]$$

Since in this algorithm the Markov parameters $\mathbf{H}_j$, $j=[0,M\text{-}1]$, are assumed known, while $\mathbf{W}_i$ and d_i, $i=[1,n]$, are to be determined, Eq.(4.70), i.e. Eq.(4.30), will now be represented differently, either by:

$$\begin{bmatrix} \mathbf{D}_0 & \mathbf{D}_1 & \cdots & \mathbf{D}_n \end{bmatrix} \begin{bmatrix} \mathbf{H}_{n+1} & \mathbf{H}_n & \cdots & \mathbf{H}_1 \\ \mathbf{H}_{n+2} & \mathbf{H}_{n+1} & \cdots & \mathbf{H}_2 \\ \cdots & & \ddots & \\ \mathbf{H}_{2n} & \mathbf{H}_{2n-1} & \cdots & \mathbf{H}_n \\ \mathbf{H}_{2n+1} & \mathbf{H}_{2n} & \cdots & \mathbf{H}_{n+1} \end{bmatrix} = \mathbf{0}$$

and (4.71a)

$$\begin{bmatrix} \mathbf{D}_0 & \mathbf{D}_1 & \cdots & \mathbf{D}_n \end{bmatrix} \begin{bmatrix} \mathbf{H}_0 & & & \\ \mathbf{H}_1 & \mathbf{H}_0 & & \\ & \cdots & \ddots & \\ \mathbf{H}_n & \cdots & \mathbf{H}_1 & \mathbf{H}_0 \end{bmatrix} = \begin{bmatrix} \mathbf{W}_0 & \mathbf{W}_1 & \cdots & \mathbf{W}_n \end{bmatrix}$$

where $\mathbf{D}_i = \mathbf{I}_p \, d_i$, or

$$\begin{bmatrix} \mathbf{H}_{n+1} & \mathbf{H}_{n+2} & \cdots & \mathbf{H}_{2n+1} \\ \mathbf{H}_n & \mathbf{H}_{n+1} & \cdots & \mathbf{H}_{2n} \\ \cdots & \ddots & & \\ \mathbf{H}_2 & \mathbf{H}_3 & \cdots & \mathbf{H}_{n+2} \\ \mathbf{H}_1 & \mathbf{H}_2 & \cdots & \mathbf{H}_{n+1} \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{D}}_0 \\ \\ \vdots \\ \\ \tilde{\mathbf{D}}_n \end{bmatrix} = \mathbf{0}$$

and (4.71b)

$$\begin{bmatrix} \mathbf{H}_0 & \mathbf{H}_1 & \cdots & \mathbf{H}_n \\ & \mathbf{H}_0 & & \vdots \\ & & \ddots & \mathbf{H}_1 \\ & & & \mathbf{H}_0 \end{bmatrix} \begin{bmatrix} \tilde{\mathbf{D}}_0 \\ \tilde{\mathbf{D}}_1 \\ \vdots \\ \tilde{\mathbf{D}}_n \end{bmatrix} = \begin{bmatrix} \mathbf{W}_0 \\ \mathbf{W}_1 \\ \vdots \\ \mathbf{W}_n \end{bmatrix}$$

where, now, $\tilde{\mathbf{D}}_i = \mathbf{I}_m\, d_i$. Eqs.(4.71), for short, will be expressed by:

$$\mathbf{D}_r \mathbf{T} = \mathbf{0} \quad \text{and} \quad \mathbf{D}_r \mathbf{R} = \mathbf{W}_r \tag{4.72a}$$

$$\mathbf{T}\,\mathbf{D}_c = \mathbf{0} \quad \text{and} \quad \mathbf{R}\,\mathbf{D}_c = \mathbf{W}_c \tag{4.72b}$$

respectively, where the notation should be clear.

It is worthwhile to compare Eqs.(4.71b) and (4.66) in Algorithm *HND*, as well as Eqs.(4.71a) with Eq.(4.60) in *HDN*. Recall that in Eq.(4.71) n is used instead of k, where $k < n$, and that $\mathbf{D}_n$, or $\tilde{\mathbf{D}}_n$, in Eq.(4.71) is, by definition, a full rank matrix, while $\mathbf{D}_k$ in both *HDN* and *HND* might be non-singular. Also, the non-zero, non-unity rows in $\mathbf{D}_c$, or the non-zero, non-unity columns in $\mathbf{D}_r$ in *HND* or *HDN*, respectively, are determined by treating full column (or row) rank matrices, while here, as will be seen, this is *not* the case. Thus, the present problem requires a different approach in calculating $d(z)$ and $W(z)$ satisfying Eq.(4.70).

It may be recognized that Eqs.(4.71) and (4.72) are based on

$$D(z)\,H(z^{-1}) = W(z) \quad \text{where} \quad D(z) = \mathbf{I}_p\, d(z) \tag{4.73a}$$

$$H(z^{-1})\,\tilde{D}(z) = W(z) \quad \text{where} \quad \tilde{D}(z) = \mathbf{I}_m\, d(z) \tag{4.73b}$$

respectively, and that the first version is more convenient if $p < m$, and vice versa. Because of that fact, we should have both versions available. In fact, Algorithm *HTF* executes either *HTFp* (first version) or *HTFm* (second version), depending on whether $p < m$ or $m < p$, respectively. Since there is complete duality between these two versions, only *HTFm* will be discussed in the sequel.

The basic steps of this algorithm are:

- Determination of the system order n
- Building the $[p(n+1) \times m(n+1)]$ matrices $\mathbf{T}$ and $\mathbf{R}$, Eq.(4.72b)
- Calculation of the $[m(n+1) \times m]$ matrix $\mathbf{D}_c$ as a null space of $\mathbf{T}$, where $\mathbf{D}_c$ should have the structure:

$$\mathbf{D}_c = \left[d_0\mathbf{I}_m \quad d_1\mathbf{I}_m \quad \cdots \quad d_n\mathbf{I}_m \right]^T, \quad d_n = 1 \tag{4.74}$$

- Calculation of $\mathbf{W}_c$ using $\mathbf{W}_c = \mathbf{R}\,\mathbf{D}_c$

To take into account Eq.(4.74), specifically $d_n = 1$, Eq.(4.72b) will be rewritten as

$$\mathbf{T}_1\mathbf{D}_{c1} = -\mathbf{T}_2 \tag{4.75}$$

where $[p(n+1) \times mn]$ and $[p(n+1) \times m]$ matrices $\mathbf{T}_1$ and $\mathbf{T}_2$ satisfy:

$$\mathbf{T} = [\,\mathbf{T}_1 \quad \mathbf{T}_2\,]$$

while $\mathbf{D}_{c1}$ contains the first n blocks $d_i\mathbf{I}_m$, $i=[0,n\text{-}1]$ from $\mathbf{D}_c$.

Equation (4.75) represents a system of algebraic equations, where $\mathbf{T}_1$ is not a full column rank matrix, leading to a non-unique solution for $\mathbf{D}_{c1}$. A procedure of calculating $\mathbf{D}_{c1}$ of the structure in Eq.(4.74) satisfying Eq.(4.75) is given in Appendix B. Algorithmic implementation of this procedure is included in Algorithm *HTFm*, given in the following.

Algorithm *HTFm*:

Syntax: $\mathbf{H}, \epsilon$ (*HTFm*) $\Rightarrow$ **d**, **W**

Input/Output Arguments:

- **H** is a $(pm \times M)$ matrix in PMF. The rows of **H** contain the first M coefficients of the polynomials $h_{ij}(z^{-1})$ in $H(z^{-1})$.
- ϵ is a sufficiently small positive number used in rank calculations.
- **d** is an $(n+1)$ dimensional row containing the coefficients d_i.
- **W** is a $[pm \times (n+1)]$ matrix in PMF. Its rows contain the coefficients of the polynomials $w_{ij}(z)$ in $W(z)$.

Algorithm:

```
1.  Set  H (Alt) ⇒ H_c, H_r
2.  Set  0 ⇒ k and 0 ⇒ n_a
```

```
3.  Set $k+1 \Rightarrow k$
4.  Build $\mathbf{T}$, using in Eq.(4.72b) $k$ instead of $n$, and set rank( $\mathbf{T}$ ) $\Rightarrow n$
5.  If $n = n_o$, go to 6; else, set $n \Rightarrow n_o$ and go to 3
6.  Set $n \Rightarrow k$ and build $\mathbf{T}$ and $\mathbf{R}$, Eq.(4.72b)
7.  Partition $\mathbf{T} \Rightarrow [\ \mathbf{T}_1 \mid \mathbf{T}_2\ ]$, $\mathbf{T}_2$ has $m$ columns
8.  Set $\mathbf{T}_1$ (Null) $\Rightarrow \mathbf{N}$, $\mathbf{N}$ satisfies $\mathbf{T}_1\,\mathbf{N} = \mathbf{0}$
9.  Solve $\mathbf{T}_1\mathbf{Y} = -\mathbf{T}_2$ for $\mathbf{Y}$
10. Set [1 0 0 ... 0] ($m$-1 zeros) $\Rightarrow \mathbf{v}$ and $[\mathbf{v} \mid \mathbf{v} \mid \ldots \mid \mathbf{v}] \Rightarrow \mathbf{v}_n$
11. Set $0 \Rightarrow i$ and [1 1 ... 1] $\Rightarrow \mathbf{v}_1$ , $\mathbf{v}_1$ has $mn$ unity elements
12. Set $i+1 \Rightarrow i$
13. Set $\mathbf{v}_1 - \mathbf{v}_n \Rightarrow \mathbf{v}_i$, $\mathbf{v}_i$ (DSM) $\Rightarrow \mathbf{S}_i$ and $\mathbf{S}_i^T\mathbf{N} \Rightarrow \mathbf{N}_i$
14. If rank ( $\mathbf{N}_i$ ) = $n(m-1)$, go to 16; else, go to 15
15. Shift $\mathbf{v}_n$ by one column right, i.e. $\mathbf{v}_n$ (SHR) $\Rightarrow \mathbf{v}_n$ and go to 12
16. Extract the $i^{th}$ column from $\mathbf{Y} \Rightarrow y_i$ and set $\mathbf{S}_i^T\mathbf{y}_i \Rightarrow \mathbf{y}_s$
17. Set $\mathbf{N}_i^{-1}\mathbf{y}_s \Rightarrow \mathbf{t}_i$ and $\mathbf{N}\,\mathbf{t}_i \Rightarrow \mathbf{d}$
18. Set $\mathbf{v}_n$ (DSM) $\Rightarrow \mathbf{S}$, $S^T\,\mathbf{d} \Rightarrow \mathbf{d}$
19. Set $[\ \mathbf{d}^T \mid 1\ ] \Rightarrow \mathbf{d}$
20. Using $\mathbf{d}$, build $\mathbf{D}_c$ and set $\mathbf{R}\,\mathbf{D}_c \Rightarrow \mathbf{W}_c$
21. Set $\mathbf{W}_c$, $p$ (PMFc) $\Rightarrow \mathbf{W}$
```

It may be verified (see Appendix B) that Steps 7 to 19 of the above algorithm calculate coefficients d_i, $i=[0,n]$ defining $\mathbf{D}_c$ satisfying Eqs.(4.72b) and (4.74). In Step 10, the m dimensional row $\mathbf{v}$ has one unity and m-1 zeros, while the mn dimensional row $\mathbf{v}_n$ is obtained by concatenating the vector $\mathbf{v}$ n times. Algorithm *DSM* used in Steps 13 and 18 generates a "selector" matrix corresponding to the row used as the input argument. The relationship between the input and output arguments in the "define selector matrix" (DSM) algorithm is explained in Section 3.3.4. and Eq.(3.79). The "shift right" (SHR) algorithm used in step 15 simply shifts input argument by one column to right. The last element (column) is lost, while the zero element (or zero column) is added as the first element (column).

Using the principle of duality, it is relatively straightforward to develop Algorithm *HTFp* which, considering the first of the two versions of Eqs.(4.71) to (4.73) calculates $d(z)$ and $W(z)$ satisfying Eq.(4.70). As was mentioned previously, within Algorithm *HTF*, having the same input/output arguments as *HTFm*, both algorithms are available, and only one is called depending on the relationship between m and p. It is left as an exercise for the reader to develop *HTFp*. Interested readers may check the *L-A-S* implementations of these algorithms.

4.3.6 Discrete-Time Response from Markov Parameters

This algorithm calculates the zero-state response of a D-T system given the Markov parameters and the samples of the input signals using D-T convolution, derived from $y(z) = H(z^{-1})\,u(z)$ i.e.

$$\mathbf{y}_i = \sum_{j=0}^{i} \mathbf{H}_{i-j}\mathbf{u}_j \tag{4.76}$$

where $\mathbf{u}_i$ and $\mathbf{y}_j$ are i^{th} and j^{th} samples of the input and output vectors, respectively, while $\mathbf{H}_k$ is the k^{th} Markov parameter. Equation (4.76) may be represented by:

$$\begin{bmatrix} \mathbf{y}_0 \\ \mathbf{y}_1 \\ \vdots \\ \mathbf{y}_{L-1} \end{bmatrix} = \begin{bmatrix} \mathbf{H}_0 & & & \\ \mathbf{H}_1 & \mathbf{H}_0 & & \\ & \dots & \ddots & \\ \mathbf{H}_{L-1} & \cdots & \mathbf{H}_1 & \mathbf{H}_0 \end{bmatrix} \begin{bmatrix} \mathbf{u}_0 \\ \mathbf{u}_1 \\ \vdots \\ \mathbf{u}_{L-1} \end{bmatrix} \tag{4.77}$$

where $L = \min\{M,N\}$, or for short by:

$$\mathbf{y}_c = \mathbf{H}\,\mathbf{u}_c \tag{4.78}$$

where $\mathbf{y}_c$ and $\mathbf{u}_c$ are Lp and Lm dimensional columns containing samples $\mathbf{y}_i$ and $\mathbf{u}_i$, $i=[0,L\text{-}1]$, respectively, while $\mathbf{H}$ is an $(Lp \times Lm)$ matrix containing the Markov parameters $\mathbf{H}_j$, $j=[0,L\text{-}1]$ arranged according to Eq.(4.77).

Syntax: u, H (*uHy*) ⇒ y

Input/Output Arguments:

- u is an $(m \times M)$ matrix containing samples of the m-dimensional input vector.
- H is a $(pm \times M)$ matrix in PMF. The rows of H contain the coefficients h_{ijk} of polynomials $h_{ij}(z^{-1})$ in $H(z^{-1})$.
- y is the $(p \times L)$ matrix containing the samples of the system response. $L = \min\{M,N\}$.

4.4 Conversions from MFD Models

Given either a left or right coprime ARMA (MFD) model, this section discusses the conversion to either state space or transfer function models as well as the calculation of a set of Markov parameters. If it is desired to calculate the system response, it is recommended that the ARMA model first be converted to state space form, then to use the state space representation to calculate the response.

4.4.1 MFD to Observable State Model

This algorithm calculates an observable form state space model $R_o = \{\mathbf{A}_o, \mathbf{B}_o, \mathbf{C}_o, \mathbf{D}_o\}$ from a corresponding left coprime column-reduced ARMA (MFD) model, $D(z)^{-1}N(z)$. One may view this algorithm as an "inverse" of Algorithm *RoDN*, described in Section 4.1.7. It is based on Eqs.(4.17) to (4.20), as well as Eqs.(4.44) to (4.47). For completeness of the algorithm presentation, some of these expressions will be repeated here.

From Section 3.4, where MFD system description was introduced, it is clear that the locations of the p columns of the identity matrix $\mathbf{I}_p$ in the matrix $\mathbf{D}_r$, Eq.(3.104), uniquely determine the column degrees of $D(z)$. Also, as has been already stated several times (Sections 4.2.7), the set of column degrees $\{ n_i \}$ of $D(z)$ corresponds to a set of admissible POI $\nu = \{ \nu_i \}$ used in building a state space representation R_o in a POF.

Thus, from a given left coprime MFD, where $D(z)$ is column-reduced and monic, it is first necessary to determine the set of column degrees. This is done by another polynomial matrix "service" algorithm refered to as *D2nv*:

Syntax: $\mathbf{D}_r, \epsilon\ (D2nv) \Rightarrow \nu$ (4.79)

Input/Output Arguments:

- $\mathbf{D}_r$ is a $[p \times (k+1)p]$ matrix of the structure in Eq.(3.104).
- ϵ is a sufficiently small positive scalar used as "machine" zero.
- ν is a p-diminsional row containing the column degrees of $D(z)$, or, as was mentioned earlier, a set of POI ν to be used in building the representation R_o.

Having determined the set ν, it is then necessary to extract the n columns from $\mathbf{D}_r$ containing non-zero non-unity elements. From the expression

$$\mathbf{y}_2 - \mathbf{A}_r\,\mathbf{y}_1 = \mathbf{N}_r\,\mathbf{u}_1$$

derived in Section 4.1.7, Eq.(4.17), it is clear that these n p-dimensional columns constitute the matrix $-\mathbf{A}_r$. At the same time p rows in this $\mathbf{A}_r$ represent the non-zero non-unity "parametric" rows of the matrix $\mathbf{A}_o$ in the representation R_o. Thus, the matrix $\mathbf{A}_r$ can be obtained from Eq.(4.45), i.e.:

$$-\mathbf{D}_r\mathbf{S}_{II} \Rightarrow \mathbf{A}_r$$

where $\mathbf{S}_{II}$ is a selector matrix, Eq.(3.79), corresponding to the set ν. The complete Algorithm *DNRo* is as follows.

Syntax: $\mathbf{D}, \mathbf{N}, \epsilon\ (DNRo) \Rightarrow \mathbf{A}_o, \mathbf{B}_o, \mathbf{C}_o, \mathbf{D}_o, \nu$

Input/Output Arguments:

- $\mathbf{D}$ is a $[p^2 \times (k+1)]$ matrix in the PMF. The rows of $\mathbf{D}$ contain the coefficients d_{ijk} of the polynomials $d_{ij}(z)$ in $D(z)$.
- $\mathbf{N}$ is a $[pm \times (k+1)]$ matrix in the PMF. The rows of $\mathbf{N}$ contain the coefficients n_{ijk} of the polynomials $n_{ij}(z)$ in $N(z)$.
- ϵ is a sufficiently small positive number used in evaluating the set ν using Eq.(4.79).
- $R_o = \{\mathbf{A}_o, \mathbf{B}_o, \mathbf{C}_o, \mathbf{D}_o\}$, state space model in a POF.
- $\nu = \{ \nu_i \}$ is an admissible set of POI corresponding to R_o.

Algorithm:

1. Set $\mathbf{D}$ (*Alt*) $\Rightarrow \mathbf{D}_c, \mathbf{D}_r$, $\mathbf{N}$ (*Alt*) $\Rightarrow \mathbf{N}_c, \mathbf{N}_r$
2. Set $\mathbf{D}_c, \epsilon$ (*D2nv*) $\Rightarrow \nu$
3. Set $\{\nu_i\}$ (*SMat*) $\Rightarrow \nu_m, \mathbf{S}_a, \mathbf{S}_l, \mathbf{S}_{ll}, \mathbf{S}_{lt}$
4. Set $-\mathbf{D}_r\mathbf{S}_{ll} \Rightarrow \mathbf{A}_r$
5. Partition $\mathbf{I}_n = \begin{bmatrix} \mathbf{C}_o \\ \mathbf{A}_2 \end{bmatrix}$, $\mathbf{C}_o$ has p rows
6. Set $\mathbf{S}_l\mathbf{A}_2 + \mathbf{S}_a\mathbf{A}_r \Rightarrow \mathbf{A}_o$
7. Set $\mathbf{A}_o, \mathbf{S}_a$ (*Qc*) $\Rightarrow \mathbf{Q}_c$, $\mathbf{Q}_c$ has ν_m+1 blocks $\{\mathbf{A}_o^i\mathbf{S}_a\}$ of p columns
8. Set $\mathbf{Q}_c\mathbf{N}_c \Rightarrow \mathbf{B}_o$
9. Set $D_o^{-1}(s_o)N_o(s_o) - \mathbf{C}_o(s_o\mathbf{I}-\mathbf{A}_o)^{-1}\mathbf{B}_o \Rightarrow \mathbf{D}_o$ for any $s_o \neq$ a system pole

Note that Steps 5 to 8 in *DNRo* are the same as the corresponding steps at the end of *TFRo*, Section 4.2.7, which is to be expected since both algorithms determine the state space model R_o in a POF. For more details, and particularly for the case when $\mathbf{A}_r$ and $\mathbf{A}_m$ are singular, see Eqs.(4.44) to (4.47).

4.4.2 MFD to Controllable State Model

This algorithm calculates a controllable form state space model $R_c = \{\mathbf{A}_c, \mathbf{B}_c, \mathbf{C}_c, \mathbf{D}_c\}$ from a corresponding right coprime row-reduced ARMA (MFD) model, $N(z)D^{-1}(z)$. Since this algorithm is dual to the previously given *DNRo*, as well as rather similar to Algorithm *TFRc* given in Section 4.2.7, it will only be listed here for reference without discussion.

Syntax: $\mathbf{N}, \mathbf{D}, \epsilon$ (*NDRc*) $\Rightarrow \mathbf{A}_c, \mathbf{B}_c, \mathbf{C}_c, \mathbf{D}_c, \mu$

Input/Output Arguments:

- $\mathbf{N}$ is a $[pm \times (k+1)]$ matrix in the PMF. The rows of $\mathbf{N}$ contain the coefficients n_{ijh} of the polynomials $n_{ij}(z)$ in $N(z)$.

- $\mathbf{D}$ is a $[m^2 \times (k+1)]$ matrix in the PMF. The rows of $\mathbf{D}$ contain the coefficients d_{ijh} of the polynomials $d_{ij}(z)$ in $D(z)$.
- ϵ is a sufficiently small positive number used in evaluating the set μ using Eq.(4.79).
- $R_c = \{\mathbf{A}_c, \mathbf{B}_c, \mathbf{C}_c, \mathbf{D}_c\}$, state space model in a PCF.
- $\mu = \{\ \mu_i\ \}$ is an admissible set of PCI corresponding to R_c.

Algorithm:

```
1.    Set $\mathbf{D}$ (Alt) $\Rightarrow \mathbf{D}_c, \mathbf{D}_r$ , $\mathbf{N}$ (Alt) $\Rightarrow \mathbf{N}_c, \mathbf{N}_r$
2.    Set $\mathbf{D}_c^T$, $\epsilon$ (D2nv) $\Rightarrow \mu$
3.    Set $\{\mu_i\}$ (SMat) $\Rightarrow \mu_m, \mathbf{S}_a, \mathbf{S}_i, \mathbf{S}_k, \mathbf{S}_M$
4.    Set $-\mathbf{S}_k^T\mathbf{D}_c \Rightarrow \mathbf{A}_{cc}$
5.    Partition $\mathbf{I}_n \Rightarrow [\ \mathbf{B}_c \quad \mathbf{A}_2\ ]$, $\mathbf{B}_c$ has $m$ columns
6.    Set $\mathbf{A}_2\mathbf{S}_i^T + \mathbf{A}_{cc}\mathbf{S}_a^T \Rightarrow \mathbf{A}_c$
7.    Set $\mathbf{A}_c, \mathbf{S}_a^T$ (Qo) $\Rightarrow \mathbf{Q}_o$ , $\mathbf{Q}_o$ has $\mu_m+1$ blocks $\{\mathbf{S}_a^T\mathbf{A}_c^i\}$ of $m$ rows
8.    Set $\mathbf{N}_r\mathbf{Q}_o \Rightarrow \mathbf{C}_c$
9.    Set $N(s_o)D_o^{-1}(s_o) - \mathbf{C}_c(s_o\mathbf{I}-\mathbf{A}_c)^{-1}\mathbf{B}_c \Rightarrow \mathbf{D}_c$ for any $s_o \neq$ a system pole
```

Note that Steps 5 to 8 in *NDRc* are the same as the corresponding steps at the end of *TFRc*, Section 4.2.7, which is to be expected since both algorithms determine the state space model R_c in a PCF. For more details, and particularly for the case when $\mathbf{A}_c$ and $\mathbf{A}_{cco}$ are singular, see Eqs.(4.44) to (4.47).

It is worth mentioning that the duality between Algorithms *DNRo* and *NDRc* is quite apparent. In other words, instead of:

$$\mathbf{D}, \mathbf{N}, \epsilon\ (NDRc) \Rightarrow \mathbf{A}_c, \mathbf{B}_c, \mathbf{C}_c, \mathbf{D}_c, \mu$$

the following sequence of algorithms may be used:

$$\mathbf{N}^T \Rightarrow \mathbf{N}_d\ , \quad \mathbf{D}^T \Rightarrow \mathbf{D}_d$$
$$\mathbf{D}_d, \mathbf{N}_d, \epsilon\ (DNRo) \Rightarrow \mathbf{A}_d, \mathbf{B}_d, \mathbf{C}_d, \mathbf{D}_d, \mu$$
$$\mathbf{A}_d^T \Rightarrow \mathbf{A}_c\ , \quad \mathbf{B}_d^T \Rightarrow \mathbf{C}_c\ , \quad \mathbf{C}_d^T \Rightarrow \mathbf{B}_c\ , \quad \mathbf{D}_d^T \Rightarrow \mathbf{D}_c$$

4.4.3 Left Coprime MFD to Markov Parameters

This algorithm calculates the set of Markov parameters from a left coprime column-reduced ARMA (MFD) model, $D(z)^{-1}N(z)$. This algorithm is, in a way, the "inverse" of Algorithm *HDN*, given in Section 4.3.3. Thus, it is based on the same expressions given in Eqs.(4.57) to (4.58), i.e.:

$$G(z) = H(z^{-1}) = D^{-1}(z)N(z)$$

$$\sum_{j=0}^{i} \mathbf{D}_{k-j}\mathbf{H}_{i-j} = \mathbf{N}_{k-i} \quad \text{for} \quad i=[0,k] \quad \text{and}$$

or

$$\sum_{j=0}^{k} \mathbf{D}_{k-j}\mathbf{H}_{i-j} = \mathbf{0} \quad \text{for} \quad i=[k+1,\infty]$$

However, since here the submatrices $\mathbf{D}_i$ and $\mathbf{N}_i$, $i=[0,k]$ are given, and the first M Markov parameters $\mathbf{H}_j$, $j=[0,M\text{-}1]$, are sought, the above equations will now be represented by:

$$\begin{bmatrix} & & & & \mathbf{D}_0 \\ & & & \mathbf{D}_0 & \mathbf{D}_1 \\ & & \cdot & \mathbf{D}_1 & \vdots \\ & & \cdot & \vdots & \mathbf{D}_k \\ & \mathbf{D}_0 & & \mathbf{D}_k & \\ \mathbf{D}_0 & \mathbf{D}_1 & \cdot & & \\ \mathbf{D}_1 & \vdots & \cdot & & \\ \vdots & \mathbf{D}_k & & & \\ \mathbf{D}_k & & & & \end{bmatrix} \begin{bmatrix} \mathbf{H}_0 \\ \vdots \\ \mathbf{H}_{M-1} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \\ \mathbf{N}_0 \\ \vdots \\ \mathbf{N}_k \end{bmatrix} \tag{4.80}$$

which could readily be used for calculating $\mathbf{H}_j$, given $\mathbf{D}_i$ and $\mathbf{N}_i$. However, as was mentioned in Section 4.2.2, Algorithm *TFH*, see Eqs.(4.31) and (4.32), Eq.(4.80) assumes that $\mathbf{H}_{M\text{-}1}$ satisfies:

$$|\mathbf{H}_{M-1}| << 1$$

Thus, the criterion for selecting the scalar M is that the norm of the last calculated Markov parameter $\mathbf{H}_{M\text{-}1}$, with $h_M = \| \mathbf{H}_{M\text{-}1} \|$, should be sufficiently small. In other words, the algorithm based on Eq.(4.80) is applicable only to MFDs whose characteristic polynomials $d(z) = \det\{ D(z) \}$ have all roots within unit circle. If this is not the case, then either:

- "time scaling" of $D(z)$ and $N(z)$ should be performed, or
- one of the following sequences of algorithms could be used

$$\mathbf{D},\ \mathbf{N},\ \epsilon\ (DNRo) \Rightarrow \mathbf{A}_o,\ \mathbf{B}_o,\ \mathbf{C}_o,\ \mathbf{D}_o,\ \nu$$
$$\mathbf{A}_o,\ \mathbf{B}_o,\ \mathbf{C}_o,\ \mathbf{D}_o,\ M\ (SSH) \Rightarrow \mathbf{H},\ h_M$$

or

$$\mathbf{D},\ \mathbf{N},\ \epsilon\ (DNTF) \Rightarrow \mathbf{d},\ \mathbf{W}$$
$$\mathbf{d},\ \mathbf{W},\ M\ (TFH) \Rightarrow \mathbf{H},\ h_M$$

The "time scaling" of $D(z)$, $N(z)$ is explained in the Example 2, given at the end of this chapter. Algorithm *DNH* is as follows:

Syntax: $\mathbf{D}, \mathbf{N}, M\ (DNH) \Rightarrow \mathbf{H}, h_M$

Input/Output Arguments:

- $\mathbf{D}$ is a $[p^2 \times (k+1)]$ matrix in the PMF. The rows of $\mathbf{D}$ contain the coefficients d_{ijk} of the polynomials $d_{ij}(z)$ in $D(z)$.
- $\mathbf{N}$ is a $[pm \times (k+1)]$ matrix in the PMF. The rows of $\mathbf{N}$ contain the coefficients n_{ijk} of the polynomials $n_{ij}(z)$ in $N(z)$.
- M is scalar specifying the number of Markov parameters $\mathbf{H}_i$, $i = [0, M\text{-}1]$, to be calculated.
- $\mathbf{H}$ is a $[pm \times M]$ matrix in PMF. Rows of $\mathbf{H}$ contain the first M coefficients of the polynomials $h_{ij}(z^{-1})$ in $H(z^{-1})$.
- h_M is a scalar equal to $\|\mathbf{H}_{M\text{-}1}\|$, where $\mathbf{H}_{M\text{-}1}$ is the last Markov parameter calculated.

Algorithm:

1. Set $\mathbf{D}\ (Alt) \Rightarrow \mathbf{D}_c, \mathbf{D}_r$, $\mathbf{N}\ (Alt) \Rightarrow \mathbf{N}_c, \mathbf{N}_r$
2. Build the matrices in Eq.(4.80) consisting of $\mathbf{D}_i$ and $\mathbf{N}_i$
3. Solve Eq.(4.80) for $\mathbf{H}_c$ containing $\mathbf{H}_j$, $j=[0,M\text{-}1]$
4. Set $\|\ \mathbf{H}_{M\text{-}1}\ \| \Rightarrow h_M$
5. Set $\mathbf{H}_c, p\ (PMFc) \Rightarrow \mathbf{H}$

4.4.4 Right Coprime MFD to Markov Parameters

This algorithm calculates the set of Markov parameters from a right coprime row-reduced ARMA (MFD) model, $N(z)D^{-1}(z)$. As was mentioned in the previous algorithm, this algorithm is like an "inverse" to Algorithm *HND*, given in Section 4.3.4. Thus, it is based on the same expressions given in Eqs.(4.64) to (4.65), i.e.:

$$G(z) = H(z^{-1}) = N(z)D^{-1}(z)$$

or

$$\sum_{j=0}^{i} \mathbf{H}_{i-j}\mathbf{D}_{k-j} = \mathbf{N}_{k-i} \quad \text{for} \quad i = [0,k] \quad \text{and}$$

$$\sum_{j=0}^{k} \mathbf{H}_{i-j}\mathbf{D}_{k-j} = \mathbf{0} \quad \text{for} \quad i = [k+1, \infty]$$

However, since here the submatrices $\mathbf{N}_i$ and $\mathbf{D}_i$, $i=[0,k]$ are given, and the first M

Markov parameters $\mathbf{H}_j$, $j=[0,M\text{-}1]$, are sought, the above equations will now be represented by:

$$[\mathbf{H}_0 \mid \cdots \mid \mathbf{H}_{M-1}] \begin{bmatrix} & & & & \mathbf{D}_0 & \mathbf{D}_1 & \cdots & \mathbf{D}_k \\ & & & \mathbf{D}_0 & \mathbf{D}_1 & \cdots & \mathbf{D}_k & \\ & & \cdot & & & \cdot & & \\ & \cdot & & & \cdot & & & \\ & \cdot & & \cdot & & & & \\ & \mathbf{D}_0 & \mathbf{D}_1 & \cdots & \mathbf{D}_k & & & \\ \mathbf{D}_0 & \mathbf{D}_1 & \cdots & \mathbf{D}_k & & & & \end{bmatrix} = \tag{4.81}$$

$$\begin{bmatrix} 0 & 0 & \cdots & 0 & \cdots & 0 & \mathbf{N}_0 & \mathbf{N}_1 & \cdots & \mathbf{N}_k \end{bmatrix}$$

Since this algorithm is dual to the previous algorithm, the comments about the applicability of Algorithm *DNH*, given above, hold also here in the dual sense.

The algorithm is described as follows:

Syntax: $\mathbf{N}$, $\mathbf{D}$, M (*NDH*) $\Rightarrow$ $\mathbf{H}$, h_M

Input/Output Arguments:

- $\mathbf{N}$ is a $[pm \times (k+1)]$ matrix in the PMF. The rows of $\mathbf{N}$ contain the coefficients n_{ijk} of the polynomials $n_{ij}(z)$ in $N(z)$.
- $\mathbf{D}$ is a $[m^2 \times (k+1)]$ matrix in the PMF. The rows of $\mathbf{D}$ contain the coefficients d_{ijk} of the polynomials $d_{ij}(z)$ in $D(z)$.
- M is scalar specifying the number of Markov parameters $\mathbf{H}_i$, $i = [0, M\text{-}1]$, to be calculated.
- $\mathbf{H}$ is a $[pm \times M]$ matrix in PMF. Rows of $\mathbf{H}$ contain the first M coefficients of the polynomials $h_{ij}(z^{-1})$ in $H(z^{-1})$.
- h_M is a scalar equal to $\|\mathbf{H}_{M-1}\|$, where $\mathbf{H}_{M-1}$ is the last Markov parameter calculated.

Algorithm:

1. Set $\mathbf{N}$ (*Alt*) $\Rightarrow$ $\mathbf{N}_c$, $\mathbf{N}_r$, $\mathbf{D}$ (*Alt*) $\Rightarrow$ $\mathbf{D}_c$, $\mathbf{D}_r$
2. Build the matrices in Eq.(4.81) consisting of $\mathbf{N}_r$ and $\mathbf{D}_r$
3. Solve Eq.(4.81) for $\mathbf{H}_r$ containing $\mathbf{H}_j$, $j=[0,M\text{-}1]$
4. Set $\|\mathbf{H}_{M-1}\| \Rightarrow h_M$
5. Set $\mathbf{H}_r$, m (*PMFr*) $\Rightarrow$ $\mathbf{H}$

To stress the duality between Algorithms *DNH* and *NDH*, let us mention that instead of

$$\mathbf{N},\ \mathbf{D},\ M\ (NDH) \Rightarrow \mathbf{H},\ h_M$$

the following sequence of algorithms could be used:

$$\mathbf{N}^T \Rightarrow \mathbf{N}_d\,,\quad \mathbf{D}^T \Rightarrow \mathbf{D}_d$$
$$\mathbf{D}_d,\ \mathbf{N}_d,\ M\ (DNH) \Rightarrow \mathbf{H}_d,\ h_M$$
$$\mathbf{H}_d^{\,T} \Rightarrow \mathbf{H}$$

4.4.5 Left Coprime MFD to Transfer Function

This algorithm converts a left coprime column-reduced C-T or D-T ARMA (MFD) model, $D^{-1}(z)N(z)$, to a matrix transfer function $G(z) = \mathbf{C}(z\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D}$. This algorithm is based on

$$D^{-1}(z)\ N(z) = W(z)/d(z)$$

which, as has been shown in Section 4.1.7, Eq.(4.21), may be rewritten as:

$$T(z)\ N(z) = W(z) \tag{4.82}$$

where $T(z) = \text{adj}\{\ D(z)\ \}$ and $d(z) = \det\{\ D(z)\ \}$. Using Eq.(4.82), the calculation of $W(z)$ reduces to a simple multiplication of polynomial matrices $T(z)$ and $N(z)$.

It should be mentioned that the adjoint of a square polynomial matrix, i.e. $\text{adj}\{\ D(z)\ \}$, is calculated by applying the Laplace expansion and direct evaluation of minors and cofactors involving polynomial manipulation, which has proven to give satisfactory accuracy for polynomial matrices of orders up to 10.

Syntax: $\mathbf{D},\ \mathbf{N},\ \epsilon\ (DNTF) \Rightarrow \mathbf{d},\ \mathbf{W}$

Input/Output Arguments:

- $\mathbf{D}$ is a $[p^2 \times (k+1)]$ matrix in the PMF. The rows of $\mathbf{D}$ contain the coefficients d_{ijk} of the polynomials $d_{ij}(z)$ in $D(z)$.
- $\mathbf{N}$ is a $[pm \times (k+1)]$ matrix in the PMF. The rows of $\mathbf{N}$ contain the coefficients n_{ijk} of the polynomials $n_{ij}(z)$ in $N(z)$.
- ϵ is a small positive number used as machine zero.
- $\mathbf{d}$ is an $(n+1)$ dimensional row containing the coefficients of $d(z)$.
- $\mathbf{W}$ is a $[pm \times n+1]$ matrix in the PMF. The rows of $\mathbf{W}$ contain the coefficients w_{ijk} of the polynomials $w_{ij}(z)$ of $W(z)$.

The basic steps of the algorithm are:

```
1.  Set det{ $D(z)$ } ⇒ $d(z)$
2.  Set adj{ $D(z)$ } ⇒ $T(z)$
3.  If the highest order coefficient in $d(z)$, $d_n = 1$, go to 5;
    else, go to 4
4.  Set $-d(z)$ ⇒ $d(z)$ and $-T(z)$ ⇒ $T(z)$
5.  Set $T(z)\,N(z)$ ⇒ $W(z)$
```

4.4.6 Right Coprime MFD to Transfer Function

This algorithm converts a right coprime row-reduced C-T or D-T ARMA (MFD) model, $N(z)D^{-1}(z)$, to a matrix transfer function $G(z) = \mathbf{C}(z\mathbf{I} - \mathbf{A})^{-1}\mathbf{B} + \mathbf{D}$. This algorithm is based on

$$N(z)\,D^{-1}(z) = W(z)/d(z)$$

which, as has been shown in Section 4.1.8, Eq.(4.24), may be rewritten as:

$$N(z)\,T(z) = W(z) \tag{4.83}$$

where $T(z)$ = adj{ $D(z)$ } and $d(z)$ = det{ $D(z)$ }. Using Eq.(4.83), the calculation of $W(z)$ reduces to a simple multiplication of polynomial matrices $N(z)$ and $T(z)$.

The syntax of the algorithm is:

$$\mathbf{N},\ \mathbf{D},\ \epsilon\ (NDTF) \Rightarrow \mathbf{d},\ \mathbf{W}$$

Input/Output Arguments:

- **N** is a $[pm \times (k+1)]$ matrix in the PMF. The rows of **N** contain the coefficients n_{ijk} of the polynomials $n_{ij}(z)$ in $N(z)$.
- **D** is a $[m^2 \times (k+1)]$ matrix in the PMF. The rows of **D** contain the coefficients d_{ijk} of the polynomials $d_{ij}(z)$ in $D(z)$.
- ϵ is a small positive number used as machine zero.
- **d** is an $(n+1)$ dimensional row containing the coefficients of $d(z)$.
- **W** is a $[pm \times n+1]$ matrix in the PMF. The rows of **W** contain the coefficients w_{ijk} of the polynomials $w_{ij}(z)$ of $W(z)$.

For more details see Section 4.4.5, Algorithm *DNTF*.

4.4.7 Other MFD Conversion Algorithms

For completeness in algorithm availability, the following four algorithms are also available:

(1) *DNRc*: Left MFD $\{D(z), N(z)\}$ into a state space model R_c in a PCF
(2) *DNND*: Left MFD $\{D(z), N(z)\}$ into a right coprime MFD $\{\tilde{N}(z), \tilde{D}(z)\}$
(3) *NDRo*: Right MFD $\{\tilde{N}(z), \tilde{D}(z)\}$ into a state space model R_o in a POF
(4) *NDDN*: Right MFD $\{\tilde{N}(z), \tilde{D}(z)\}$ into a left coprime MFD $\{D(z), N(z)\}$

The syntax of these algorithms are as follows:

$$\begin{aligned}
&\mathbf{D},\ \mathbf{N},\ \epsilon,\ \mu_d\ (\mathit{DNRc}) \Rightarrow \mathbf{A}_c,\ \mathbf{B}_c,\ \mathbf{C}_c,\ \mathbf{D}_c,\ \mu,\ C\# \\
&\mathbf{D},\ \mathbf{N},\ \epsilon,\ \mathbf{n}_d\ (\mathit{DNND}) \Rightarrow \tilde{\mathbf{N}},\ \tilde{\mathbf{D}},\ C\# \\
&\tilde{\mathbf{N}},\ \tilde{\mathbf{D}},\ \epsilon,\ \nu_d\ (\mathit{NDRo}) \Rightarrow \mathbf{A}_o,\ \mathbf{B}_o,\ \mathbf{C}_o,\ \mathbf{D}_o,\ \nu,\ C\# \\
&\tilde{\mathbf{N}},\ \tilde{\mathbf{D}},\ \epsilon,\ \mathbf{n}_d\ (\mathit{NDDN}) \Rightarrow \mathbf{D},\ \mathbf{N},\ C\#
\end{aligned} \tag{4.84}$$

For input/output arguments see Algorithms:

$$\mathit{TFRc}\ ,\ \mathit{TFND}\ ,\ \mathit{TFRo}\ \text{ and }\ \mathit{TFDN} \tag{4.85}$$

respectively, as well as some other previously discussed algorithms.

It is worth mentioning that these model conversions may, for instance, be performed by the following sequences of algorithms already described:

- Instead of (1):
  $\mathbf{D},\ \mathbf{N},\ \epsilon\ (\mathit{DNRo}) \Rightarrow \mathbf{A}_o,\ \mathbf{B}_o,\ \mathbf{C}_o,\ \mathbf{D}_o,\ \nu$
  (5) $\mathbf{A}_o,\ \mathbf{B}_o,\ \mathbf{C}_o,\ \mathbf{D}_o,\ \mu_d\ (\mathit{SSRc}) \Rightarrow \mathbf{A}_c,\ \mathbf{B}_c,\ \mathbf{C}_c,\ \mathbf{D}_c,\ C\#$

- Instead of (2): the sequence (5) and
  $\mathbf{n}_d \Rightarrow \{\ \mu_i\ \}$
  $\mathbf{A}_c,\ \mathbf{B}_c,\ \mathbf{C}_c,\ \mathbf{D}_c,\ \mu\ (\mathit{RcND}) \Rightarrow \tilde{\mathbf{N}},\ \tilde{\mathbf{D}}$ (4.86)

- Instead of (3):
  $\tilde{\mathbf{N}},\ \tilde{\mathbf{D}},\ \epsilon\ (\mathit{NDRc}) \Rightarrow \mathbf{A}_c,\ \mathbf{B}_c,\ \mathbf{C}_c,\ \mathbf{D}_c,\ \mu$
  (6) $\mathbf{A}_c,\ \mathbf{B}_c,\ \mathbf{C}_c,\ \mathbf{D}_c,\ \nu_d\ (\mathit{SSRo}) \Rightarrow \mathbf{A}_o,\ \mathbf{B}_o,\ \mathbf{C}_o,\ \mathbf{D}_o,\ C\#$

- Instead of (4): the sequence (6) and
  $\mathbf{n}_d \Rightarrow \{\ \nu_i\ \}$
  $\mathbf{A}_o,\ \mathbf{B}_o,\ \mathbf{C}_o,\ \mathbf{D}_o,\ \nu\ (\mathit{RoDN}) \Rightarrow \mathbf{D},\ \mathbf{N}$

Of course, the algorithms to be described in this section are computationally more convenient than the sequences suggested above. All four algorithms are based on:

$$D^{-1}(z)\,N(z) = \tilde{N}(z)\,\tilde{D}^{-1}(z) \tag{4.87}$$

which may be rewritten as:

$$D(z)\,\tilde{N}(z) - N(z)\,\tilde{D}(z) = \mathbf{0} \tag{4.88}$$

where:

$$D(z) = \sum_{i=0}^{k} \mathbf{D}_i z^i\,, \quad N(z) = \sum_{i=0}^{k} \mathbf{N}_i z^i\,, \quad \tilde{D}(z) = \sum_{i=0}^{h} \tilde{\mathbf{D}}_i z^i\,, \quad \tilde{N}(z) = \sum_{i=0}^{h} \tilde{\mathbf{N}}_i z^i \tag{4.89}$$

Note that for a given system, the integers k and h, defining the numbers of terms in the left and right MFDs, are not necessarily equal. Recall that in the case of Algorithms (1) and (2), i.e. *DNRc* and *DNND*, the given left MFD $\{D(z), N(z)\}$ is not necessarily left coprime nor is $D(z)$ required to be monic and column-reduced. Similarly, i.e. dually, in the case of Algorithms (3) and (4), i.e. *NDRo* and *NDDN*, the given right MFD $\{\tilde{N}(z), \tilde{D}(z)\}$ is not necessarily right coprime, nor is $\tilde{D}(z)$ required to be monic and row-reduced.

For the purpose of Algorithms (1) and (2), i.e. *DNRc* and *DNND*, Eq.(4.88), should be represented by:

$$\left[\begin{array}{cccc|cccc}
\mathbf{D}_0 & & & & -\mathbf{N}_0 & & & \\
\mathbf{D}_1 & \mathbf{D}_0 & & & -\mathbf{N}_1 & -\mathbf{N}_0 & & \\
\vdots & & \ddots & & \vdots & & \ddots & \\
\mathbf{D}_k & & & \mathbf{D}_0 & -\mathbf{N}_k & & & -\mathbf{N}_0 \\
 & \mathbf{D}_k & & \mathbf{D}_1 & & -\mathbf{N}_k & & -\mathbf{N}_1 \\
 & & \ddots & \vdots & & & \ddots & \vdots \\
 & & & \mathbf{D}_k & & & & -\mathbf{N}_k
\end{array}\right]
\left[\begin{array}{c}
\tilde{\mathbf{N}}_0 \\ \vdots \\ \tilde{\mathbf{N}}_h \\ \hline \tilde{\mathbf{D}}_0 \\ \vdots \\ \tilde{\mathbf{D}}_h
\end{array}\right] = \mathbf{0} \tag{4.90}$$

while for Algorithms (3) and (4), i.e. *NDRo* and *NDDN*, Eq.(4.89), becomes:

$$\left[\, \mathbf{N}_0 \ \cdots \ \mathbf{N}_k \mid \mathbf{D}_0 \ \cdots \ \mathbf{D}_k \,\right] \left[\begin{array}{cccccc} \tilde{\mathbf{D}}_0 & \tilde{\mathbf{D}}_1 & \cdots & \tilde{\mathbf{D}}_h & & \\ & \ddots & \ddots & & \ddots & \\ & & \tilde{\mathbf{D}}_0 & \tilde{\mathbf{D}}_1 & \cdots & \tilde{\mathbf{D}}_h \\ \hline -\tilde{\mathbf{N}}_0 & -\tilde{\mathbf{N}}_1 & \cdots & -\tilde{\mathbf{N}}_h & & \\ & \ddots & \ddots & & \ddots & \\ & & -\tilde{\mathbf{N}}_0 & -\tilde{\mathbf{N}}_1 & \cdots & -\tilde{\mathbf{N}}_h \end{array}\right] = \mathbf{0} \tag{4.91}$$

Using the previously introduced notation, Eqs.(4.90) and (4.91) may be represented by:

$$\mathbf{T}_h \begin{bmatrix} \tilde{\mathbf{N}}_c \\ --- \\ \tilde{\mathbf{D}}_c \end{bmatrix} = \mathbf{0} \quad \text{and} \quad \left[\, \mathbf{N}_r \mid \mathbf{D}_r \,\right] \hat{\mathbf{T}}_k = \mathbf{0} \tag{4.92}$$

respectively, where in both cases matrices $\mathbf{T}_h$ and $\hat{\mathbf{T}}_k$ are build out of known submatrices, while matrices $\tilde{\mathbf{N}}_i$ and $\tilde{\mathbf{D}}_i$, $i=[0,h]$, entering in $\tilde{\mathbf{N}}_c$ and $\tilde{\mathbf{D}}_c$ in the case of Eq.(4.90), and $\mathbf{N}_j$ and $\mathbf{D}_j$, $j=[0,k]$, entering in $\mathbf{N}_r$ and $\mathbf{D}_r$ in the case of Eq.(4.91), are unknown. Recall that, not only are the matrices unknown, but the integers h and k should also be determined.

The reason for denoting the matrices in Eq.(4.92) by subscripts h and k is that in the $[p(k+h+1) \times (h+1)(p+m)]$ matrix $\mathbf{T}_h$ and the $[(p+m)(k+1) \times m(k+h+1)]$ matrix $\hat{\mathbf{T}}_k$ the integers h and k, respectively, are unknown. As will be shown later, the values of these integers are determined by building $\mathbf{T}_h$ and $\hat{\mathbf{T}}_k$ sequentially starting with $h = 1$ and $k = 1$, and ending with:

- $h = \max\{ n_i \}$, the row degrees of $\tilde{D}(z)$, or max $\{ \mu_i \}$ of a corresponding PCF R_c, and
- $k = \max\{ n_i \}$, the column degrees of $D(z)$, or max $\{ \nu_i \}$ of a corresponding POF R_o, respectively.

Note that Eqs.(4.90) and (4.91) are, respectively, similar to Eqs.(4.41) and (4.34), which are used for the transfer function conversion algorithms listed in Eq.(4.85). Comparing the expression of Eq.(4.90) with that of Eq.(4.41) and Eq.(4.91) with Eq.(4.34), the similarities and differences are easily determined.

It should be mentioned that the implementation of the four algorithms in Eq.(4.84) is similar to the algorithms listed in Eq.(4.85). For example, the second equation in Eqs.(4.92) is formally equal to Eq.(4.34) in *TFDN*. Thus here, it is also necessary to build $\hat{\mathbf{T}}_k$ and to determine the smallest integer k satisfying the previously given Eq.(4.35), i.e.:

$$\text{rank}\,[\,\hat{\mathbf{T}}_k\,] = (k+1)m + n$$

leading to

$$n = \text{rank}\,[\,\hat{\mathbf{T}}_k\,] - (k+1)m \tag{4.93}$$

which permits determination of the system order n and the value of the integer k. After having the values of n and k, an "admissible" set of row degrees $\{\, n_i \,\}$ satisfying:

$$\sum_{i=1}^{m} n_i = n \quad \text{and} \quad \max\{\, n_i \,\} = k \tag{4.94}$$

should be determined. Similarly, as in Eqs.(4.36) to (4.38), this could be done by checking the "auxiliary" selector vector $\hat{\mathbf{v}}_{lt}$, indicating linear independent rows in $\mathbf{T}_k$, which, of course, leads to the unique set, μ, of controllability indices of the corresponding controllability form R_c. If the admissibility degree, *C#*, corresponding to this set is too small, which happens sometimes, then, it is advisable to select an appropriate set $\{\, n_i \,\}$, satisfying Eq.(4.94) and check its degree *C#*.

Similar arguments hold for the first equation in Eqs.(4.92), i.e. for Algorithms *DNND* and *DNRc*. For additional details see the steps of the algorithms listed in Eq.(4.85). The reader is urged to examine the *L-A-S* implementation of these algorithms as well as the listings of all other algorithms discussed in this chapter.

4.5 Summary of Conversion Options

This chapter has presented the reader with a wide variety of numerically stable algorithms with which to convert from one model form to another. Since so many variations were covered, a brief summary is thought to be necessary. Table 4.1 below illustrates the large number of options that are accessible. All of the algorithms indicated in Table 4.1, except for those associated with system identification, i.e. conversion from input/output data to some model form, to be discussed in Chapter 5, have been presented in this chapter.

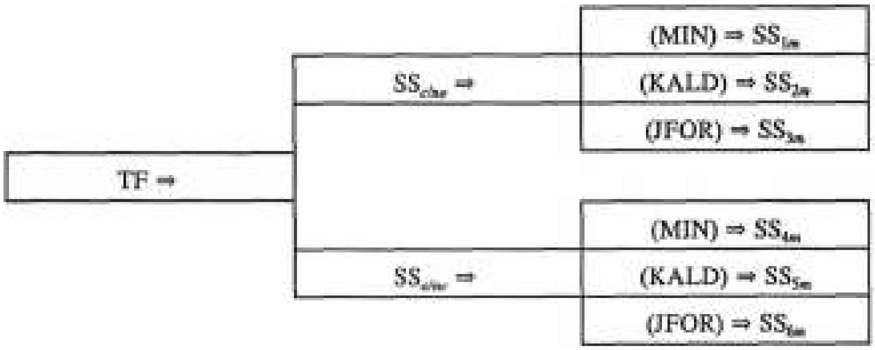
TABLE 4.1 Available Conversion Algorithms

	To:	1			2	3	4		5
From:		SS	R_o	R_c	TF	H	DN	ND	y
1	SS	*	SSRo	SSRo	SSTF	SSH			CDSR
	R_o	STR	*	SSRo	SSTF	SSH	RoDN		CDSR
	R_c	STR	SSRo	*	SSTF	SSH		RcND	CDSR
2	TF	(1)*	TFRo	TFRc	*	TFH	TFDN	TFND	CDTR
3	H	(2)*	HRo	HRc	HTF	*	HDN	HND	uHy
4	DN		DNRo	DNRc	DNTF	DNH*	*	DNND	
	ND		NDRo	NDRc	NDTF	NDH*	NDDN	*	
5	y		uyRo		uyTF	uyH*	uyDN		*

Comments:

(1) In addition to *TFRo* and *TFRc*, other available algorithms are "classical" minimal realization procedures such as Hessenberg's, Kalman decomposition and a Jordan form procedure.

Possible sequences of algorithms are given in the table below which represent conversions from a transfer function matrix to one of the state space minimal realizations. Of course, all of the resulting state space forms, SS_i, have the same transfer function matrix, i.e. SS_{im} (*SSTF*) ⇒ TF_i, where TF_i = TF for all i=[1,6].



The problems with these procedures are that they require a non-minimal state space representations which are in the case of MIMO system either of the order nm or np where n = order of characteristic polynomial $d(z)$, and m and p are the dimensions of the input and output vectors, respectively. Moreover, in the TF $\Rightarrow$ SS procedure we often "know" that the order of the minimal state space representation is equal to n. Our *TFRo* and *TFRc* procedures are less computationally "intensive" and are well "suited" for this intermodel conversion.

(2) In addition to *HRo* and *HRc* there are other "partial" realization algorithms which, using "several" Markov parameters $\mathbf{H}_i$, i=0,1,2,..., determine SS, for example, the Ho-Kalman, or ERA, procedure. The problem with these procedures are that they determine just a state space (SS) representation, without investigating whether there is another equivalent, computationally more convenient representation. Under the heading "computational convenience," we consider condition numbers of the similarity transformation matrices, i.e. admissibility degrees of those sets of POI and PCI used.

(3) The algorithms *DNH* and *NDH* are directly applicable only if the system is "D-T" stable, i.e. if all roots of $d(z)$ = det{ $D(z)$ } are within the unit circle. If not, then a "time scaling" of $D(z)$ and $N(z)$ should be performed first, and then the obtained time scaled Markov parameters $\mathbf{H}_i$ should be multiplied with f^i, where f is the "time scaling" factor.

The suggested sequence of algorithms is:

$D(z) \Rightarrow d(z) \Rightarrow$ maximum root of ($d(z) = 0$) $\Rightarrow$ time scaling factor f
$D(z)$, $N(z)$, $f \Rightarrow$ (time scaling) $\Rightarrow$ $D_s(z)$, $N_s(z)$
$D_s(z)$, $N_s(z)$ (*DNH*) $\Rightarrow$ truncated and time scaled $H_s(z^{-1})$
$\mathbf{H}_{si}$, $f_i \Rightarrow$ (time scaling "up") $\Rightarrow$ $\mathbf{H}_i$, i=[0,M-1]

Algorithms for time scaling of MFD's and Markov parameters are also available.

This is equally applicable to a left or right coprime MFD, as has been illustrated in the Example Section, see Section 4.6.

(4) The algorithm *uyH* is applicable only to stable D-T systems, i.e. to systems where $\|\mathbf{H}_{M-1}\| << 1$, for a sufficiently large finite M. Also all of these M Markov parameters $\mathbf{H}_i$, i=[0,M-1], should be determined.

4.6 Examples

In this section two comprehensive examples will be presented to illustrate the power and flexibility of the conversion process. The first example begins with a state space model of a 4th order, 3-input, 2-output C-T system and generates six different equivalent models. These six models are developed redundantly and cross-checked by 22 distinct conversions to show that the various models are compatible. These conversions are represented in Fig. 4.2 by arrows. In Example 2 additional conversions are presented which illustrate going from MFD models to Markov parameters when the system is not D-T stable, i.e. requiring scaling. Figures are presented with the examples to provide a graphical picture of the conversions.

4.6.1 Example 1 (Model Conversions)

We will first look at the system given in state space form and the admissible pseudo-controllability (PCI) and pseudo-observability (POI) indices:

$$R = \left[\begin{array}{cccc|ccc} -1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & -2 & 1 & 0 & .001 & 0 & 0 \\ 0 & -1 & -2 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & -2 & 0 & 0 & 1 \\ \hline 0 & .001 & 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

Assuming independent inputs and outputs, the system has three possible sets of controllability (PCI) indices and three possible sets of observability (POI) indices given in Tables 4.2 and 4.3:

TABLE 4.2 PCI

$\{n_{c1}\ n_{c2}\ n_{c3}\}$	rank	degree
{1 2 1}	4	.17E+00
{2 1 1}	4	.50E-03
{1 1 2}	3	.00E+00

TABLE 4.3 POI

$\{n_{o1}\ n_{o2}\}$	rank	degree
{1 3}	4	.70E-01
{2 2}	4	.20E-03
{3 1}	4	.35E-04

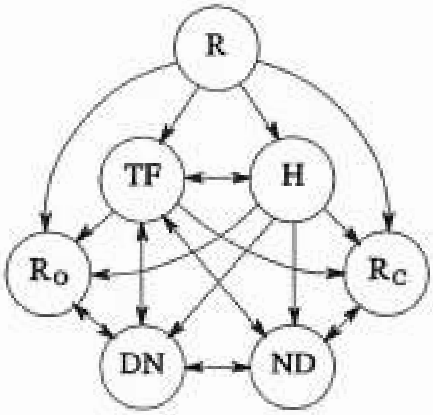


FIGURE 4.2 Conversions for Example 1

In Tables 4.2 and 4.3 "degree" refers to the *degree of admissibility*, the inverse of the condition number of the similarity transformation matrix **T** used in obtaining the corresponding PCF or POF. Note that the last set of PCI is *not* admissible since the rank of **T** is less than $n = 4$. The best selection is associated with the highest degree of admissibility, which is the first set in each case. Note that the best selections in both cases are different from the unique controllability and observability indices, which are {2,1,1} and {2,2}, respectively.

In the following development, which illustrates intermodel conversions between SS , TF , H and MFD models, the same model may be repeatedly generated by different methods. In all instances the models agree closely.

Sequence of algorithm executions:

R (*SSRo*) ⇒ Ro
R (*SSRc*) ⇒ Rc
R (*SSTF*) ⇒ TF
R (*SSH*) ⇒ H
TF (*TFH*) ⇒ H1
H (*HRo*) ⇒ Ro1
H (*HRc*) ⇒ Rc1
Ro (*RoDN*) ⇒ DN
Rc (*RcND*) ⇒ ND
DN (*DNTF*) ⇒ TF1
ND (*NDTF*) ⇒ TF2

H (*HDN*) ⇒ DN1
H (*HND*) ⇒ ND1
DN (*DNRo*) ⇒ Ro2
ND (*NDRc*) ⇒ Rc2
DN (*DNND*) ⇒ ND2
ND (*NDDN*) ⇒ DN2
TF (*TFND*) ⇒ ND3
TF (*TFDN*) ⇒ DN3
TF (*TRRo*) ⇒ Ro3
TF (*TFRc*) ⇒ Rc3
H (*HTF*) ⇒ TF3

Multiply-Generated Models:

Transfer function matrices: TF , TF1 , TF2 , TF3
State Space representations in POF: Ro , Ro1 , Ro2 , Ro3
State Space Representations in PCF: Rc , Rc1 , Rc2 , Rc3
Markov Parameters in: H , H1
Left coprime MFD in: DN , DN1 , DN2 , DN3
Right coprime MFD in: ND , ND1 , ND2 , ND3

Since the eigenvalues of **A** in R are not within unit circle, the algorithms *DNH* and *NDH* were not used. The use of these two algorithms will be illustrated in Example 2.

Results:

Using the "best" sets of structural indices from Tables 4.2 and 4.3, the following observable and controllable state space models were calculated:

Observable Form (ν = {1,3})

$$R_o = \left[\begin{array}{cccc|ccc} -2.0 & .002 & .003 & .001 & 0.0 & 0.0 & 1.0 \\ 0 & 0 & 1 & 0 & 1.0 & 0.0 & 0.0 \\ 0 & 0 & 0 & 1 & -.999 & 0.0 & 0.0 \\ 1.0 & -5.001 & -9.001 & -5.0 & .997 & 1.0 & 0.0 \\ \hline 1 & 0 & 0 & 0 & 0.0 & 1.0 & 0.0 \\ 0 & 1 & 0 & 0 & 0.0 & 0.0 & 0.0 \end{array}\right]$$

Controllable Form (μ = {1,2,1})

$$R_c = \left[\begin{array}{cccc|ccc} -.999 & 0 & 0.0 & 1.0 & 1 & 0 & 0 \\ -.003 & 0 & 1.0 & -5.002 & 0 & 1 & 0 \\ 0.0 & 0 & -2.0 & 0.0 & 0 & 0 & 1 \\ -.001 & 1 & 0.0 & -4.001 & 0 & 0 & 0 \\ \hline 0.0 & 0.0 & 1.0 & .001 & 0.0 & 1.0 & 0.0 \\ 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{array}\right]$$

Transfer Function Matrix $G(s) = \dfrac{\mathbf{W}(s)}{d(s)}$

where
$$\mathbf{W}(s) = \begin{bmatrix} w_{11}(s) & w_{12}(s) & w_{13}(s) \\ w_{21}(s) & w_{22}(s) & w_{23}(s) \end{bmatrix}$$

$$\begin{array}{c|ccccc|}
 & s^0 & s^1 & s^2 & s^3 & s^4 \\
w_{11} & 0 & 0 & 0 & 0 & 0 \\
w_{21} & 10.004 & 13.004 & 6.001 & 1 & 0 \\
\hline
w_{12} & 10.002 & 23.003 & 19.001 & 7.0 & 1.0 \\
w_{22} & 2.0 & 1.0 & 0 & 0 & 0 \\
\hline
w_{13} & 5.001 & 9.001 & 5.0 & 1.0 & 0 \\
w_{23} & 1.0 & 0 & 0 & 0 & 0
\end{array}$$

For example, $\quad w_{12}(s) = 10 + 23s + 19s^2 + 7s^3 + s^4$

which happens to be equal to the characteristic polynomial $d(s)$.

Markov Parameters $\mathbf{H}_i$

The first few terms of the 3-column polynomial matrix $H(s^{-1}) = \{h_{ij}(s^{-1})\}$ are:

$$\begin{array}{c|cccccccccc|}
 & s^0 & s^{-1} & s^{-2} & s^{-3} & s^{-4} & s^{-5} & s^{-6} & s^{-7} & s^{-8} & s^{-9} \\
h_{11} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
h_{21} & 0 & 1 & -1 & 1 & -1 & 1 & -1 & 1 & -1.2 & 1.4 \\
\hline
h_{12} & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0.3 \\
h_{22} & 0 & 0 & 0 & 1 & -5 & 16 & -40 & 81 & -125 & 96 \\
\hline
h_{13} & 0 & 1 & -2 & 4 & -8 & 16 & -32.1 & 64.2 & -128.4 & 257.8 \\
h_{23} & 0 & 0 & 0 & 0 & 1 & -7 & 30 & -100 & 281 & -687
\end{array}$$

Left Coprime MFD

This form is given by $D^{-1}(s)N(s)$ where $D(s)$ is monic and column-reduced, i.e.:

$$D_l(s) = \begin{bmatrix} d_{11}(s) & d_{12}(s) \\ d_{21}(s) & d_{22}(s) \end{bmatrix}$$

$$N_l(s) = \begin{bmatrix} n_{11}(s) & n_{12}(s) & n_{13}(s) \\ n_{21}(s) & n_{22}(s) & n_{23}(s) \end{bmatrix}$$

denoted by

$$\begin{array}{c|cccc|} & s^0 & s^1 & s^2 & s^3 \\ d_{11} & 2.0 & 1.0 & 0 & 0 \\ d_{21} & -1.0 & 0.0 & 0.0 & 0.0 \\ \hdashline d_{12} & -.002 & -.003 & -.001 & 0.0 \\ d_{22} & 5.001 & 9.001 & 5.0 & 1.0 \end{array}$$

and

$$\begin{array}{c|ccc|} & s^0 & s^1 & s^2 \\ n_{11} & -.002 & -.001 & 0 \\ n_{21} & 5.003 & 4.001 & 1 \\ \hdashline n_{12} & 2.0 & 1.0 & 0.0 \\ n_{22} & 0.0 & 0.0 & 0.0 \\ \hdashline n_{13} & 1.0 & 0.0 & 0.0 \\ n_{23} & 0.0 & 0.0 & 0.0 \end{array}$$

Right Coprime MFD

This form is given by $N(s)D^{-1}(s)$ where $D(s)$ is monic and row-reduced, i.e.:

$$D_r(s) = \begin{bmatrix} d_{11}(s) & d_{12}(s) & d_{13}(s) \\ d_{21}(s) & d_{22}(s) & d_{23}(s) \\ d_{31}(s) & d_{32}(s) & d_{33}(s) \end{bmatrix}$$

$$N_r(s) = \begin{bmatrix} n_{11}(s) & n_{12}(s) & n_{13}(s) \\ n_{21}(s) & n_{22}(s) & n_{23}(s) \end{bmatrix}$$

denoted by

$$\begin{array}{c|ccc|} & s^0 & s^1 & s^2 \\ d_{11} & .999 & 1.0 & 0 \\ d_{21} & .003 & .001 & 0.0 \\ d_{31} & 0.0 & 0.0 & 0.0 \\ \hdashline d_{12} & 0.0 & 0.0 & 0.0 \\ d_{22} & -1.0 & 0.0 & 0.0 \\ d_{32} & 2.0 & 1.0 & 0.0 \\ \hdashline d_{13} & -1.0 & 0.0 & 0.0 \\ d_{23} & 5.002 & 4.001 & 1.0 \\ d_{33} & 0.0 & 0.0 & 0.0 \end{array}$$

and

$$\begin{array}{c|ccc|} & s^0 & s^1 & s^2 \\ n_{11} & .003 & .001 & 0 \\ n_{21} & 1.0 & 0.0 & 0.0 \\ \hdashline n_{12} & 0.0 & 0.0 & 0.0 \\ n_{22} & 0.0 & 0.0 & 0.0 \\ \hdashline n_{13} & 5.003 & 4.001 & 1.0 \\ n_{23} & 0.0 & 0.0 & 0.0 \end{array}$$

To compare $D_l(s)$ and $D_r(s)$, we can write out the polynomial matrices in their common form from the information above, rounding to integer values for convenience:

$$D_l(s) = \begin{bmatrix} 2+s & 0 \\ -1 & 5+9s+5s^2+s^3 \end{bmatrix}$$

$$D_r(s) = \begin{bmatrix} 1+s & 0 & -1 \\ 0 & -1 & 5+4s+s^2 \\ 0 & 2+s & 0 \end{bmatrix}$$

Note that the *column* degrees of $D_l(s)$ are {1, 3} corresponding to the set ν, and that the *row* degrees of $D_r(s)$ are {1, 2, 1} corresponding to the set μ.

Figure 4.2 indicates the following (22) conversions:

R $\Rightarrow$ Ro , Rc , TF , H
TF $\Rightarrow$ Ro , Rc , DN , ND , H
H $\Rightarrow$ Ro , Rc , TF , DN , ND
DN $\Rightarrow$ Ro , TF , ND
ND $\Rightarrow$ Rc , TF , DN
Ro $\Rightarrow$ DN
Rc $\Rightarrow$ ND

The repeated models are essentially the same and serve as checks on the others.

4.6.2 Example 2 (Time Scaling)

This example illustrates the conversion between MFD and the Markov parameters when the system is not D-T stable. The given system is the same one used in Example 1. Only additional results are presented for brevity. Figure 4.3 illustrates the 13 distinct conversions made.

Sequence of algorithm executions:

R (*SSRo*) $\Rightarrow$ Ro
R (*SSRc*) $\Rightarrow$ Rc
R (*SSH*) $\Rightarrow$ H
Ro (*RoDN*) $\Rightarrow$ DN
Rc (*RcND*) $\Rightarrow$ ND
DN (*DNts*) $\Rightarrow$ DNs
ND (*NDts*) $\Rightarrow$ NDs

DNs (*DNH*) $\Rightarrow$ Hs1
NDs (*NDH*) $\Rightarrow$ Hs2
Hs1 (*Hts*) $\Rightarrow$ H1
Hs2 (*Hts*) $\Rightarrow$ H2
H2 (*HRo*) $\Rightarrow$ Ro1
H1 (*HRc*) $\Rightarrow$ Rc1

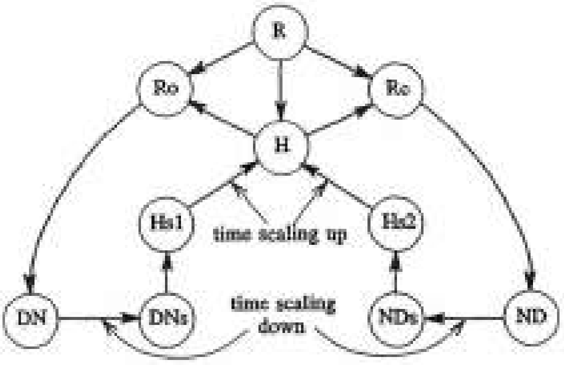


FIGURE 4.3 Conversions for Example 2

This example presents the following generated models:

State space representations in POF: Ro , Ro1
State space representations in PCF: Rc , Rc1
Left and right coprime MFD: DN , ND
Time-scaled left and right coprime MFD: DNs , NDs
Markov parameters in: H , H1 , H2
"Time scaled" Markov parameters in: Hs1 , Hs2

The original model R, as well as the derived models R_o, R_c, $G(s)$ and $H(s^{-1})$ are identical to those given in Example 1. Therefore, only an extension of $H(s^{-1})$ will be given here, since the number of "useful" terms is larger than presented in the results of Example 1:

$$
\begin{array}{c|cccccccccc|}
 & s^{0} & s^{-1} & \cdots & s^{-9} & s^{-10} & s^{-11} & s^{-12} & s^{-13} & s^{-14} \\
h_{11} & 0 & 0 & \cdots & 0 & 0 & 0 & 0 & 0 & 0 \\
h_{21} & 0 & 1 & \cdots & 1.4 & -2.0 & 2.7 & -2.9 & 0.2 & 11.4 \\
\hdashline
h_{12} & 1 & 0 & \cdots & 0.3 & -1.2 & 3.1 & -6.5 & 10.2 & -8.8 \\
h_{22} & 0 & 0 & \cdots & 96.0 & 240 & -1439 & 4555 & -11024 & 21320 \\
\hdashline
h_{13} & 0 & 1 & \cdots & 257.8 & -513.2 & 1025 & -2047 & 4088 & -8166 \\
h_{23} & 0 & 0 & \cdots & -687.0 & 1470 & -2700 & 3961 & -3367 & -4290
\end{array}
$$

In order to illustrate *scaling* in MFD, we will repeat some of the results from Example 1. First, let us compare the unscaled and scaled left coprime MFD:

Left Coprime MFD (unscaled)

This form is given by $D^{1}(s)N(s)$ where $D(s)$ is monic and column-reduced. These results are taken from Example 1:

$$D_l(s) = \begin{bmatrix} d_{11}(s) & d_{12}(s) \\ d_{21}(s) & d_{22}(s) \end{bmatrix}$$

$$N_l(s) = \begin{bmatrix} n_{11}(s) & n_{12}(s) & n_{13}(s) \\ n_{21}(s) & n_{22}(s) & n_{23}(s) \end{bmatrix}$$

denoted by

$$\begin{array}{c|cccc|} & s^0 & s^1 & s^2 & s^3 \\ d_{11} & 2.0 & 1.0 & 0 & 0 \\ d_{21} & -1.0 & 0.0 & 0.0 & 0.0 \\ \hline d_{12} & -.002 & -.003 & -.001 & 0.0 \\ d_{22} & 5.001 & 9.001 & 5.0 & 1.0 \end{array} = D_l(s)$$

and

$$\begin{array}{c|ccc|} & s^0 & s^1 & s^2 \\ n_{11} & -.002 & -.001 & 0 \\ n_{21} & 5.003 & 4.001 & 1 \\ \hline n_{12} & 2.0 & 1.0 & 0.0 \\ n_{22} & 0.0 & 0.0 & 0.0 \\ \hline n_{13} & 1.0 & 0.0 & 0.0 \\ n_{23} & 0.0 & 0.0 & 0.0 \end{array} = N_l(s)$$

The scaling factor f is set at a value of 0.1. This "time scaling" is reflected into the s-domain as a scaling factor on the s variable. A well known fact from Fourier theory is that a contraction of the time axis corresponds to an expansion of the frequency axis and vice versa. Here, because the system response becomes large too quickly in normal time, we expand the time axis by a factor $f=10$. Thus, if p

represents the scaled Laplace variable, then $s = 10p$, and e.g., a polynomial

$$d(s) = 1 + 5s^{-1} + 9s^{-2}$$

becomes

$$d(p) = 1 + 0.5p^{-1} + 0.09p^{-2}$$

The reader should compare the following scaled version with the above left coprime MFD:

Left Coprime MFD (scaled)

$$\begin{array}{c} \\ d_{11} \\ d_{21} \\ \\ d_{12} \\ d_{22} \end{array} \left[\begin{array}{cccc} s^0 & s^1 & s^2 & s^3 \\ 0.2 & 1.0 & 0.0 & 0.0 \\ -.001 & 0.0 & 0.0 & 0.0 \\ \hdashline 0.0 & -.003 & -.01 & 0.0 \\ .005 & .090 & 0.5 & 1.0 \end{array}\right] = D_{ls}(s)$$

and

$$\begin{array}{c} \\ n_{11} \\ n_{21} \\ \\ n_{12} \\ n_{22} \\ \\ n_{13} \\ n_{23} \end{array} \left[\begin{array}{ccc} s^0 & s^1 & s^2 \\ 0.0 & -.001 & 0.0 \\ .005 & .040 & 0.1 \\ \hdashline 0.2 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \\ \hdashline 0.1 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 \end{array}\right] = N_{ls}(s)$$

Right Coprime MFD (unscaled)

This form is given by $N(s)D^{-1}(s)$ where $D(s)$ is monic and row-reduced. This is the same scaling that was previously applied to the left coprime MFD above. The unscaled results are repeated from Example 1:

$$D_r(s) = \begin{bmatrix} d_{11}(s) & d_{12}(s) & d_{13}(s) \\ d_{21}(s) & d_{22}(s) & d_{23}(s) \\ d_{31}(s) & d_{32}(s) & d_{33}(s) \end{bmatrix}$$

$$N_r(s) = \begin{bmatrix} n_{11}(s) & n_{12}(s) & n_{13}(s) \\ n_{21}(s) & n_{22}(s) & n_{23}(s) \end{bmatrix}$$

denoted by

$$\begin{array}{c|ccc|}
 & s^0 & s^1 & s^2 \\
d_{11} & .999 & 1.0 & 0 \\
d_{21} & .003 & .001 & 0.0 \\
d_{31} & 0.0 & 0.0 & 0.0 \\
 & \text{-----} & \text{-----} & \text{-----} \\
d_{12} & 0.0 & 0.0 & 0.0 \\
d_{22} & -1.0 & 0.0 & 0.0 \\
d_{32} & 2.0 & 1.0 & 0.0 \\
 & \text{----} & \text{----} & \text{----} \\
d_{13} & -1.0 & 0.0 & 0.0 \\
d_{23} & 5.002 & 4.001 & 1.0 \\
d_{33} & 0.0 & 0.0 & 0.0
\end{array} = D_r(s)$$

and

$$\begin{array}{c|ccc|}
 & s^0 & s^1 & s^2 \\
n_{11} & .003 & .001 & 0 \\
n_{21} & 1.0 & 0.0 & 0.0 \\
 & \text{-----} & \text{-----} & \text{-----} \\
n_{12} & 0.0 & 0.0 & 0.0 \\
n_{22} & 0.0 & 0.0 & 0.0 \\
 & \text{-----} & \text{-----} & \text{-----} \\
n_{13} & 5.003 & 4.001 & 1.0 \\
n_{23} & 0.0 & 0.0 & 0.0
\end{array} = N_r(s)$$

Right Coprime MFD (scaled)

$$\begin{array}{c|ccc|}
 & s^0 & s^1 & s^2 \\
d_{11} & 0.1 & 1.0 & 0.0 \\
d_{21} & 0.0 & .001 & 0.0 \\
d_{31} & 0.0 & 0.0 & 0.0 \\
\hline
d_{12} & 0.0 & 0.0 & 0.0 \\
d_{22} & -0.1 & 0.0 & 0.0 \\
d_{32} & 0.2 & 1.0 & 0.0 \\
\hline
d_{13} & -.01 & 0.0 & 0.0 \\
d_{23} & .05 & 0.4 & 1.0 \\
d_{33} & 0.0 & 0.0 & 0.0
\end{array} = D_{rs}(s)$$

and

$$\begin{array}{c|ccc|}
 & s^0 & s^1 & s^2 \\
n_{11} & 0.0 & .001 & 0.0 \\
n_{21} & 0.1 & 0.0 & 0.0 \\
\hline
n_{12} & 0.0 & 0.0 & 0.0 \\
n_{22} & 0.0 & 0.0 & 0.0 \\
\hline
n_{13} & .05 & 0.4 & 1.0 \\
n_{23} & 0.0 & 0.0 & 0.0
\end{array} = N_{rs}(s)$$

Markov Parameters (scaled)

The first few terms of the 3-column polynomial matrix $H(s^{-1}) = \{h_{ij}(s^{-1})\}$ were given in Example 1. That result illustrated the increasing values of the parameters. A further extension was given earlier in this example. Now consider the *scaled*

Markov parameters (again with $f = 0.1$), and note the dramatic change:

$$\begin{array}{c|ccccccccc|}
 & s^{0} & s^{-1} & s^{-2} & s^{-3} & s^{-4} & s^{-5} & s^{-6} & s^{-7} & s^{-8} \\
h_{11} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
h_{21} & 0 & .1 & -.01 & .001 & -.0001 & 10^{-5} & 0.0 & 0.0 & 0.0 \\
\hline
h_{12} & 1.0 & 0.0 & 10^{-5} & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
h_{22} & 0.0 & 0.0 & 0.0 & .001 & -.0005 & .00016 & -.00004 & 10^{-5} & 0.0 \\
\hline
h_{13} & 0.0 & 0.1 & -.002 & .004 & -.0008 & .00016 & -.00003 & 10^{-5} & 0.0 \\
h_{23} & 0.0 & 0.0 & 0.0 & 0.0 & .0001 & -.00007 & .00003 & -10^{-5} & 0.0
\end{array}$$

These Markov parameters are obtained by Algorithms *DNH* and *NDH* which require D-T stable MFDs.

Explanation of time scaling of MFDs, time scaling with the factor $f = 0.1$:

Original left and right coprime MFDs (without scaling):

$$\boldsymbol{D}_l(s) = \begin{bmatrix} 2+1s & -.002-.003s-.001s^2 \\ -1 & 5.001+9.001s+5s^2+1s^3 \end{bmatrix}$$

$$\boldsymbol{D}_r(s) = \begin{bmatrix} .999+1s & 0 & -1 \\ .003+.001s & -1 & 5.002+4.001s+1s^2 \\ 0 & 2+1s & 0 \end{bmatrix}$$

Left and right coprime MFDs with scaling:

$$\begin{array}{cc} \boldsymbol{D}_l(s) = & \begin{bmatrix} .2+1s & -.0002-.003s-.01s^2 \\ -.001 & .005001+.09001s+.5s^2+1s^3 \end{bmatrix} \\ & \begin{array}{ccc} 1 & \qquad 3 & \qquad \text{column degrees} \end{array} \end{array}$$

$$\boldsymbol{D}_r(s) = \begin{bmatrix} .0999+1s & 0 & -.01 \\ .0003+.001s & -0.1 & .05002+.4001s+1s^2 \\ 0 & 0.2+1s & 0 \end{bmatrix} \begin{array}{c} 1 \\ 2 \\ 1 \end{array} \ \begin{array}{c} \text{row} \\ \text{degrees} \end{array}$$

The coefficients d_{ijh} of $D_l(s)$ in an i^{th} row where the polynomial with the column degree n_j is located are multiplied with f^{n_j-h}, $h=[0,n_m]$, $n_m = \max\{ n_j \}$. Thus, the highest degree coefficients in all polynomials defining the column degrees are unchanged and the time scaled polynomial matrix $D_{lt}(s)$ remains monic.

Similarly, the coefficients d_{ijh} of $D_r(s)$ in a j^{th} column where the polynomial with the row degree n_i is located are multiplied with f^{n_i-h}, $h=[0,n_m]$, $n_m = \max\{ n_j \}$. Thus, the highest degree coefficients in all polynomials defining the row degrees are unchanged and the time scaled polynomial matrix $D_{rt}(s)$ remains monic. Note the verification of these statements in the example above.

4.7 Summary

To summarize the developments in this chapter, a large collection of algorithms was presented. These algorithms provide conversions between any two types of system models: state space canonical forms, transfer function matrices and matrix fraction descriptions (ARMA models), as well as the Markov parameter (Hankel matrix) description. Appropriate application of this group of algorithms will allow the designer to view the system from every "perspective," and to work with the most convenient model.

In conclusion of this chapter it should be pointed out that the pseudo-controllable and pseudo-observable forms, PCF and POF, used in the majority of intermodel conversions, have not so far been widely used in the systems/controls literature. The reason for that is, no doubt, due to the great popularity of Luenberger canonical forms and the fact that PCFs and POFs used here are just "permutations" of Luenberger forms. Recall that in Chapter 3 it was stated that our versions of PCFs and POFs, based on admissible sets of pseudo-controllability and pseudo-observability indices, POI and POI, are more "natural" than other approaches in representing MIMO systems. This "naturalness" stems from the extremely simple relationship between the state space and input/output, i.e. MFD, models and the conclusion that there is a one-to-one correspondence between state space and MFD models, which has been established by Remarks 4.1 and 4.2 and illustrated by Examples 4.1 and 4.2.

To emphasize these simple, important and straightforward relationships let us review them once more in a slightly different way. Consider a state space representation:

$$R_v = \{\mathbf{A}_o, \mathbf{B}_o, \mathbf{C}_o, \mathbf{D}_o\} \text{ in a POF}$$

based on an admissible set of POI:

$$\mathbf{v} = \{ v_i \}, \quad i=[1,p], \quad k=\max\{ v_i \}$$

and its one-to-one "counterpart," i.e. a left coprime MFD:

$$\{D(z),\, N(z)\}$$

where $D(z)$ is monic and column-reduced having column degrees $\{ n_i \}$ equal to POI, i.e.:

$$\{ n_i \} = \{ \nu_i \}$$

Of course, these two models are related to each other by:

$$\mathbf{C}_a(z\mathbf{I} - \mathbf{A}_a)^{-1}\mathbf{B}_a + \mathbf{D}_a = D^{-1}(z)\, N(z)$$

In Section 4.1.7 it was established that the non-zero, non-unity elements in $\mathbf{A}_a$ and the negatives of the corresponding non-zero, non-unity coefficients d_{ijk} of the polynomials in $D(z)$ are equal to each other. Also, having these elements, it is extremely easy to build either $\mathbf{A}_a$ or $D(z)$, since the locations of those elements in both $\mathbf{A}_a$ and $D(z)$ are uniquely determined by the selector vectors $\mathbf{v}_a$, $\mathbf{v}_i$, $\mathbf{v}_{ii}$ and $\mathbf{v}_{ia}$ (and associated selector matrices) generated by the underlying set $\{ n_i \}$ or $\{ \nu_i \}$. In Section 3.3.4 it was shown that the selector vectors are a simple consequence of (or easily obtainable from) the crate diagram based on the set POI.

It has been shown that the total number of above mentioned non-zero, non-unity elements is np, and that they appear in $\mathbf{A}_a$ in some p rows, while in $D(z)$ they appear in some n columns in the $[p \times (k+1)p]$ matrix $\mathbf{D}_r$, where $\mathbf{D}_r$ is related to $D(z)$ by:

$$\mathbf{D}_r = \left[\mathbf{D}_0 \mid \mathbf{D}_1 \mid \cdots\; \mathbf{D}_k\right] \quad \text{and} \quad D(z) = \sum_{i=0}^{k} \mathbf{D}_i\, z^i$$

If the p rows from $\mathbf{A}_a$, containing non-zero, non-unity elements, are arranged in the $(p \times n)$ matrix $\mathbf{A}_r$, and if the n columns from the matrix $\mathbf{D}_r$ are arranged into another $(p \times n)$ matrix, say $\mathbf{D}_{rr}$, then it has been shown that:

$$\mathbf{D}_{rr} = -\mathbf{A}_r$$

It has also been established that the locations of the non-zero, non-unity rows in $\mathbf{A}_a$ are determined by the locations of unities in the selector vector $\mathbf{v}_a$, i.e. the selector matrix $\mathbf{S}_a$, and that the locations of the non-zero, non-unity columns in $\mathbf{D}_r$ are determined by the location of unities in the selector vector $\mathbf{v}_{ii}$, i.e. the selector matrix $\mathbf{S}_{ii}$.

Thus, as has been shown previously, and used effectively in a number of algorithms in this chapter, the relationship between $\mathbf{A}_a$ and $\mathbf{D}_r$ may be expressed as:

$$\mathbf{A}_r = \mathbf{S}_a^T \mathbf{A}_a = -\mathbf{D}_r \mathbf{S}_{ii} \tag{4.95}$$

which, in fact, may be considered as the basis of almost all of the previously described algorithms.

With the help of two other selector vectors, namely, $\mathbf{v}_i$ and $\mathbf{v}_{id}$ (and selector matrices $\mathbf{S}_i$ and $\mathbf{S}_{id}$) we may:

- Given $\mathbf{A}_o$, calculate $\mathbf{D}_r$ by:

$$\mathbf{D}_r = \mathbf{S}_{id}^T - \mathbf{S}_a^T \mathbf{A}_o \mathbf{S}_{ii}^T$$

or conversely,

- Given $D(z)$, i.e. $\mathbf{D}_r$, then $\mathbf{A}_o$ may be obtained simply by:

$$\mathbf{A}_o = \mathbf{S}_i \mathbf{A}_2 - \mathbf{S}_a \mathbf{D}_r \mathbf{S}_{ii}$$

where $\mathbf{A}_2 = [\ \mathbf{0} \mid \mathbf{I}_{n-p}\ ]$, while the selector vectors and selector matrices used are defined in Section 3.3.4.

Applying the principle of duality, it may be shown that the relationship between $\mathbf{A}_c$ in a PCF:

$$R_c = \{\mathbf{A}_c, \mathbf{B}_c, \mathbf{C}_c, \mathbf{D}_c\}$$

based on a set of PCI

$$\{\ \mu_i\ \},\ k = \max\ \{\ \mu_i\ \},\ i=[1,m]$$

and its one-to-one counterpart, monic and row-reduced $D(z)$ in a right coprime MFD

$$\{N(z),\ D(z)\}$$

having row degrees $\{\ n_i\ \}$ equal to the PCI, i.e.:

$$\{\ n_i\ \} = \{\ \mu_i\ \}$$

could be expressed by:

$$\mathbf{D}_{cc} = \mathbf{S}_{id} - \mathbf{S}_{ii}\ \mathbf{A}_c\ \mathbf{S}_a$$

or

$$\mathbf{A}_c = \mathbf{A}_2\ \mathbf{S}_i^T - \mathbf{S}_{ii}^T\ \mathbf{D}_{cc}\ \mathbf{S}_a^T$$

where, now:

$$\mathbf{A}_2 = \begin{bmatrix} \mathbf{0} \\ \text{----} \\ \mathbf{I}_{n-m} \end{bmatrix}, \quad D(z) = \sum_{i=0}^{k} \mathbf{D}_i z^i\ , \quad \mathbf{D}_{cc} = \begin{bmatrix} \mathbf{D}_0 \\ \mathbf{D}_1 \\ \vdots \\ \mathbf{D}_k \end{bmatrix}$$

and of course:

$$\mathbf{C}_c(z\mathbf{I} - \mathbf{A}_c)^{-1}\mathbf{B}_c + \mathbf{D}_c = N(z)\ D^{-1}(z)$$

4.8 References

In addition to the standard linear system books of Brogan (1991), Chen (1984) and Kailath (1980), a classical paper concerning system identification is Ho and Kalman (1965). A useful modification of Ho and Kalman's work is reported in Bingulac *et al* (1990). Both Rosenbrock (1970) and Wolovich (1974) provide alternative discussions on the relationships between the MFD and state space models. See Ackermann (1985), Appendix A, for background on Hessenberg forms. Bingulac and Djorovic (1975) and Bingulac and Sinha (1990) are recommended reading for specific discussions on Jordan forms and C-T system identification, respectively.

Ackermann, J. (1985), *Sampled-Data Control Systems*, Springer Verlag, Berlin.

Bingulac, S. and M. Djorovic (1975), "On the minimality of the Jordan form state equations," *IEEE Trans. on Automatic Control*, **AC-20**, 5, 687-789.

Bingulac, S., B. Gorti and H.F. VanLandingham (1990), "Computational simplification in the ERA (Eigensystem Realization Algorithm)," *Proceedings of the 28th Allerton Conference*, October 3-5, 1990, University of Illinois, pp. 529-530.

Bingulac, S. and W. Luse (1989), "Calculation of generalized eigenvectors," *Journal on Computers and Electrical Engineering*, **15**, 1, 29-32.

Bingulac, S. and N.K. Sinha (1990), "On the identification of continuous-time multivariable systems from samples of input-output data," *Math. Comput. Modelling*, **14**, pp. 203-208.

Brogan, W.L. (1991), *Modern Control Theory, 3rd Edition*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Chen, C-T. (1984), *Linear System Theory and Design*, Holt, Rinehart and Winston, Inc., New York, NY.

Ho, B.L. and R.E. Kalman (1966), "Effective construction of linear state-variable models from input/output data," *Proceedings of the 3rd Annual Allerton Conference on Circuit and System Theory*, University of Illinois, pp. 449-459.

Kailath, T. (1980), *Linear Systems*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Rosenbrock, H.H. (1970), *State Space and Multivariable Theory*, John Wiley and Sons, Inc., New York, NY.

Wolovitch, W.A. (1974), *Linear Multivariable Systems*, Springer Verlag, Berlin.

4.9 Exercises

4.1 The first 8 Markov parameters, $\mathbf{H}_i$, $i=[0,7]$, of a system are given below:

$$\{\mathbf{H}_i\} = \left\{ \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} -1 & 1 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 1 \\ -1 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix}, \begin{bmatrix} 1 & 4 \\ 1 & 1 \end{bmatrix}, \begin{bmatrix} 2 & 7 \\ 1 & 2 \end{bmatrix}, \begin{bmatrix} 4 & 12 \\ 1 & 4 \end{bmatrix} \right\}$$

These Markov parameters arranged in the PMF, $H(s^{-1})$, are:

$$\begin{bmatrix} 0 & -1 & 0 & 1 & 1 & 1 & 2 & 4 \\ 0 & 0 & 0 & -1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 2 & 4 & 7 & 12 \\ 0 & 0 & 0 & 1 & 1 & 1 & 2 & 4 \end{bmatrix}$$

Determine:

(a) —all admissible POFs and the corresponding sets of POI; use Algorithm *HRo*,
(b) —which set(s) of POI are not admissible,
(c) —all admissible PCFs and the corresponding sets of PCI; use Algorithm *HRc*,
(d) —which set(s) of PCI are not admissible,
(e) —all admissible left coprime MFDs and the corresponding sets of column degrees; use Algorithm *HDN*,
(f) —all admissible right coprime MFDs and the corresponding sets of row degrees; use Algorithm *HND*.

A version of an *L-A-S* program which solves this exercise is available in the *L-A-S* subdirectory C:\LAS\DPF\EXER41.DPF.

4.2 Given below is a right coprime MFD; $D(s)$ is not row-reduced:

$$N(s) = \begin{bmatrix} 0 & -s^2 \\ 0 & -1 \end{bmatrix}, \quad D(s) = \begin{bmatrix} -1+s & 1-s^2+s^3 \\ -2+s & 2-s^2 \end{bmatrix}$$

The matrices above can be written in PMF as follows:

$$\mathbf{N} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ -1 & 0 & 0 & 0 \end{bmatrix}, \quad \mathbf{D} = \begin{bmatrix} -1 & 1 & 0 & 0 \\ -2 & 1 & 0 & 0 \\ 1 & 0 & -1 & 1 \\ 2 & 0 & -1 & 0 \end{bmatrix}$$

Determine:

(a) —all admissible left coprime MFDs, $\{D_l(s), N_l(s)\}$, with $D_l(s)$ column-reduced; use Algorithm *NDDN*,

(b) —all admissible POFs, R_o, and corresponding sets of POI; use Algorithm *NDRo*.

(c) Using one of the obtained admissible left coprime MFDs in (a), calculate all admissible right coprime MFDs, $\{N_r(s), D_r(s)\}$, with $D_r(s)$ row-reduced; use Algorithm *DNND*.

(d) Using one of the obtained admissible left coprime MFDs in (a), calculate all admissible PCFs, R_c, and corresponding sets of PCI; use Algorithm *DNRc*.

A version of an *L-A-S* program which solves this exercise is available in the *L-A-S* subdirectory C:\LAS\DPF\EXER42.DPF.

4.3 Given the transfer function matrix $G(s) = W(s)/d(s)$ where:

$$W(s) = \begin{bmatrix} s & s(1+s)^2 \\ -s(1+s)^2 & -s(1+s)^2 \end{bmatrix}$$

and

$$d(s) = (1+s)^2(2+s)^2$$

W in PMF and the coefficients of $d(s)$ are:

$$\mathbf{W} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & -1 & -2 & -1 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & -1 & -2 & -1 & 0 \end{bmatrix}$$
$$\mathbf{d} = [\,4 \quad 12 \quad 13 \quad 6 \quad 1\,]$$

Calculate:

(a) —the order, n, of a minimal realization;

(b) —all admissible POFs and corresponding sets of POI, (Use *TFRo*.);

(c) —all admissible PCFs and corresponding sets of PCI, (Use *TFRc*.);

(d) —all admissible left coprime MFDs, (Use either *TFDN* or *RoDN*, with all POFs from (b) as input.);
(e) —all admissible right coprime MFDs; (Use either *TFND* or *RcND*, with all PCFs from (c) as input.);
(f) —the first 9 Markov parameters; Use either *RoH* or *RcH*; (As input arguments, use any of the previously obtained admissible realizations, R_o or R_c).

A version of an *L-A-S* program which solves this exercise is available in the *L-A-S* subdirectory C:\LAS\DPF\EXER43.DPF.

4.4 The purpose of these exercises is to provide a hands-on experience on almost all algorithms discussed in Chapter 4, as well as on various details about the *L-A-S* implementation of these algorithms. Various parts may be assigned, depending on the particular topics desired.

(1) Define an arbitrary (random) 5th order state space representation R_s = $\{\mathbf{A}_s,\mathbf{B},\mathbf{C},\mathbf{D}\}$ with $m=2$ inputs and $p=3$ outputs.
(2) Calculate the eigenvalues $\lambda_i = \sigma_i + j\,\omega_i$ of $\mathbf{A}_s$.
(3) Divide $\mathbf{A}_s$ with $3|\lambda_{max}|$; i.e. $\mathbf{A}_s/(3|\lambda_{max}|) = \mathbf{A}$.

Using R, calculate:

(4) —the transfer function $G(z) = W(z)/d(z)$,
(5) —the first 15 Markov parameters $\mathbf{H}_i$ in $H(z^{-1})$,
(6) —the PCF $R_c = \{\mathbf{A}_c,\mathbf{B}_c,\mathbf{C}_c,\mathbf{D}_c\}$ corresponding to $\mu = \{4,1\}$,
(7) —the POF $R_o = \{\mathbf{A}_o,\mathbf{B}_o,\mathbf{C}_o,\mathbf{D}_o\}$ corresponding to $\nu = \{1,3,1\}$,
(8) —the right coprime MFD $\{N_r(z),D_r(z)\}$ with $\{ n_i \} = \{ \mu_i \}$,
(9) —the left coprime MFD $\{D_l(z),N_l(z)\}$ with $\{ n_i \} = \{ \nu_i \}$.

Using the models (one at a time):

$\{ d(z),W(z) \}$, $H(z^{-1})$ and $\{ N_r(z),D_r(z) \}$, determine:
(10) —the POF R_{o1} corresponding to the given $\{ \nu_i \}$,
(11) —the POF R_{o2} corresponding to the given $\{ \nu_i \}$,
(12) —the POF R_{o3} corresponding to the given $\{ \nu_i \}$.

Using the models (one at a time):

$\{ d(z),W(z) \}$, $H(z^{-1})$ and $\{ N_r(z),D_r(z) \}$, determine:
(13) —the left coprime MFD $\{D_{l1}(z),N_{l1}(z)\}$ corresponding to $\{ n_i \} = \{ \nu_i \}$,

(14) —the left coprime MFD $\{D_{l2}(z), N_{l2}(z)\}$ corresponding to $\{ n_i \} = \{ \nu_i \}$.
(15) —the left coprime MFD $\{D_{l3}(z), N_{l3}(z)\}$ corresponding to $\{ n_i \} = \{ \nu_i \}$.
(16) Check that all the R_{oi}, $i=[1,3]$ obtained are equal to R_o.
(17) Check that all the $\{D_{li}(z), N_{li}(z)\}$, $i=[1,3]$ obtained are equal to $\{D_l(z), N_l(z)\}$.

Using the models (one at a time):

$\{ d(z), W(z) \}$, $H(z^{-1})$ and $\{ D_l(z), N_l(z) \}$, determine:
(18) —the PCF R_{c1} corresponding to the given $\{ \mu_i \}$,
(19) —the PCF R_{c2} corresponding to the given $\{ \mu_i \}$,
(20) —the PCF R_{c3} corresponding to the given $\{ \mu_i \}$.

Using the models (one at a time):

$\{ d(z), W(z) \}$, $H(z^{-1})$ and $\{ D_l(z), N_l(z) \}$, determine:
(21) —the right coprime MFD $\{N_{r1}(z), D_{r1}(z)\}$ corresponding to $\{ n_i \} = \{ \mu_i \}$,
(22) —the right coprime MFD $\{N_{r2}(z), D_{r2}(z)\}$ corresponding to $\{ n_i \} = \{ \mu_i \}$,
(23) —the right coprime MFD $\{N_{r3}(z), D_{r3}(z)\}$ corresponding to $\{ n_i \} = \{ \mu_i \}$.
(24) Check that all the R_{ci}, $i=[1,3]$ obtained are equal to R_c.
(25) Check that all the $\{N_{ri}(z), D_{ri}(z)\}$, $i=[1,3]$ obtained are equal to $\{N_r(z), D_r(z)\}$.
(26) Using $\{D_l(z), N_l(z)\}$ calculate the first 15 Markov parameters in $H_1(z^{-1})$.
(27) Using $\{N_r(z), D_r(z)\}$ calculate the first 15 Markov parameters in $H_2(z^{-1})$.
(28) Using $\{d(z), W(z)\}$ calculate the first 15 Markov parameters in $H_3(z^{-1})$.
(29) Check that all the $H_i(z^{-1})$, $i=[1,3]$ obtained are equal to $H(z^{-1})$.
(30) Using $H(z^{-1})$ calculate the transfer function $\{d_1(z), W_1(z)\}$.
(31) Using $\{D_l(z), N_l(z)\}$ calculate the transfer function $\{d_2(z), W_2(z)\}$.
(32) Using $\{N_r(z), D_r(z)\}$ calculate the transfer function $\{d_3(z), W_3(z)\}$.
(33) Check that all the $\{d_i(z), W_i(z)\}$, $i=[1,3]$ obtained are equal to $\{d(z), W(z)\}$.

Hints:

(1) Use either the *L-A-S* subroutine ABCD.SUB or the operator DPM four times.
(2) Use the operator EGV.
(3) Use the operators RPT , S* and S/ .
(4) Use the operator SSTF.
(5) Use the subroutine SSH.SUB.
(6) Use the subroutine SSRc.SUB.
(7) Use the subroutine SSRo.SUB.
(8) Use the subroutine RcND.SUB.
(9) Use the subroutine RoDN.SUB.
(10) Use the subroutine TFRo.SBR.

(11) Use the subroutine HRo.SBR.
(12) Use the subroutine NDRo.SBR.
(13) Use the subroutine TFDN.SBR.
(14) Use the subroutine HDN.SBR.
(15) Use the subroutine NDDN.SBR.
(16) Use the operators - and OUT in the MOS Roi,Ro(-)(out)= ; with *i*=[1,3].
(17) Use the operators - and OUT in the MOS Dli,Dl(-),Nli,Nl(-)(out)= .
(18) Use the subroutine TFRc.SBR.
(19) Use the subroutine HRc.SBR.
(20) Use the subroutine DNRc.SBR.
(21) Use the subroutine TFND.SBR.
(22) Use the subroutine HND.SBR.
(23) Use the subroutine DNND.SBR.
(24) Use the operators - and OUT in the MOS Rci,Rc(-)(out)= ; with *i*=[1,3].
(25) Use the operators - and OUT in the MOS Nri,Nr(-),Dri,Dr(-)(out)= .
(26) Use the subroutine DNH.SBR.
(27) Use the subroutine NDH.SBR.
(28) Use the subroutine TFH.SBR.
(29) Use the operators - and OUT in the MOS Hi,H(-)(out)= ; with *i*=[1,3].
(30) Use the subroutine HTF.SBR.
(31) Use the subroutine DNTF.SUB.
(32) Use the subroutine NDTF.SUB.
(33) Use the operators - and OUT in the MOS di,d(-),Wi,W(-)(out)= .

Additional "general" hints:

(1) From time to time enter the interpreter commands (IC) STATUS and NAMES to check the number of arrays defined and the total number of elements used by these arrays. If necessary, by the IC ELM, or the operator (ELM)= eliminate some of arrays. This can be done by either:

```
*  ELM ,x1,x2,x3, ... ,xn              (IC version) or
* (ELM)=x1,x2,x3, ... ,xn              (OS version)
```

(2) By using the IC HELP (H), syntactical descriptions of any operator statement (OS) or interpreter command (IC), as well as any subroutine of the type SUB or SBR can be obtained by:

```
*  h,xyz       for  any IC  or OS
*  h,sub,xyz   for any subroutine of type SUB or SBR
```

(3) At any time during the *L-A-S* session by using:

```
*   x,y,z(out)=            or
*   x,y,z(out,e)=          or
*   x,y,z(out,t,<n>)=        ; <n>:= 0 ¦ 1 ¦ 2
```

the desired arrays may be displayed on the screen. Similarly, by using:

```
*   x,y,z(out,L)=        or
*   x,y,z(out,L,e)=      or
*   x,y,z(out,L,<n>)=       ; <n>:= 0 ¦ 1 ¦ 2
```

the arrays may be written to the LASR "print" file.

(4) Before ending the session, the use of:

```
*   w,Prg                   or wpf,Prg
```

stores the sequence of *L-A-S* operators, i.e. the *L-A-S* program, on the Disk Program File. This program may later be retrieved from "DPF" and executed without retyping all statements. This can be done by:

```
* r,Prg                     or rpf,Prg
```

A version of an *L-A-S* program which solves this exercise is available in the *L-A-S* subdirectory C:\LAS\DPF\EXER44.DPF.

4.5 A 5th order uncontrollable and unobservable strictly proper system with $m=2$ inputs and $p=2$ outputs is given below in the system matrix form:

$$R_s = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}$$

namely,

$$R_s = \left[\begin{array}{ccccc|cc} 1.0 & -.5 & -3.0 & .0 & -1.5 & 1.0 & -1.0 \\ .0 & 2.5 & -1.0 & .0 & -1.5 & .0 & .0 \\ .0 & .5 & 4.0 & .0 & 1.5 & .0 & 1.0 \\ 3.0 & .5 & 3.0 & 4.0 & 1.5 & .0 & 1.0 \\ .0 & -.5 & 1.0 & .0 & 3.5 & .0 & .0 \\ \hline 1.0 & 1.0 & 1.0 & .0 & 1.0 & .0 & .0 \\ 1.0 & 1.0 & 1.0 & 1.0 & 2.0 & .0 & .0 \end{array}\right]$$

Determine minimal state space representations using:

(a) —a Hessenberg transformation,
(b) —a Kalman decomposition. Calculate also the dimensions of the subspaces $c\bar{o}$, $\bar{c}\bar{o}$ and $\bar{c}o$, and
(c) —the Jordan form decomposition.

Hints:

- See Appendix B for a discussion on these methods of obtaining a minimal realization.
- Define the representation R_s using the operator DMA, or INPM.
- For the Hessenberg transformation use either the operator MIN, subroutine MIN.SUB (twice), or subroutine MIN.SBR. For extra effort use each option. To check if all procedures give the same minimal representation, use the operator SSTF.
- For the Kalman decomposition use the subroutine KALD.SBR.
- For the Jordan Form minimal representation use the operators JFR and STR and eliminate the uncontrollable and/or unobservable modes. This can be done, for instance, using the operator DSM.
- Minimal representations $R_m = \{\mathbf{A}_m,\mathbf{B}_m,\mathbf{C}_m,\mathbf{D}_m\}$ can be built using the subroutine SYSM.SUB.

A version of an *L-A-S* program which solves this exercise is available in the *L-A-S* subdirectory C:\LAS\DPF\EXER45.DPF.

This Page Intentionally Left Blank

Chapter 5 System Identification

In the previous chapter several methods of conversion between system representations were presented. The purpose of this chapter is to present the important conversions between input/output data to various system models. This special catagory of conversions is called *identification*. In the first section the structural relationship, called the *identification identity*, which was discussed in Chapter 4, is reviewed.

5.1 The Identification Identity

Several of the intermodel conversions discussed in Chapter 4 were based on the relationship between the state space model in a POF and a corresponding left coprime MFD. Since this fundamental relation is also useful in system identification, it will be, in large part, repeated here.

As was done in Section 4.1.7, consider the order-n D-T system with m-inputs and p-outputs:

$$\begin{aligned} \mathbf{x}(t+1) &= \mathbf{A}_o\mathbf{x}(t) + \mathbf{B}_o\mathbf{u}(t) \\ \mathbf{y}(t) &= \mathbf{C}_o\mathbf{x}(t) + \mathbf{D}_o\mathbf{u}(t) \end{aligned} \tag{5.1}$$

where $R_o = \{\mathbf{A}_o, \mathbf{B}_o, \mathbf{C}_o, \mathbf{D}_o\}$ is in a POF corresponding to a set of admissible POI, $\boldsymbol{\nu} = \{\nu_i\}$. From Eq.(5.1) we may write

$$\begin{bmatrix} \mathbf{y}(t) \\ \mathbf{y}(t+1) \\ \vdots \\ \mathbf{y}(t+r) \end{bmatrix} = \begin{bmatrix} \mathbf{C}_o \\ \mathbf{C}_o\mathbf{A}_o \\ \vdots \\ \mathbf{C}_o\mathbf{A}_o^r \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} \mathbf{D}_o & 0 & \cdots & 0 & 0 \\ \mathbf{C}_o\mathbf{B}_o & \mathbf{D}_o & \cdots & 0 & 0 \\ & & \cdots & & \\ \mathbf{C}_o\mathbf{A}_o^{r-1}\mathbf{B}_o & \cdots & \mathbf{C}_o\mathbf{A}_o\mathbf{B}_o & \mathbf{C}_o\mathbf{B}_o & \mathbf{D}_o \end{bmatrix} \begin{bmatrix} \mathbf{u}(t) \\ \mathbf{u}(t+1) \\ \vdots \\ \mathbf{u}(t+r) \end{bmatrix} \tag{5.2}$$

Now we let $r = \nu_m = \max\{\nu_i\}$. Clearly, Eq.(5.2) holds for any integer $t = [0, N-r]$ and can be rewritten as

$$\mathbf{y}_t = \mathbf{Q}_{ov}\mathbf{x}(t) + \mathbf{H}\mathbf{u}_t \tag{5.3}$$

where $\mathbf{y}_t$ and $\mathbf{u}_t$ are $(\nu_m+1)p$ and $(\nu_m+1)m$ dimensional columns containing output and input vectors $\mathbf{y}(t+j)$ and $\mathbf{u}(t+j)$, $j = [0, \nu_m]$. The matrix $\mathbf{Q}_{ov}$ is the observability matrix of the pair $\{\mathbf{A}_o, \mathbf{C}_o\}$, while $\mathbf{H}$ is the $(r+1)p \times (r+1)m$ lower block triangular matrix containing along the main diagonal the $(p\times m)$ blocks $\mathbf{D}_o$. The other nonzero blocks of $\mathbf{H}$ are the $p\times m$ dimensional *Markov parameters*:

$$\mathbf{C}_o\mathbf{A}_o^j\mathbf{B}_o\ , \quad \text{for } j=[0,\ \nu_m-1] \tag{5.4}$$

Note that $\mathbf{H}$ in Eq.(5.3) equals $\mathbf{R}_k$ in Eq.(4.61), used in Algorithm *HDN*. Our goal is to eliminate from Eq.(5.2) the $\mathbf{x}(t)$ terms, thereby obtaining an expression which relates the sampled data to the elements in R_o.

Equation (5.2) can be considered to represent $(\nu_m+1)p$ scalar equations in the samples

$$y_{ij} = y_i(t+j) \tag{5.5}$$

i.e. the i^{th} component of the output vector $\mathbf{y}(t+j)$, $i=[1,\ p]$, $j=[0,\ \nu_m]$. In Section 3.3 it was shown that $\mathbf{Q}_{os}$ has n rows of an identity matrix and p rows that correspond to the rows of $\mathbf{A}_o$ with non-zero/non-unity elements. Furthermore, the location of these rows are determined by the selector vectors $\mathbf{v}_k$ and $\mathbf{v}_{kl}$, respectively.

Premultiplying Eq.(5.3) by the selector matrices $\mathbf{S}_{li}^T$ and $\mathbf{S}_{ld}^T$ defined by Eq.(3.79), we obtain, respectively,

$$\mathbf{y}_{1t} = \mathbf{x}(t) + \mathbf{H}_1\mathbf{u}_t\ , \quad \text{and} \quad \mathbf{y}_{2t} = \mathbf{A}_r\mathbf{x}(t) + \mathbf{H}_2\mathbf{u}_t \tag{5.6}$$

where

$$\mathbf{y}_{1t} = \mathbf{S}_{li}^T\mathbf{y}_t\ , \quad \mathbf{y}_{2t} = \mathbf{S}_{ld}^T\mathbf{y}_t \quad \text{with} \quad \mathbf{H}_1 = \mathbf{S}_{li}^T\mathbf{H}\ , \quad \mathbf{H}_2 = \mathbf{S}_{ld}^T\mathbf{H}$$

Eliminating $\mathbf{x}(t)$ from Eq.(5.6),

$$\mathbf{y}_{2t} = \begin{bmatrix} (\mathbf{H}_2 - \mathbf{A}_r\mathbf{H}_1) & \mathbf{A}_r \end{bmatrix} \begin{bmatrix} \mathbf{u}_t \\ \mathbf{y}_{1t} \end{bmatrix} \tag{5.7}$$

The matrix $\mathbf{A}_r$ in Eqs.(5.6) and (5.7) is a $(p\times n)$ matrix containing the rows of $\mathbf{A}_o$ with non-zero non-unity elements, whose locations in $\mathbf{A}_o$ are specified by the selector vector $\mathbf{v}_o$. Equation (5.7) may be expressed in a more concise form by

$$\mathbf{y}_{2t} = \begin{bmatrix} \mathbf{N}_r & \mathbf{A}_r \end{bmatrix}\mathbf{z}_t \tag{5.8}$$

where $\mathbf{N}_r = \mathbf{H}_2 - \mathbf{A}_r\mathbf{H}_1$ is a $p \times (\nu_m+1)m$ matrix and $\mathbf{z}_k$ is an h-dimensional vector containing $\mathbf{u}_t$ and y_{1t}, where $h = (\nu_m+1)m + n$. Equation (5.8) is referred to as the *identification identity* since it relates input/output data samples arranged into columns $\mathbf{y}_{2t}$ and $\mathbf{z}_t$ to parameters of the state space representation R_o, i.e. in the matrices $\mathbf{A}_o$, $\mathbf{B}_o$ and $\mathbf{D}_o$. The *identification identity* is the basis for conversions between input/output data and either state space or MFD models.

Equation (5.8) may now be rewritten as

$$\mathbf{y}_{2t} - \mathbf{A}_r\mathbf{y}_{1t} = \mathbf{N}_r\mathbf{u}_t \tag{5.9}$$

Note that Eq.(5.9) is a time-domain input/output expression. Applying the z-transform and taking into account the arrangements of the samples $u_i(t+j)$ and $y_k(t+j)$, $i=[1,m]$, $k=[1,p]$, $j=[0,\nu_m]$ in the vectors $\mathbf{u}_t$, $\mathbf{y}_{1t}$ and $\mathbf{y}_{2t}$, we obtain:

$$D(z)\,y(z) = N(z)\,u(z) \tag{5.10}$$

which is a left coprime MFD. Since in Eq.(5.9) the p dimensional vector $\mathbf{y}_{2t}$ is multiplied by the identity matrix $\mathbf{I}_p$, it may be concluded that $D(z)$ in Eq.(5.10) is monic. For further details see Section 3.4 and Eq.(3.104). Thus, in order to obtain the $[p \times (\nu_m+1)p]$ matrix $\mathbf{D}_r$, which leads directly to $D(z)$, it is first necessary to obtain the matrix $\mathbf{A}_r$. From the discussion in Section 3.3.4 it is clear that $\mathbf{A}_r$ may be obtained from $\mathbf{A}_o$ in a POF by:

$$\mathbf{S}_a^T \mathbf{A}_o = \mathbf{A}_r \tag{5.11}$$

where $\mathbf{S}_a$ is one of the selector matrices uniquely defined by the particular set of admissible POI $\boldsymbol{\nu}$ and generated by Algorithm *SMat*:

$$\boldsymbol{\nu}\ (SMat) \rightarrow \nu_n, \mathbf{S}_a, \mathbf{S}_b, \mathbf{S}_{ll}, \mathbf{S}_{ld}$$

Then the matrix $\mathbf{D}_r$ becomes

$$\mathbf{S}_{ld}^T - \mathbf{A}_r \mathbf{S}_{ll}^T = \mathbf{D}_r \tag{5.12}$$

For more details see Section 3.3.4.

5.2 Conversions from Input/Output Samples

Many times only input/output data is available, without a given system model. The process of creating a system model from the data is called system identification. The algorithms of this section can "identify" a system in either state space form or as an ARMA (MFD) model. In addition, the Markov parameters may be calculated from the input/output data. To obtain a matrix transfer function, it is recommended that one of the above mentioned forms be calculated first, i.e. state space or ARMA, although there is a procedure for the identification of the corresponding transfer matrix directly.

5.2.1 Input/Output Data to Observable State Form

This algorithm performs a deterministic D-T system identification by calculating an observable form state space model $R_o = \{\mathbf{A}_o, \mathbf{B}_o, \mathbf{C}_o, \mathbf{D}_o\}$ from a set of input and corresponding output data. Certain restrictions are placed on the input signals to ensure that the system excitation is "sufficiently rich." This will be explained subsequently. The algorithm is based on the identification identity, Eq.(5.8). The reader is urged to review Section 5.1 since we will assume familiarity here with the identification identity.

Thus, Eq.(5.8), i.e.

$$\mathbf{y}_{2t} = [\,\mathbf{N}_r \mid \mathbf{A}_r\,]\mathbf{z}_t\,, \quad \text{where} \quad \mathbf{z}_t = \begin{bmatrix} \mathbf{u}_t \\ --- \\ \mathbf{y}_{1t} \end{bmatrix} \tag{5.13}$$

establishes the linear dependance between the p dimensional vector $\mathbf{y}_{2t}$, containing samples $y_i(t+\nu_i)$, $i=[1,p]$, of the output vector $y(t)$ and:

- $(\nu_m+1)m$ dimensional vector $\mathbf{u}_t$ containing samples of the input vectors $\mathbf{u}(t+j)$, $j=[0,\nu_m]$ and
- n dimensional vector $\mathbf{y}_{1t}$ containing the samples $y_i(t+j)$, $j=[0,\nu_i\text{-}1]$ of the output vector $\mathbf{y}(t)$.

where ν is a set of admissible POI used in representing the system to be identified, while $k = \nu_m = \max\{\,\nu_i\,\}$. With $h = (\nu_m+1)m + n$ recall that $(p \times h)$ and $(p \times n)$ matrices $\mathbf{N}_r$ and $\mathbf{A}_r$ defining linear dependence in Eq.(5.8) contain:

- matrices $\mathbf{N}_i$ in the polynomial matrix $N(z)$ of the left coprime MFD relating $y(z)$ to $u(z)$, Eq.(5.10), and
- p rows with non-zero non-unity elements in the matrix $\mathbf{A}_o$, Eq.(5.11), of the state space representation R_o in a POF.

In order to determine $\mathbf{N}_r$ and $\mathbf{A}_r$, as well as to select an appropriate set ν of POI, the following is suggested. Concatenate the vectors y_{2t} and z_t corresponding to samples t=0,1,2, ..., q-1 into $(p \times q)$ and $(h \times q)$ matrices $\mathbf{Y}_2$ and $\mathbf{Z}$, respectively, (where it is assumed that $h < q$ and $q+\nu_m < N$), yielding:

$$\mathbf{Y}_2 = [\,\mathbf{N}_r \mid \mathbf{A}_r\,]\mathbf{Z}\,, \quad \text{where} \quad \mathbf{Z} = \begin{array}{c l} \left[\begin{array}{c} \mathbf{U} \\ --- \\ \mathbf{Y}_1 \end{array}\right] & \begin{array}{l} \}\,(\nu_m+1)m \\ \\ \}\,n \end{array} \end{array} \tag{5.14}$$

Note that the structure of the matrix $\mathbf{U}$ is given by:

$$\mathbf{U}_k = \begin{bmatrix} \mathbf{u}(0) & \mathbf{u}(1) & \cdots & \mathbf{u}(q-1) \\ \mathbf{u}(1) & \mathbf{u}(2) & \cdots & \mathbf{u}(q) \\ \vdots & & & \vdots \\ \mathbf{u}(k) & \mathbf{u}(k+1) & \cdots & \mathbf{u}(q+k-1) \end{bmatrix} \tag{5.15}$$

where $k = \nu_m$, while the matrices $\mathbf{Y}_1$ and $\mathbf{Y}_2$ appearing in Eq.(5.14) may be obtained by premultiplying the following $[(k+1)p \times q]$ matrix $\mathbf{Y}_k$.

$$\mathbf{Y}_k = \begin{bmatrix} \mathbf{y}(0) & \mathbf{y}(1) & \cdots & \mathbf{y}(q-1) \\ \mathbf{y}(1) & \mathbf{y}(2) & \cdots & \mathbf{y}(q) \\ \vdots & & & \vdots \\ \mathbf{y}(k) & \mathbf{y}(k+1) & \cdots & \mathbf{y}(q+k-1) \end{bmatrix} \tag{5.16}$$

by selector matrices $\mathbf{S}_{ii}^T$ and $\mathbf{S}_{id}^T$, respectively, i.e.

$$\mathbf{Y}_1 = \mathbf{S}_{ii}^T \mathbf{Y}_k \quad \text{and} \quad \mathbf{Y}_2 = \mathbf{S}_{id}^T \mathbf{Y}_k \tag{5.17}$$

Of course, the selector matrices used in Eq.(5.17) are generated by the set $\boldsymbol{\nu}$.

To emphasize similarities between this algorithm and the algorithms *TFDN* (*TFRo*) and *NDDN* (*NDRo*) discussed in Sections 4.2.5, 4.2.7, 4.4.7, and to facilitate understanding of the algorithm steps let:

$$\mathbf{Z}_k = \begin{bmatrix} \mathbf{U}_k \\ --- \\ \mathbf{Y}_k \end{bmatrix} \tag{5.18}$$

where $\mathbf{Z}_k$ is a $[(k+1)(m+p) \times q]$ matrix obtained by concatenating $\mathbf{U}_k$ and $\mathbf{Y}_k$ of the form in Eqs.(5.16) and (5.17), but now with the integer $k = 1, 2, \ldots, \nu_m$.

It is worth mentioning that Eq.(5.14) is similar to Eq.(4.34) used in Algorithm *TFDN* as well as *TFRo*, (and likewise Eq.(4.91) in *NDDN* and *NDRo*), which is to be expected since all these algorithms determine a left coprime MFD, i.e. a model which corresponds to a state space model in POF. Therefore, these algorithms are also rather similar. The following facts will also be seen from the algorithm, but they do not make much of a difference:

- —in the *TFDN* algorithm the transfer function matrix is given,
- —in the *NDDN* algorithm the right MFD is given,
- —while here only input/output samples are available.

Only one difference in the case of Eq.(5.14) is worth mentioning. Recall that, for instance, in Eq.(4.34) the first $m(k+1)$ rows in $\mathbf{T}_k$ are, by definition, linearly independent, and that in the last $p(k+1)$ rows of $\mathbf{T}_k$ there are n additional linearly independent rows. Since here, i.e. in Eq.(5.18), the same situation must occur, all $m(k+1)$ rows in the matrix $\mathbf{U}_k$ must be linearly independent, i.e. $\mathbf{U}_k$, containing only samples of the input vector $\mathbf{u}(t)$, must be a full row rank matrix, leading to:

$$\operatorname{rank}[\,\mathbf{U}_k\,] = m(k+1)\ , \quad \text{for all } k = [1, \nu_m] \tag{5.19}$$

An input signal $\mathbf{u}(t)$ satisfying the condition of Eq.(5.19) will be referred to as a "sufficiently rich" input, i.e. a *persistent excitation* capable of exciting all system modes. Note that for a given $\mathbf{u}(t)$ the condition of Eq.(5.19) might be satisfied for some value of k, but it might fail for a higher value of k.

As an example, consider a periodic input signal $\mathbf{u}(t)$, with $m=1$, given by:

$$\mathbf{u}(t) = [\ 1\ 0 \ldots 0\ 1\ 0 \ldots 0\ 1\ 0 \ldots 0\ 1 \ldots\ ]$$

having in a period one unity and h zeros. Building the matrix $\mathbf{U}_k$, Eq.(5.15), for various values of k, it may be concluded that $\mathbf{U}_k$ would be of full row rank only for $k \leq h$. Thus, this input signal may be used for identifying a D-T system of *any* order n, provided that the system may be represented by a POF having an admissible set of POI satisfying:

$$\max \{ \nu_i \} = \nu_m \leq h$$

Therefore, before using this algorithm it is advisable to check the available input and to determine the maximum value of k leading to a full row rank $\mathbf{U}_k$.

Syntax: $\mathbf{u}, \mathbf{y}, \epsilon, \nu_d$ *(uyRo)* $\Rightarrow \mathbf{A}_o, \mathbf{B}_o, \mathbf{C}_o, \mathbf{D}_o, \nu, \mathbf{x}(0), C\#$

Input/Output Arguments:

- $\mathbf{u}$ is an $(m \times N)$ matrix containing samples of m dimensional input vector.
- $\mathbf{y}$ is an $(p \times N)$ matrix containing samples of the system response.
- ϵ is a sufficiently small positive number used in rank calculations.
- $\nu_d = \{ \nu_i \}$, the set of "desired" POI. If ν_d is not known, any scalar, e.g. ϵ, may be used as the fourth argument.
- $R_o = \{\mathbf{A}_o, \mathbf{B}_o, \mathbf{C}_o, \mathbf{D}_o\}$, a state space representation in a POF.
- $\nu = \{ \nu_i \}$, a set of admissible POI corresponding to R_o.
- $\mathbf{x}(0)$ is the initial condition of the state vector $\mathbf{x}(t)$ corresponding to R_o.
- $C\#$ is the degree of admissibility of the set ν.

Algorithm:

1. If ν_d is specified, set $\nu_d \Rightarrow \nu$, set $\nu_m \Rightarrow k$, build $\mathbf{Z}_k$ Eq.(5.18) and go to 8; else, go to 2
2. Set $0 \Rightarrow k$ and $0 \Rightarrow n_o$
3. Set $k+1 \Rightarrow k$
4. Using the current k, build $\mathbf{Z}_k$, Eq.(5.18), and set rank($\mathbf{Z}_k$) - $mk \Rightarrow n$
5. If $n = n_o$ go to 6; else, set $n \Rightarrow n_o$ and go to 3
6. From $\mathbf{Z}_k$ determine the unique observability indices ν_u, i.e. $\mathbf{Z}_k$ *(IND)* $\Rightarrow \nu_u$
7. Define an appropriate admissible set of POI ν
8. Set ν *(SMat)* $\Rightarrow \nu_m, \mathbf{S}_a, \mathbf{S}_b, \mathbf{S}_{ll}, \mathbf{S}_{tl}$

```
9.  Using $S_{li}$, $S_{ld}$ and $k$, define the auxiliary matrices $\bar{S}_{li}$ and $\bar{S}_{ld}$,
    Eqs.(4.37) to (4.38)
10. Set $\bar{S}_{li}^T Z_k \Rightarrow Z$ and $\bar{S}_{ld}^T Z_k \Rightarrow Y_2$
11. Calculate the degree of admissibility of $Z$, i.e. $Z$ (C#) $\Rightarrow$ C#
12. If C# is "too small," go to 7; else, go to 13
13. Solve $XZ = Y_2$ for $X$, where $X = [\,N_r \mid A_r\,]$
14. Parition $I_n \Rightarrow \begin{bmatrix} C_o \\ A_2 \end{bmatrix}\}\, p$ . $C_o$ has $p$ rows
15. Set $S_l A_2 + S_d A_r \Rightarrow A_o$
16. Set $N_r$, $m$ (R2C) $\Rightarrow N_c$
17. Set $A_o$, $S_o$ (Qc) $\Rightarrow Q_c$, $Q_c$ has $(k+1)$ blocks $\{A_o^i S_o\}$ of $p$ columns
18. Set $Q_c N_c \Rightarrow B_o$
19. Set $D_o^{-1}(s_o)N_o(s_o) - C_o(s_o I - A_o)^{-1}B_o \Rightarrow D_o$ for any $s_o \neq$ a system pole
20. Using $R_o$ calculate the matrix $H$ in Eq.(5.3), set $S_l^T H \Rightarrow H_1$
21. Extract the first column from $Z \Rightarrow z_o$
22. Partition $z_o \Rightarrow [\,u_o^T \mid y_{1o}^T\,]^T$, $y_{1o}$ has $n$ elements
23. Set $y_{1o} - H_1 u_o \Rightarrow x(0)$
```

Since this algorithm is, formally, rather similar to the previously mentioned algorithms, *TFDN* and *NDDN*, it suffices to mention that Step 10 implements Eq.(5.17) and that the matrix Z_k used there corresponds to the matrix Z_k given by Eq.(5.18).

The last four steps of the above algorithm calculate the initial condition vector $x(0)$. The calculation is based on Eq.(5.6), i.e.:

$$x(0) = y_{1o} - S_{li}^T H\, u_o$$

where y_{1o} and u_o are the first columns of Y_1 and U_k, Eqs.(5.17) and (5.15), respectively, while H containing the first $\nu_m + 1$ Markov parameters of R_o is defined by Eq.(5.2).

The role of the fourth input argument, i.e. ν_d, is very crucial in Algorithm *uyRo*. It should be realized that for the case when the input/output sequences $u(t)$ and $y(t)$ are corrupted by measurement noise (or computational round-off errors for that matter) then the determination of linearly dependent rows in Z_k, done in Steps 4 and 6, leading to the unique set of ν might be rather unreliable. Computational experience has revealed that in the case of significant noise, the algorithm tends to "suggest" a system order higher than the true order. This is why Algorithm *uyRo* has an option of using ν_d. If ν_d is specified, then, as may be seen from the algorithm, the process of checking for linear dependent rows in Z_k is bypassed, and the algorithm operates in the "mode" of *model reduction*, where, of course, the order of the reduced-order model is equal to the sum of the elements in ν_d. In this case it is advisable to use several sets of indices ν_d having the same (or even a different order n) and to select the one which gives the largest degree of admissibility $C\#$. (Note that all algorithms calculate $C\#$ = the reciprocal of the "standard" condition number to avoid infinite numbers when a set ν_d is not admissible, or when **Z** is not a full row rank matrix.)

In fact, all other previously discussed algorithms having as the last input argument the set of "desired" indices such as:

{—the set of POI ν_d, or the set of PCI μ_d, or the set of column or row degrees $\mathbf{n}_d$}

have an option of operating in the "model reduction mode" which should somehow aleviate problems resulting from accumulated round-off error leading to erroneous rank determination and detection of linearly independent rows or columns.

The quantity ϵ used in these algorithms has a similar role. Computational experience reveals that a good value for ϵ is on the order of magnitude of 10^{-5} in the case of double precision calculations and "moderately well conditioned" problems and procedures leading to relatively small computational errors.

5.2.2 Input/Output Data to Left Coprime MFD

This algorithm calculates a left coprime column-reduced ARMA (MFD) model, $D(z)^{-1}N(z)$, from a corresponding set of input/output data. It is expected that the reader has so far realized significant (even striking) similarities between algorithms calculating a POF R_o and a left coprime MFD { $D(z)$, $N(z)$ } as well as between algorithms calculating a PCF R_c and a right coprime MFD { $N(z)$, $D(z)$ }. Checking previously given algorithms, it may be concluded that these algorithms differ only in several last steps where the specific matrices in R_o or in {$D(z)$, $N(z)$} are calculated. Thus, in this section we will, for completness, present an algorithm and emphasize that everything that was stated in Section 5.2.1 holds here as well.

Syntax: $\mathbf{u}, \mathbf{y}, \epsilon, \mathbf{n}_d$ (*uyDN*) $\Rightarrow$ **D**, **N**, $C\#$

Input/Output Arguments:

- **u** is an $(m \times N)$ matrix containing samples of the m dimensional input vector.
- **y** is an $(p \times N)$ matrix containing samples of the system response.
- ϵ is a sufficiently small positive number used in rank calculations.
- $\mathbf{n}_d$ = { n_i } is the set of "desired" column degrees. If $\mathbf{n}_d$ is not known, any scalar, e.g. ϵ, may be used as the fourth argument.
- **D** is a $[p^2 \times (k+1)]$ matrix in PMF. The rows of **D** contain the

coefficients d_{ijk} of the polynomials $d_{ij}(z)$ in $D(z)$.
- $\mathbf{N}$ is a $[pm \times (k+1)]$ matrix in PMF. The rows of $\mathbf{N}$ contain the coefficients n_{ijk} of the polynomials $n_{ij}(z)$ in $N(z)$.
- $C\#$ is the degree of admissibility of the set $\mathbf{n}_d$.

Algorithm:

```
1.  If $\mathbf{n}_d$ is specified, set $\mathbf{n}_d = \{n_i\}$, set $n_m \Rightarrow k$, build $\mathbf{Z}_k$, Eq.(5.18),
    and go to 8; else, go to 2
2.  Set $0 \Rightarrow k$ and $0 \Rightarrow n_o$
3.  Set $k+1 \Rightarrow k$
4.  Using the current $k$, build $\mathbf{Z}_k$, Eq.(5.18), and set rank($\mathbf{Z}_k$) $-mk \Rightarrow n$
5.  If $n = n_o$ go to 6; else, set $n \Rightarrow n_o$ and go to 3
6.  From $\mathbf{Z}_k$ determine the unique observability indices $\mathbf{n}_o$, i.e. $\mathbf{Z}_k$
    $(IND) \Rightarrow \mathbf{n}_o$
7.  Define an appropriate admissible set of column degrees $\mathbf{n}$
8.  Set $\mathbf{n}$ $(SMat) \Rightarrow n_m, \mathbf{S}_a, \mathbf{S}_l, \mathbf{S}_{ll}, \mathbf{S}_{ld}$
9.  Using $\mathbf{S}_{ll}$, $\mathbf{S}_{ld}$ and $k$, define the auxiliary matrices $\hat{\mathbf{S}}_{ll}$ and $\hat{\mathbf{S}}_{ld}$,
    Eqs.(4.37) - (4.38)
10. Set $\hat{\mathbf{S}}_{ll}^T\mathbf{Z}_k \Rightarrow \mathbf{Z}$ and $\hat{\mathbf{S}}_{ld}^T\mathbf{Z}_k \Rightarrow \mathbf{Y}_2$
11. Calculate the degree of admissibility of $\mathbf{Z}$, i.e. $\mathbf{Z}$ $(C\#) \Rightarrow C\#$
12. If $C\#$ is "too small," go to 7; else, go to 13
13. Solve $\mathbf{XZ} = \mathbf{Y}_2$ for $\mathbf{X}$, where $\mathbf{X} = [\,\mathbf{N}_r \mid \mathbf{A}_r\,]$
14. Set $\mathbf{S}_{ld}^T - \mathbf{A}_r\mathbf{S}_{ll}^T \Rightarrow \mathbf{D}_r$
15. Set $\mathbf{D}_r, p(PMFr) \Rightarrow \mathbf{D}$ and $\mathbf{N}_r, m(PMFr) \Rightarrow \mathbf{N}$
```

5.2.3 Input/Output Data to Markov Parameters

This algorithm calculates the Markov parameters of a system from its input/output data. It is based on Eq.(4.76) in Algorithm *uHy*, i.e.:

$$\mathbf{y}_i = \sum_{j=0}^{i} \mathbf{H}_{i-j}\mathbf{u}_j$$

but now, since the Markov parameters are unknown, this equation will be represented by:

$$[\, \mathbf{H}_0 \;\; \mathbf{H}_1 \;\cdots\; \mathbf{H}_{M-1} \,] \begin{bmatrix} \mathbf{u}_0 & \mathbf{u}_1 & & \cdots & \mathbf{u}_{N-1} \\ & \mathbf{u}_0 & & \cdots & \mathbf{u}_{N-2} \\ & & \ddots & & \vdots \\ & & & \mathbf{u}_0 \;\cdots & \mathbf{u}_{n-M} \end{bmatrix} = \tag{5.20}$$

$$[\mathbf{y}_0 \;\; \mathbf{y}_1 \;\; \cdots \;\; \mathbf{y}_{N-1}]$$

or, for short by:

$$\mathbf{H}_r \mathbf{U} = \mathbf{y}$$

where $\mathbf{H}_r$ is a $(p \times Mm)$ matrix containing the Markov parameters $\mathbf{H}_i$, $i=[0,M-1]$, to be determined, $\mathbf{U}$ is an $(Mm \times N)$ matrix containing the samples $\mathbf{u}_i$, $i=[0,N-1]$, of the input vector $\mathbf{u}$ arranged according to Eq.(5.20), while $\mathbf{y}$ is a $(p \times N-1)$ matrix containing N samples of the output vector $\mathbf{y}$.

In order to obtain the unique solution for the Markov parameters $\mathbf{H}_i$ satisfying Eqs.(4.76) and (5.20), the matrix $\mathbf{U}$ must be a full row rank matrix, i.e.:

$$\text{rank}\{\, \mathbf{U} \,\} = Mm \tag{5.21}$$

leading to the following constraint:

$$Mm \leq N \tag{5.22}$$

Note that Eq.(5.20), as do some other equations, e.g. Eqs.(4.6), (4.32), (4.80) and (4.81), assumes that:

$$h_M = \| \mathbf{H}_{M-1} \| << 1 \tag{5.23}$$

Thus, this algorithm is applicable only to stable D-T systems, under the condition that the input/output sequences are sufficiently long to satisfy Eqs.(5.21) - (5.23). Note that Eq.(5.21) requires that the rows of the input signal $\mathbf{u}(t)$ be linearly independent, which is another condition specifying a "sufficiently rich" input. For more details see Section 5.2.1.

Syntax: $\mathbf{u}, \mathbf{y}, M \; (uyH) \Rightarrow \mathbf{H}, h_M$

Input/Output Arguments:

- $\mathbf{u}$ is an $(m \times N)$ matrix containing samples of m dimensional input vector.
- $\mathbf{y}$ is an $(p \times N)$ matrix containing samples of the system response.
- M is an integer specifying the number of Markov parameters to be calculated.
- $\mathbf{H}$ is a $[pm \times M]$ matrix in the PMF. The rows of $\mathbf{H}$ contain the first M coefficients h_{ijk} of the polynomials $h_{ij}(z^{-1})$ in $H(z^{-1})$.
- h_M is the norm of the last Markov parameter calculated, Eq.(5.23).

5.2.4 Input/Output Data to Transfer Function

This algorithm calculates transfer function matrix $G(z)$ of a D-T system from its input/output data. It is based on:

$$y(z) = G(z)\, u(z) \tag{5.24}$$

where $G(z)$ can be expressed as:

$$G(z) = \{\, g_{ij}(z)\, \} \tag{5.25}$$

with individual transfer functions $g_{ij}(z)$, relating the contribution of the j^{th} input $u_j(z)$ to the i^{th} output $y_i(z)$, $j=[1,m]$, $i=[1,p]$, by:

$$y_{ij}(z) = g_{ij}(z)\, u_j(z)\ , \quad \text{where} \quad g_{ij}(z) = \frac{\tilde{w}_{ij}(z)}{d_i(z)} \tag{5.26}$$

In other words, in this algorithm a given MIMO system is decomposed into a set of p MISO (multi-input, single-output) subsystems, and each subsystem is identified one at a time. This is why the denominators $d_i(z)$ in Eq.(5.26) are different for different values of i, $i=[1,p]$. Of course, the common denominator $d(z)$ for all $d_i(z)$ used and calculated in previous algorithms satisfies:

$$G(z) = \frac{W(z)}{d(z)} \quad \text{and} \quad d(z) = f_i(z)\, d_i(z) \tag{5.27}$$

It is worth mentioning that the orders n_i of polynomials $d_i(z)$ are equal to the "individual observability indices" which are dual to the "individual controllability indices" $\{\, \alpha_i \,\}$ introduced by Definition 3.1 in Section 3.3.3. The roots of the monic polynomials $f_i(z)$, appearing in Eq.(5.27), correspond to modes (poles) of the given MIMO system "not seen" by the i^{th} MISO subsystem individually.

From the relationship between polynomials $d(z)$ and $d_i(z)$ in Eq.(5.27), it may be concluded that the numerator polynomial matrix $W(z)$ in Eq.(5.27) is related to the $(p \times m)$ matrix $\tilde{W}(z)$ containing $\tilde{w}_{ij}(z)$ from Eq.(5.26), i.e.:

$$\begin{gathered} \tilde{W}(z) = \{\, \tilde{w}_{ij}(z)\, \} \quad \text{by} \\ W(z) = F(z)\, \tilde{W}(z)\ , \quad \text{where} \quad F(z) = \operatorname{diag}\{\, f_i(z)\, \} \end{gathered} \tag{5.28}$$

The algorithm is as follows:

Syntax: u, y, ϵ, n$_d$ *(uyTF)* → D, $\tilde{W}$, n$_o$, C#

Input/Output Arguments:

- u is an $(m \times N)$ matrix containing samples of the m dimensional input vector.

- $\mathbf{y}$ is an $(p \times N)$ matrix containing samples of the system response.
- ϵ is a sufficiently small positive number used in rank calculations.
- $\mathbf{n}_d = \{ n_i \}$ is the set of "desired" individual observability indices of p MISO subsystems. If $\mathbf{n}_d$ is not known, any scalar, e.g. ϵ, may be used as the fourth argument.
- $\mathbf{D}$ is a $[p \times (n_m+1)]$ matrix in PMF. The rows of $\mathbf{D}$ contain the coefficients d_{ik} of the polynomials $d_i(z)$, $n_m = \max\{n_i\}$.
- $\bar{\mathbf{W}}$ is a $[pm \times (n_m+1)]$ matrix in PMF. The rows of $\bar{\mathbf{W}}$ contain the coefficients $\bar{w}_{ijk}$ of the polynomials $\bar{w}_{ij}(z)$ in $\bar{W}(z)$, Eq.(5.26).
- $\mathbf{n}_o$ is a set of individual observability indices $\{n_i\}$ containing the orders of the polynomials $d_i(z)$. If $\mathbf{n}_d$ is used, then $\mathbf{n}_o = \mathbf{n}_d$.
- $\mathbf{C\#}$ is the p-dimensional row vector containing the admissibility degrees of the matrices used in identifying the individual MISO subsystems.

Remarks:

In order to obtain the transfer function matrix $G(z)$, i.e. $W(z)$ and $d(z)$, Eq.(5.27), of the overall MIMO system, $G(z) = W(z)/d(z)$, a service algorithm "Common Denominator" (*ComD*) may be used. Its syntax is:

$$\mathbf{D},\ \epsilon\ (ComD) \Rightarrow \mathbf{d},\ \mathbf{F}$$

The algorithm *ComD* uses the column $D(z) = \{ d_i(z) \}$ and calculates a common denominator $d(z)$ and the diagonal polynomial matrix $F(z)$, Eq.(5.28). Then, the matrix $W(z)$ of the overall MIMO system may be obtained using Eq.(5.28), i.e. by premultiplying $\bar{W}(z)$ with $F(z)$.

In spite of the availability of this algorithm, as was mentioned in the beginning of this subsection, due to the relatively involved procedure used, its use is recommended only, if for some reasons, it is required to have the individual transfer functions $g_{ij}(z)$ and denominators $d_i(z)$, Eq.(5.26), defining the single output subsystems.

More details about the procedure used in the *uyTF* algorithm will be given later.

Basic steps of the algorithm are:

1. Set $\mathbf{0}_{1,0} \Rightarrow \mathbf{C\#}$, $\mathbf{0}_{1,0} \Rightarrow \mathbf{n}_o$, $\max\{n_{di}\} + 1 = k$, $\mathbf{0}_{0,k} \Rightarrow \mathbf{D}$, $\mathbf{0}_{0,k} \Rightarrow \bar{\mathbf{W}}$
2. Set $0 \Rightarrow 1$
3. Set $i+1 \Rightarrow i$
4. If $\mathbf{n}_d$ is specified, set $n_d(i) \Rightarrow n_{di}$, and go to 6; else, go to 5
5. Set $\epsilon \Rightarrow n_{di}$

```
6.  Extract the $i^{th}$ row from $\mathbf{y} \Rightarrow \mathbf{y}_i$
7.  Set $\mathbf{u}$, $\mathbf{y}_i$, $\epsilon$, $n_{it}$ (uyDN) $\Rightarrow \mathbf{d}_i$, $\tilde{\mathbf{W}}_i$, $n_i$, $C\#_i$
8.  Set $[\ \mathbf{C\#} \mid C\#_i\ ] \Rightarrow \mathbf{C\#}$ and $[\ \mathbf{n}_v \mid n_i\ ] \Rightarrow \mathbf{n}_v$
9.  Set $\begin{bmatrix} \mathbf{D} \\ \mathbf{d}_i \end{bmatrix} \Rightarrow \mathbf{D}$ and $\begin{bmatrix} \tilde{\mathbf{W}} \\ \tilde{\mathbf{W}}_i \end{bmatrix} \Rightarrow \tilde{\mathbf{W}}$
10. If $i < p$, go to 3; else, go to 11
11. Set $\tilde{W}^T(z) \Rightarrow \tilde{W}(z)$
```

It is seen from the algorithm steps that Algorithm *uyTF* executes the previously explained Algorithm *uyDN*, Step 7, p times. Since $\mathbf{y}_i$ contains just one row, the obtained $\mathbf{d}_i$ is a single row containing an $(n_i + 1)$ dimensional row with the coefficients of $d_i(z)$, and $\tilde{\mathbf{W}}_i$ contains the coefficients $\tilde{w}_{ijh}$, $j=[1,m]$, $h=[0,n_i]$, of all the polynomials $\tilde{w}_{ij}(z)$ in the i^{th} row of $\tilde{W}(z)$. This is the reason why in Step 9 the concatenation of $\mathbf{D}$ with $\mathbf{d}_i$ and $\tilde{\mathbf{W}}$ with $\tilde{\mathbf{W}}_i$ is done. In order to have the matrix $\tilde{\mathbf{W}}$ in the PMF "structure," it is necessary in Step 11 to perform a polynomial matrix transposition, which is done by a service algorithm called *PMT*.

At this point it of some interest to compare the computational aspects of Algorithms *uyDN* and *uyTF*. Recall that in treating MIMO systems, i.e. in treating all p outputs simultaneously, Algorithm *uyDN*, the number of rows in the matrix $\mathbf{Z}$, Eq.(5.14), which greatly influences the computational aspect, is given by:

$$n + (\nu_m + 1)m$$

where it may be shown that: $[(n-1)/p] + 1 \le \nu_m \le n-p+1$ with $[\ x\ ]$ being the integer part of x.

In the case of treating MISO subsystems one at a time, Algorithm *uyTF*, the numbers of rows in the matrices $\mathbf{Z}$ are given by:

$$n_i + (n_i + 1)m \quad \text{for} \quad i=[1,p]$$

where, of course, the individual observability index n_i satisfies: $n_i \le n$. However, when at least one MISO subsystem "sees" all n MIMO system modes, then we have $n_i = n$.

From the above analysis it may be concluded that in Algorithm *uyTF*:

- The algorithm requires the building and manipulation of the matrix $\mathbf{Z}$, Eq.(5.14), p times.

- In addition, it frequently occurs that the dimensions of the matrix **Z** are considerably larger than the dimensions of the single **Z** required in Algorithm *uyDN*.

Recall that when the number of rows in **Z** is smaller, a smaller number of samples is needed, which considerably reduces the computational effort of Algorithm *uyDN* and, consequently, also of *uyRo*, since they are similar.

For example, in the case of an MIMO system with $n=6$ and $m=p=3$, Algorithm *uyTF* may require building and manipulating the matrix **Z** three times with:

$$6 + 7 \times 3 = 27 \text{ rows}$$

and at least 27 columns, while in the case of identifying the same MIMO system using either *uyDN* or *uyRo*, it suffices to build only one **Z** having the number of rows as low as

$$6 + 3 \times 3 = 15$$

and at least 15 columns, which illustrates the computational advantage of Algorithms *uyDN* and *uyRo* over *uyTF*.

The reason for insisting on Algorithm *uyTF* is that quite a number of papers and books on system identification advocate treating individual outputs separately, i.e. decomposing a particular MIMO system into a set of p MISO subsystems, which clearly is not a good policy, particularly with the advent of algorithms such as *uyRo* and *uyDN*, which utilize to the fullest extent the structural properties of MIMO systems. Another reason is to draw attention to Algorithms *uyDN* and *uyRo* and to show how a judicious choice of a MIMO system "canonical" form can reduce the computational effort of identification algorithms.

5.3 Conversions between D-T and C-T

We have already mentioned that many of the conversions presented for D-T systems are useable for C-T systems as well. It is also important to recall the algorithms presented in Chapter 2 for converting between C-T and D-T domains. For completeness these algorithms, which offer the user some flexibility in converting between continuous and discrete domains, are listed below. The possible conversions are listed in Table 5.1.

Table 5.1
Conversions between D-T and C-T Domains

	Available Conversions	Algorithm Flag (*Isrb*)
1.	C-T state space model ⇒ D-T (step invariant model)	1
2.	C-T state space model ⇒ D-T (ramp invariant model)	2
3.	C-T state space model ⇒ D-T (bilinear transformation)	3
4.	D-T state space model ⇒ C-T (step invariant model)	-1
5.	D-T state space model ⇒ C-T (ramp invariant model)	-2
6.	D-T state space model ⇒ C-T (bilinear transformation)	-3

To perform all these conversions, a single algorithm referred to as CTDT has been developed. The general syntax is:

Algorithm *CTDT*:

$$\mathbf{A},\ \mathbf{B},\ \mathbf{C},\ \mathbf{D},\ T,\ \epsilon,\ Isrb\ (CTDT) \Rightarrow \mathbf{A}_1,\ \mathbf{B}_1,\ \mathbf{C}_1,\ \mathbf{D}_1$$

Input/Output Arguments:

- $R = \{\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}\}$ is a state space representation in either the C-T or D-T domain to be converted by one of the procedures discussed in Chapter 2; see Table 5.1 above.
- T is the sampling interval.
- ϵ is a sufficiently small positive number used as machine zero.
- *Isrb* is the algorithm flag, *Isrb* = [1, 2, 3, -1, -2, -3], specifying a desired conversion.
- $R_1 = \{\mathbf{A}_1, \mathbf{B}_1, \mathbf{C}_1, \mathbf{D}_1\}$ is the representation obtained after the conversion of R according to the specified value of the algorithm flag *Isrb*.

To specify a desired conversion, the value of the algorithm flag *Isrb* (for step, ramp, or bilinear transformation) should be selected in accordance with a value given in Table 5.1.

For instance, if a conversion from C-T into D-T domain using the ramp invariant model is desired, then the value of *Isrb* should be equal to 2. In the case of converting a given D-T model into the C-T domain using the bilinear transformation, the value of *Isrb* should be set to -3.

According to the expressions and examples given in Chapter 2, it may be concluded that the following sequence of algorithms:

$$\mathbf{A},\ \mathbf{B},\ \mathbf{C},\ \mathbf{D},\ T,\ \epsilon,\ k\ (CTDT) \Rightarrow \mathbf{A}_1,\ \mathbf{B}_1,\ \mathbf{C}_1,\ \mathbf{D}_1$$
$$\mathbf{A}_1,\ \mathbf{B}_1,\ \mathbf{C}_1,\ \mathbf{D}_1,\ T,\ \epsilon,\ -k\ (CTDT) \Rightarrow \mathbf{A}_2,\ \mathbf{B}_2,\ \mathbf{C}_2,\ \mathbf{D}_2$$

for any value of $k = [\ 1,\ 2,\ 3]$, produces a representation R_2 which is very close, if not equal, to the given representation R.

For more specific details about the structure of Algorithm *CTDT*, see Chapter 2 and particularly Section 2.5.

Finally, Section 5.4 illustrates the conversion process between C-T and D-T representations as well as within the same domain. A primary emphasis in this example is system identification, i.e. "conversion" from input/output data to state space, or other forms. Figure 5.1 is presented with the section to provide a graphical picture of the conversions of that example.

5.4 Identification Examples

In this example we will begin with the C-T state space that was used in the two examples of Section 4.6. The purpose here is to illustrate the process of conversion between the continuous-time and the discrete-time domain, as well as conversion among the different model forms within a given domain. Figure 5.1 shows the particular operations, which are also listed below.

The differently derived responses: y, yt, yd, yd1, ydh and yc are basically identical. The models developed are the continuous-time transfer function matrices: TF and TFc, and the discrete-time state space representations in POF: Rdo , Ro1 and Roh, the discrete-time system Markov parameters in Hd and H and the discrete-time left coprime MFD's: {D,N} and {D1,N1}.

The given C-T state space model and corresponding transfer function were already presented in Example 1 and will not be repeated here. Conversion from a C-T state space model to a D-T state space model, using a sampling interval of 1 second, is first developed. Then, using a set of admissible POI, the D-T models corresponding to the C-T models of Example 4.1 are obtained, including those models obtained by applying the identification procedures of this chapter.

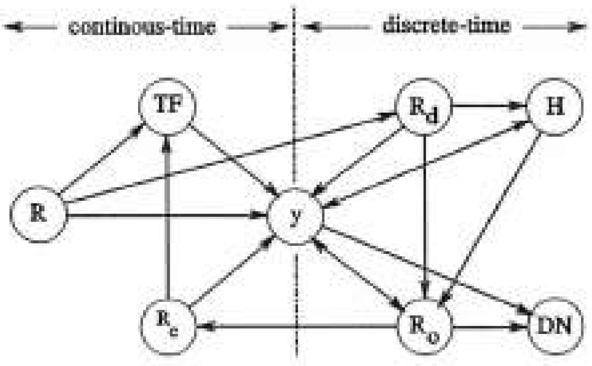


FIGURE 5.1 Conversions for the Example

Sequence of algorithm executions:

R (*SSTF*) ⇒ TF
R (*CDSR*) ⇒ y
TF (*CDTR*) ⇒ yt
R (*CTDT*) ⇒ Rd
Rd (*CDSR*) ⇒ yd
Rd (*SSRo*) ⇒ Rdo
Rd (*SSH*) ⇒ Hd
Rdo (*RoDN*) ⇒ D,N

u,y (*uyRo*) ⇒ Ro1
u,y (*uyDN*) ⇒ D1,N1
u,y,M (*uyH*) ⇒ H
u,H (*uHy*) ⇒ yd1
H (*HRo*) ⇒ Roh
Roh (*CDSR*) ⇒ ydh
Ro1 (*CTDT*) ⇒ Rc
Rc (*CDSR*) ⇒ yc
Rc (*SSTF*) ⇒ TFc

Results:

D-T conversion to $R_d = \{\mathbf{A}_d, \mathbf{B}_d, \mathbf{C}_d, \mathbf{D}_d\}$:

$$R_d = \left[\begin{array}{cccc|ccc}
.368 & .204 & .090 & .028 & 0.4 & .077 & .026 \\
0.0 & .073 & .114 & .062 & 0.0 & .110 & .052 \\
0.0 & -.114 & .073 & .114 & 0.0 & .135 & .110 \\
0.0 & 0.0 & 0.0 & .135 & 0.0 & 0.0 & .187 \\
\hdashline
0.0 & .001 & 0.0 & 1.0 & 0.0 & 1.0 & .284 \\
1.0 & 0.0 & 0.0 & 0.0 & .368 & .016 & .003
\end{array}\right]$$

As in Example 1 of Section 4.6 the following tables give the structural information regarding the D-T model. The "best" controllable and observable structures are specified by PCI = {1,2,1} and POI = {1,3}, respectively, which are again different from the corresponding unique controllability and observability indices, {2,1,1} and {2,2}.

TABLE 5.1 PCI

$\{n_{c1}\ n_{c2}\ n_{c3}\}$	rank	degree
{1 2 1}	4	.46E-01
{1 1 2}	4	.25E-01
{2 1 1}	4	.11E-03

TABLE 5.2 POI

$\{n_{o1}\ n_{o2}\}$	rank	degree
{1 3}	4	.22E-01
{2 2}	4	.12E-03
{3 1}	4	.14E-04

$$\textbf{Transfer Function Matrix} \quad G_d(z) = \frac{W_d(z)}{d_d(z)}$$

where
$$W_d(z) = \begin{bmatrix} w_{11}(z) & w_{12}(z) & w_{13}(z) \\ w_{21}(z) & w_{22}(z) & w_{23}(z) \end{bmatrix}$$

$$\begin{array}{c|ccccc|}
 & z^0 & z^1 & z^2 & z^3 & z^4 \\
w_{11} & 0 & 0 & 0 & 0 & 0 \\
w_{21} & -.001 & .004 & -.060 & .161 & .368 \\
\hdashline
w_{12} & .001 & -.016 & .142 & -.649 & 1.0 \\
w_{22} & 0.0 & -.003 & .015 & .067 & .016 \\
\hdashline
w_{13} & -.001 & .009 & -.056 & .003 & .284 \\
w_{23} & 0.0 & .002 & .019 & .024 & .003
\end{array} = W_d(z)$$

The characteristic polynomial $d_d(z)$ is given by:

$$d_d(z) = 0.001 - 0.016\,z + 0.142\,z^2 - .649\,z^3 + 1\,z^4$$

Left Coprime MFD

This form is given by $D_l^{-1}(z)N_l(z)$ where $D_l(z)$ is monic and column-reduced, i.e.:

$$D_l(z) = \begin{bmatrix} d_{11}(z) & d_{12}(z) \\ d_{21}(z) & d_{22}(z) \end{bmatrix}$$

$$N_l(z) = \begin{bmatrix} n_{11}(z) & n_{12}(z) & n_{13}(z) \\ n_{21}(z) & n_{22}(z) & n_{23}(z) \end{bmatrix}$$

specifically, the D-T left coprime MFD form is presented, with column degrees {1,3} corresponding to the selected POI above. Compare with the C-T left coprime MFD in Example 1, Section 4.6.

$$\begin{array}{c} \\ d_{11} \\ d_{21} \\ \\ d_{12} \\ d_{22} \end{array} \begin{array}{c} \begin{array}{cccc} z^0 & z^1 & z^2 & z^3 \end{array} \\ \left[\begin{array}{cccc} -.135 & 1.0 & 0.0 & 0.0 \\ -.004 & 0.0 & 0.0 & 0.0 \\ \hline 0.0 & .002 & -.005 & 0.0 \\ -.007 & .072 & -.512 & 1.0 \end{array}\right] \end{array} = D_l(z)$$

and

$$\begin{array}{c} \\ n_{11} \\ n_{21} \\ \\ n_{12} \\ n_{22} \\ \\ n_{13} \\ n_{23} \end{array} \begin{array}{c} \begin{array}{cccc} z^0 & z^1 & z^2 & z^3 \end{array} \\ \left[\begin{array}{cccc} 0.0 & -.001 & -.002 & 0.0 \\ .005 & -.032 & .211 & .368 \\ \hline -.135 & 1.0 & 0.0 & 0.0 \\ -.003 & .025 & .069 & .016 \\ \hline .148 & .284 & 0.0 & 0.0 \\ .004 & .022 & .024 & .003 \end{array}\right] \end{array} = N_l(z)$$

Markov Parameters $\mathbf{H}_i$

The first few terms of the 3-column polynomial matrix $H(z^{-1}) = \{h_{ij}(z^{-1})\}$ are given below. Because the terms decrease so fast, it is not necessary to extend the series.

$$\begin{array}{c|ccccccc|}
 & z^{0} & z^{-1} & z^{-2} & z^{-3} & z^{-4} & z^{-5} & z^{-6} \\
h_{11} & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\
h_{21} & .3680 & .3998 & .1471 & .0541 & .0199 & .0073 & .0027 \\
 & \text{----} & \text{----} & \text{----} & \text{----} & \text{----} & \text{----} & \text{----} \\
h_{12} & 1.0000 & .0001 & .0000 & 0.0 & 0.0 & 0.0 & 0.0 \\
h_{22} & .0160 & .0771 & .0631 & .0278 & .0102 & .0037 & .0014 \\
 & \text{----} & \text{----} & \text{----} & \text{----} & \text{----} & \text{----} & \text{----} \\
h_{13} & .2838 & .1870 & .0253 & .0034 & .0005 & .0000 & .0000 \\
h_{23} & .0027 & .0255 & .0352 & .0215 & .0094 & .0036 & .0013
\end{array}$$

By the 12th element in the above series all terms are zero to 4 decimal places.

System Identification

One of the most useful "conversions" is that from input/output data to a system model, system identification. An important aspect is determining and using the most appropriate structural information, e.g. the most numerically stable POF. This part of the example presents the results of such an identification process, using the sampled data from the given C-T system model to obtain an observable form D-T model, which is subseqently *continualized* using the methods established in Chapter 2. The following C-T model is the result of this series of conversions:

$$R = \left[\begin{array}{cccc|ccc} -2.001 & 0.0 & -.012 & .030 & 0.0 & 0.0 & 1.0 \\ .281 & -2.934 & 11.130 & -15.963 & 1.0 & .001 & -.001 \\ -.025 & -.107 & -1.789 & 2.934 & .368 & .090 & .028 \\ .008 & .020 & -.318 & -.282 & .135 & .063 & .037 \\ \hdashline 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \end{array}\right]$$

The input/output data from the C-T system that was used in the identification of the D-T models, Ro1, {D1,N1} and H (see Fig.5.1), are presented in Figs. 5.2 and 5.3.

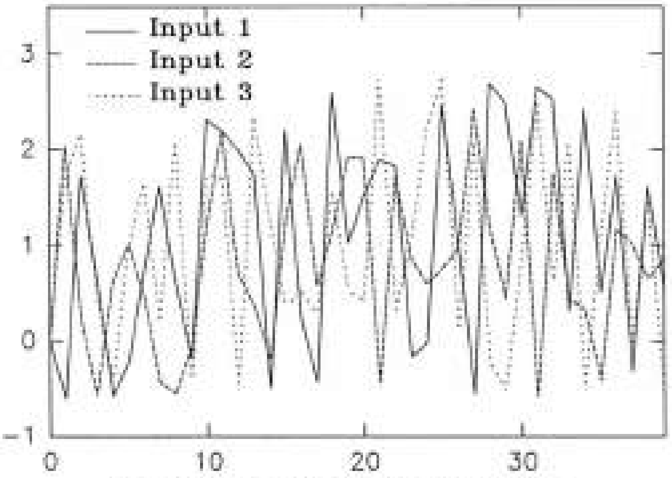


Figure 5.2 Pseudo-Random System Excitation

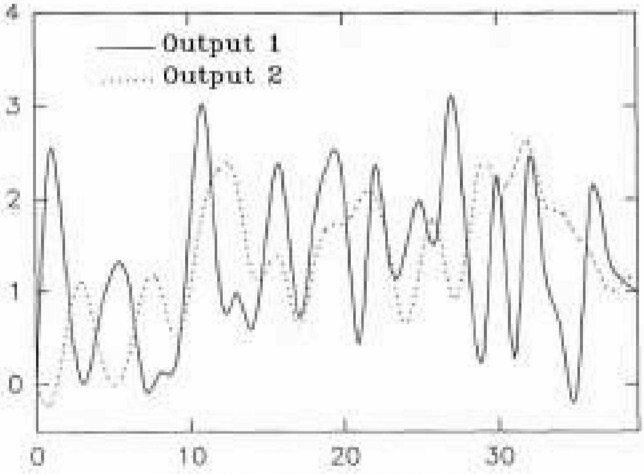


Figure 5.3 C-T System Responses

It is interesting to compare the eigenvalues (poles) of the C-T state space model that was given initially with the identified C-T model, i.e. with the model obtained by *continualizing* the identified D-T model:

Poles of the original C-T system:
$-2 \pm j1, \quad -2, \quad -1$

Poles of the identified C-T system:
$-2.003 \pm j1.001, \quad -2.0, \quad -1.0$

As derived from the identified D-T system with poles:
$0.073 \pm j0.114, \quad 0.135, \quad 0.368$

which confirms that a series of conversions that "loops" back on itself is numerically stable. By the procedure described in Appendix B, *degrees of observability* of the modes in the C-T model can be checked. It has been shown that the mode $-2 \pm j1$ has a considerably smaller "degree of observability" than do the other two modes.

5.5 Summary

In this chapter the process of conversion from system input/output data to various representations was stressed. This process is referred to as system identification. If we consider the input/output data to be a system "representation," then the algorithms in this chapter fit with the many algorithms of Chapter 4 in the sense of Table 4.1. On the other hand, system identification plays an eminent role in that it is this process that is generally required first, to obtain a more standard model with which to work. Because of this importance, this chapter was presented to emphasize this conversion type.

5.6 References

There are many good references written for system identification. To mention and recommend for further reading just a few of these: Sinha and Kuszta (1983) and Ljung (1987) for a general survey of identification methods. For more specific background and information, particularly on identification of MIMO systems using pseudo-observable forms, several other articles and chapters are listed below.

Bingulac, S. and D.L. Cooper (1991), "Identification of first-order hold continuous-time systems," *Proceedings of the 9th IFAC Symposium on Identification*, Budapest, Hungary, August 20-25, 1991, pp. 1185-1190.

Bingulac, S. and D.L. Cooper (1991), "Use of pseudo-observability indices in identification of continuous-time multivariable models," *Identification of Continuous-Time Systems* (N.K. Sinha and G.P. Rao, editors), Kluwer Academic Publishers, Amsterdam.

Bingulac, S. and R. Krtolica (1988), "An algorithm for simultaneous order and parameter identification in multivariable systems," *Proceedings of the 8th IFAC Symposium on Identification and System Parameter Estimation*, Beijing, August 27-31, 1988, pp. 1020-1025.

Bingulac, S. and R. Krtolica (1985), "Generalized ARMA model for MIMO system identification," *Proceedings of the American Control Conference*, Boston MA, June 11-14, 1985, pp. 1336-1341.

Bingulac, S. and N.K. Sinha (1990), "Identification of continuous MIMO systems from input/output data," *Journal of Mathematical and Computer Modelling*, **5**, 3, pp. 203-208.

Bingulac, S. and H.F. VanLandingham (1992), "Multivariable system identification with noisy data," *Proceedings of the IEEE International Symposium on Systems, Man and Cybernetics*, October 5-8, 1992, Chicago IL.

Gorti, B., S. Bingulac and H.F. VanLandingham (1990), "Deterministic identification of linear MIMO systems," *Proceedings of the 22nd Southeastern Symposuim on System Theory*, Cookeville TN, April 19-22, 1990, pp. 126-131.

Ljung, L. (1987), *System Identification: Theory for the User*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Sinha, N.K. and B. Kuszta (1983), *Modeling and Identification of Dynamic Systems*, Van Nostrand Reinhold, Inc., New York.

VanLandingham, H.F., S. Bingulac and M. Tran (1992), "A comparison of conventional and neural network approaches to system identification," *Journal of Control Theory and Advanced Technology*, **8**, 4, MITA Press, Kyoto, Japan.

5.7 Exercises

5.1 Given the state space representation $R = \{\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}\}$ of a stable D-T system, where:

$$R = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} = \left[\begin{array}{cccc|cc} -.15 & -.1 & .1 & -.05 & 1 & -.5 \\ .05 & -.3 & .1 & -.05 & 1 & .5 \\ .05 & .1 & -.2 & .05 & 0 & .5 \\ -.05 & -.1 & -.1 & -.35 & 0 & .5 \\ \hline -1 & 2 & 0 & 1 & 0 & 0 \\ -1 & 2 & 1 & 2 & 1 & 0 \\ 0 & 1 & 2 & 1 & 0 & 0 \end{array}\right]$$

Calculate:

(a) —the state space representation R_o in a POF using $\nu = \{1,2,1\}$,
(b) —the left coprime MFD $\{D(z), N(z)\}$ corresponding to R_o,
(c) —the transfer function matrix $G(z) = W(z)/d(z)$, and
(d) —the first 14 Markov parameters $\mathbf{H}_i$, $i=[0,13]$, in $H(z^{-1})$.

Define:

(e) —an $(m \times N)$ pseudo-random matrix $\mathbf{u}$ containing samples u_k, $k=[0,N\text{-}1]$, of the input vector $\mathbf{u}(k)$. For $\mathbf{u}(0)$ use an m-dimensional zero vector. For N use $N=31$.

Calculate:

(f) —the response $y(k)$ of R to $\mathbf{u}(k)$ with zero initial conditions.

Using the input/output pairs $\{\ \mathbf{u}(k), y(k)\ \}$, identify:

(g) —a corresponding D-T model R_o in POF using $\nu = \{1,2,1\}$,
(h) —a left coprime MFD having column degrees $\mathbf{n} = \nu$,
(i) —the first 14 Markov parameters $\mathbf{H}_i$, $i=[0,13]$ in $H(z^{-1})$,
(j) —the individual observability indices and transfer functions of $G(z) = \{g_{ij}(z)\}$, where $g_{ij}(z) = w_{ij}(z)/d(z)$.
(k) From the individual transfer functions $g_{ij}(z)$ determine $G(z)$ as: $W(z)/d(z)$.
(l) Check that the results obtained in parts (b), (c), (d) and (e) correspond to the identified models obtained in parts (g), (h), (i) and (k), respectively.

Hints:

For part (a):	To define R and ν, use the DMA operator.
For part (b):	Use subroutine SSRo.SUB.
For part (c):	Use operator SSTF.
For part (d):	Use subroutine SSH.SUB.
For part (e):	Use operators DPM and SHR.
For part (f):	Use subroutine CDSR.
For part (g):	Use subroutine uyRo.SBR.
For part (h):	Use subroutine uyDN.SBR.
For part (i):	Use subroutine uyH.SUB.
For part (j):	Use subroutine uyTF.SBR.
For part (k):	Use subroutine ComD.SBR.
For part (l):	Use operators - and OUT in the MOS, as: `A,Ai(-),B,Bi(-)....(out)=`

A version of an *L-A-S* program which solves this exercise is available in the *L-A-S* subdirectory C:\LAS\DPF\EXER51.DPF.

5.2 Given the state space representation $R = \{\mathbf{A}, \mathbf{B}, \mathbf{C}, \mathbf{D}\}$ of a stable C-T system, where:

$$R = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} = \left[\begin{array}{cccc|cc} -1.5 & -1 & 1 & -.5 & 1 & -.5 \\ .5 & -3 & 1 & -.5 & 1 & .5 \\ .5 & 1 & -2 & .5 & 0 & .5 \\ -.5 & -1 & -1 & -3.5 & 0 & .5 \\ \hline -1 & 2 & 0 & 1 & 0 & 0 \\ -1 & 2 & 1 & 2 & 1 & 0 \\ 0 & 1 & 2 & 1 & 0 & 0 \end{array}\right]$$

Note that the matrix **A** in this exercise is equal to 10 times **A** of Exercise 5.1.

Calculate:

(a) —the transfer function matrix $G(z) = W(z)/d(z)$.

Define:

(b) —an $(m \times N)$ pseudo-random matrix **u** containing values $\mathbf{u}(t_k)$, $k=[0,N\text{-}1]$, of the input vector $\mathbf{u}(t)$. For $\mathbf{u}(0)$ use an m-dimensional zero vector. For N use $N=31$.

Calculate:

(c) —the response $\mathbf{y}(t)$ of R to $\mathbf{u}(t)$ with zero initial conditions. For the "total" (simulation) time T, use $T = 10$ sec. The sampling interval is, of course, $dT = T/(N-1)$, and

(d) —the eigenvalues of $\mathbf{A}$.

Using the input/output pairs $\{ \mathbf{u}(t_k), \mathbf{y}(t_k) \}$, identify:

(e) —a corresponding D-T model R_v in POF using $\boldsymbol{\nu} = \{1,2,1\}$.

(f) From the "four" matrix D-T model R_v, determine a corresponding four matrix C-T model $R_c = \{\mathbf{A}_c, \mathbf{B}_c, \mathbf{C}_c, \mathbf{D}_c\}$ using the ramp invariant (RI) approximation.

(g) Calculate the transfer function matrix $G_c(s)$ of R_c.

(h) Calculate the response $\mathbf{y}(t)$ of R_c to $\mathbf{u}(t)$ with zero initial conditions.

(i) Find the eigenvalues of $\mathbf{A}_c$.

(j) Check that the results in parts (a), (c) and (d) correspond to the identified models in parts (g), (h) and (i), respectively.

Hints:

- To define R and $\boldsymbol{\nu}$, use operator DMA.
- The scalars N and T could be defined using DMA or "interactively" with DSC.

For part (a):	Use operator SSTF.
For part (b):	Use operators DPM and SHR.
For part (c):	Use subroutine CDSR.SUB.
For part (d):	Use operator EGV.
For part (e):	Use subroutine uyRo.SBR.
For part (f):	Use either subroutine CTDT.SBR, with *Isrb*=-2, or a corresponding sequence of operators: LNM, EATF, ... and the subroutine R5R4. Also, subroutine SRDC.SBR could be used.

A version of an *L-A-S* program which solves this exercise is available in the *L-A-S* subdirectory C:\LAS\DPF\EXER52.DPF.

This Page Intentionally Left Blank

Appendix A Matrix Algebra

In this appendix we will review a few basic ideas in dealing with vectors and matrices. It is assumed that the reader is already familiar with the concepts and needs only a brief review of the topics.

A.1 Linear Equations

Consider a common problem in analysis, namely that of solving a set of simultaneous linear algebraic equations:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n} &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ \cdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n &= b_n \end{aligned} \tag{A.1}$$

We may represent the set of equations in Eq.(A.1) in "matrix-vector" form:

$$\mathbf{A}\mathbf{x} = \mathbf{b} \tag{A.2}$$

where **A** is the array (*matrix*) of coefficients:

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ & \cdots & & \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} \tag{A.3}$$

and both **x** and **b** are *vectors*:

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} , \qquad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix} \tag{A.4}$$

The matrix **A** is said to have dimensions $m \times n$, the number of rows by the number of columns. Similarly, **x** is an $n \times 1$ matrix, or an n-dimensional (column) vector; and, **b** is an m-vector (for short).

Typically, the problem is to find, or solve for, **x** which satisfies Eq.(A.1) when both **A** and **b** are known arrays. An elementary case to begin with is $n = m$. In this case **A** is a "square" matrix, say $n \times n$, and the "solution" is:

$$\mathbf{x} = \mathbf{A}^{-1}\mathbf{b} \tag{A.5}$$

Equation (A.5) assumes that the matrix **A** is "invertible," or "non-singular," which is true if the determinant of **A** is not equal to zero.

A useful interpretation of Eq.(A.2) is to think of the matrix **A** as an "operator" that transfers, or maps, the vector **x** to the vector **b**. As such, then, for a solution **x** to exist, **b** must be in the *range* of **A**, the set of vectors that are images of some vector under the mapping **A**. Any vector **y** in the range of **A** can be written as a *linear combination* of the columns of **A**; that is,

$$\mathbf{y} = c_1\mathbf{a}_1 + c_2\mathbf{a}_2 + \cdots + c_n\mathbf{a}_n \tag{A.6}$$

where

$$\mathbf{A} = \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \cdots & \mathbf{a}_n \end{bmatrix}$$

The $\mathbf{a}_i$ are columns of **A**, i.e. m-vectors; and, the c_i are appropriate (constant) coefficients, for $i = 1, 2, \ldots, n$. The concept of "linear combination," as in Eq.(A.6), is used to formulate the following definition.

Definition A.1: A set of vectors is *linearly independent* if no vector in the set can be written as a linear combination of the others.

Example A.1: Linear Independence

Consider the matrix **A** given by

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

It is straightforward to show that there are no constants c_1 and c_2 such that any one column of **A** is a linear combination of the other two. Consequently, the column vectors of **A** are linearly independent.

Returning to Eq.(A.2), we can say that a solution **x** exists if and only if **b** is linearly *dependent* on the columns of **A**. In fact, we already know that for a square matrix **A**, Eq.(A.5) holds when the columns of **A** are linearly independent. This is due to the equivalence between (linearly) independent columns of a square matrix and the fact that the $\det(\mathbf{A}) \neq 0$. More generally, we present the following definition:

Definition A.2: The *rank* of a matrix **A** equals the number of linearly independent columns of **A**.

A recommended technique for determining the rank of a matrix is to use "row reduction" on the array to "zero out" the elements below the diagonal. It is then easy to determine the number of linearly independent columns (or rows) by inspection.

Finally, we can summarize with the following statement:

Remark: In the set of equations represented by Eqs.(A.1), or (A.2), a solution **x** exists if and only if:

$$\mathrm{rank}[\,\mathbf{A} \mid \mathbf{b}\,] = \mathrm{rank}[\,\mathbf{A}\,] \tag{A.7}$$

i.e. the rank of **A** is not changed by adding the extra column **b**.

Let us now return to the original problem. There are two important cases to consider; namely, when the number of equations in Eq.(A.1) is greater than, or less than, the number of unknowns (components of **x**):

(1) **Overdetermined equations:** $m > n$, or
(2) **Underdetermined equations:** $m < n$.

Case 1: Overdetermined Equations

For this case, $m > n$, there are more equations than unknowns. This is often true when, e.g. multiple measurements are taken to overcome measurement inaccuracies. Typically, this set of equations may even be inconsistent in that **b** is *not* in the range of **A**, as "required" by Eq.(A.7). Because Eq.(A.7) is not satisfied, there is, strictly speaking, no solution; however, even in this situation a "best," or "closest," solution can be calculated.

The so-called *least squares solution* can be derived by premultiplying Eq.(A.2) on the left by the transpose of **A**, and then inverting $(\mathbf{A}^T\mathbf{A})$ to obtain:

$$\hat{\mathbf{x}} = [\,(\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\,]\,\mathbf{b} \qquad \blacksquare\text{(A.8)}$$

Eq.(A.8) assume that **A** is full column rank. Since there is normally no exact solution, $\hat{\mathbf{x}}$ is the "least squared error solution" in the sense that

$$\min_{\mathbf{x}} \|\mathbf{A}\mathbf{x} - \mathbf{b}\| = \|\mathbf{A}\hat{\mathbf{x}} - \mathbf{b}\| \tag{A.9}$$

where the "error," $\mathbf{e} = \mathbf{A}\mathbf{x} - \mathbf{b}$, is the equation error. The norm is the Euclidian norm, i.e.

$$\|\mathbf{e}\| = \left[\sum_{i=1}^{m} e_i^2\right]^{1/2} \tag{A.10}$$

Thus, $\hat{\mathbf{x}}$ is the "solution" that most nearly reduces Eq.(A.2) to an equality, even though no **x** will do it exactly. For this reason the solution is known as a "least squares" solution because $\hat{\mathbf{x}}$ minimizes Eq.(A.9), which is equivalent to minimizing the sum of the squares of the components of the error vector $\mathbf{e}$. Notice that the factor in the brackets in Eq.(A.8) serves as $\mathbf{A}^{-1}$ and, for this reason, is called a *pseudo-inverse* of $\mathbf{A}$. See the Glossary of Symbols for pseudo-inverse matrices.

Example A.2: Least Squares Solution of Overdetermined Equations

Consider the set of equations given by

$$\begin{aligned} \alpha_1 + \alpha_2 &= 2 \\ -\alpha_1 + \alpha_2 &= -2 \\ \alpha_2 &= 3 \end{aligned}$$

In matrix form we have

$$\begin{bmatrix} 1 & 1 \\ -1 & 1 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \end{bmatrix} = \begin{bmatrix} 2 \\ -2 \\ 3 \end{bmatrix}$$

Applying Eq.(A.8), we first find that

$$\mathbf{A}^T\mathbf{A} = \begin{bmatrix} 2 & 0 \\ 0 & 3 \end{bmatrix}$$

confirming that it is invertible. Completing the solution,

$$\begin{bmatrix} \hat{\alpha}_1 \\ \hat{\alpha}_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix}$$

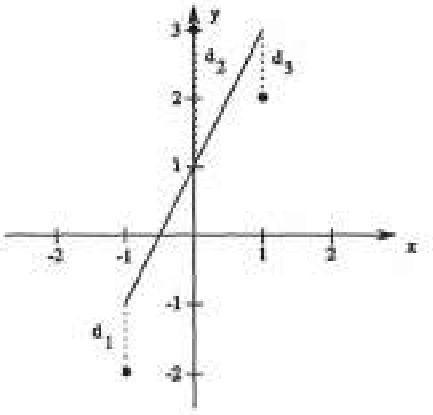


FIGURE A.1 Graphical Solution to Example A.2

This solution may be interpreted graphically if we think of the original equations as measurements relating an x variable with a y variable as follows:

$$y(x) = \alpha_1 x + \alpha_2$$

Having solved for α, we now have the "best" straight line fit. The solution is illustrated graphically in Fig. A.1, showing the "fit" as a line with slope=2 and intercept=1. It may be verified that this solution is identical with the solution to minimizing

$$J = d_1^2 + d_2^2 + d_3^2$$

where the distances d_i, $i=[1,3]$ are shown in Fig A.1, e.g. $d_1 = -2 - y(-1) = -2 + \alpha_1 - \alpha_2$, which can be done by setting the partial derivatives with respect to α_1 and α_2 to zero and solving simultaneously the two resulting equations.

Case 2: Underdetermined Equations

For this case, $m < n$, there are more unknowns than equations. Again we will assume that the matrix $\mathbf{A}$ of Eq.(A.2) is full rank, that is, rank($\mathbf{A}$) = m, the smaller dimension. This means that all dependent equations have been eliminated. We now assume an arbitrary vector

$$\mathbf{x} = \mathbf{x}_1 + \mathbf{x}_2 \tag{A.11}$$

where $\mathbf{x}_1$ is in the range of $\mathbf{A}^T$ and $\mathbf{x}_2$ is in the null space of $\mathbf{A}$, that is, $\mathbf{A}\mathbf{x}_2 = \mathbf{0}$. In particular, if $\mathbf{r}_j$ is a $1 \times n$ array representing the j^{th} row of $\mathbf{A}$, then $\mathbf{r}_j\mathbf{x}_2 = \mathbf{0}$ for j=[1,m]; and, $\mathbf{x}_1$ is some linear combination of the rows taken as vectors ($\mathbf{x}_1$ being in the range of $\mathbf{A}$):

$$\mathbf{x}_1 = \sum_{j=1}^{m} v_j r_j^T = \mathbf{A}^T\mathbf{v} \tag{A.12}$$

Thus, if

$$\mathbf{A}\mathbf{x}_1 = \mathbf{b}$$

then

$$\mathbf{A}\mathbf{A}^T\mathbf{v} = \mathbf{b} \qquad \text{or} \qquad \mathbf{v} = (\mathbf{A}\mathbf{A}^T)^{-1}\mathbf{b}$$

and, finally,

$$\hat{\mathbf{x}} = \mathbf{x}_1 = \left[\mathbf{A}^T(\mathbf{A}\mathbf{A}^T)^{-1}\right]\mathbf{b} \qquad ■ \tag{A.13}$$

Equation (A.13) is the second form of solution we desired. In this case $\mathbf{x}_1$ is the (unique) orthogonal projection of $\mathbf{x}$ onto the range space of $\mathbf{A}$, and thereby represents the *minimum-norm* vector that satisfies Eq.(A.2). In other words, there are *many* exact solutions, and we are selecting that one that has minimum length.

Example A.3: Minimum Norm Solution for Underdetermined Equations

Given the set of two equations in four unknowns

$$x_1 - x_2 + x_3 - 2x_4 = 1$$
$$2x_2 + x_3 - x_4 = 2$$

we will first solve for x_1 and x_2 in terms of x_3 and x_4 and, second, determine the solution according to Eq.(A.13). The results of the first step are:

$$x_1 = \frac{1}{2}(4 - 3x_3 + 5x_4)$$

$$x_2 = \frac{1}{2}(2 - x_3 + x_4)$$

For the second part we rewrite the original equations in vector form:

$$\begin{bmatrix} 1 & -1 & 1 & -2 \\ 0 & 2 & 1 & -1 \end{bmatrix} \mathbf{x} = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$$

In applying Eq.(A.13), we first calculate

$$\mathbf{A}\mathbf{A}^T = \begin{bmatrix} 7 & 1 \\ 1 & 6 \end{bmatrix}$$

Completing the indicated operations, the minimum-norm solution is

$$\hat{\mathbf{x}} = \frac{1}{41}[\ 4 \ \ 22 \ \ 17 \ \ -21\ ]^T$$

The norm, or length, of this vector is

$$\| \hat{\mathbf{x}} \| = 0.855$$

and *any* other of the many solutions will be longer. For example, setting both x_3 and x_4 to zero, we obtain the solution

$$\mathbf{x}' = [\ 2 \ \ 1 \ \ 0 \ \ 0\ ]^T$$

whose norm is $\| \mathbf{x}' \| = 2.236$.

A.2 Eigensystems

The *eigenvalues* λ and *eigenvectors* $\mathbf{e}$ of a (square) matrix $\mathbf{A}$ must satisfy that

$$\mathbf{A}\mathbf{e} = \lambda\mathbf{e} \tag{A.14}$$

Eq.(A.14) may be rewritten as

$$(\mathbf{A} - \lambda\mathbf{I})\mathbf{e} = \mathbf{0} \tag{A.15}$$

In order that a nontrivial solution ($\mathbf{e} \neq \mathbf{0}$) exist, $(\mathbf{A} - \lambda\mathbf{I})$ must be singular; that is, $\det(\mathbf{A} - \lambda\mathbf{I}) = 0$. However, if $\mathbf{A}$ is an $n \times n$ matrix, then there are n (possibly some repeated) roots of this n^{th} order polynomial equation. These roots are called the *eigenvalues* of $\mathbf{A}$. Corresponding to each distinct eigenvalue, there is at least a one dimensional solution $\mathbf{e}$ to Eq.(A.14), called an *eigenvector*. The collection of eigenvalues and corresponding eigenvectors is called the *eigensystem* of $\mathbf{A}$.

Whenever there are n distinct eigenvalues for an $n \times n$ matrix, there will be n linearly independent eigenvectors. By collecting these eigenvectors to form an $n \times n$ matrix $\mathbf{E}$, we can write from Eq.(A.14) that

$$\mathbf{A}\mathbf{E} = \mathbf{E}\mathbf{\Lambda} \quad \text{or} \quad \mathbf{E}^{-1}\mathbf{A}\mathbf{E} = \mathbf{\Lambda} \tag{A.16}$$

where $\mathbf{E} = [\mathbf{e}_1 \ \ \mathbf{e}_2 \ \ldots \ \mathbf{e}_n]$ is called the *modal matrix* of $\mathbf{A}$ and $\mathbf{\Lambda} = \mathrm{diag}\{\lambda_1 \ \ldots \ \lambda_n\}$

is the diagonal (or Jordan) form of **A**. The relation between **A** and **Λ** through the matrix **E**, Eq.(A.16), is known as a *similarity transformation*.

Special Cases:

- If **A** is a symmetric matrix, i.e. $a_{ij} = a_{ji}$, then there will always exist n linearly independent eigenvectors.
- The eigenvectors of a symmetric matrix **A** are mutually orthogonal:

$$\mathbf{e}_i^T \mathbf{e}_j = 0 \quad \text{for} \quad i \neq j \tag{A.17}$$

- Generally, when **A** has repeated eigenvalues, there will not be n linearly independent eigenvectors, and **A** cannot be "diagonalized." A generalization is the Jordan form, which is block diagonal; see Section 3.1.3 and Appendix B.

Example A.4: A Modal Matrix

Determine the modal matrix **P** for the matrix **A** given below and show that:

$$\mathbf{P}^{-1}\,\mathbf{A}\,\mathbf{P} = \mathbf{J}$$

where **J** is a diagonal matrix.

$$\mathbf{A} = \begin{bmatrix} 1.5 & 0 & -.5 \\ 0 & 1 & 0 \\ -.5 & 0 & 1.5 \end{bmatrix}$$

(1) Eigenvalues (roots of $\det[\lambda\mathbf{I} - \mathbf{A}] = 0$):

$$\begin{vmatrix} \lambda - 1.5 & 0 & .5 \\ 0 & \lambda - 1 & 0 \\ .5 & 0 & \lambda - 1.5 \end{vmatrix} = (\lambda - 1)^2 (\lambda - 2)$$

Therefore, the set of eigenvalues are $\{ 1, 1, 2 \} = \{ \lambda_i \}$.

(2) Eigenvectors (nontrivial solutions $\mathbf{v}_i$ of $[\lambda_i\mathbf{I} - \mathbf{A}]\mathbf{v}_i = \mathbf{0}$):

For $\lambda_1 = 1$:

$$\begin{bmatrix} -.5 & 0 & .5 \\ 0 & 0 & 0 \\ .5 & 0 & -.5 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

We find that the rank of the coefficient matrix (dimension of the largest non-zero determinant) is 1, therefore, n-1=2 linearly independent vector solutions. The constraints on the components v_i are:

$$v_1 = v_3 \quad \text{and} \quad v_2 \text{ is arbitrary.}$$

We choose two linearly independent solutions, say

$$\mathbf{v}_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \quad \mathbf{v}_2 = \begin{bmatrix} 1 \\ 0 \\ 1 \end{bmatrix}$$

For $\lambda_3 = 2$:

$$\begin{bmatrix} .5 & 0 & .5 \\ 0 & 1 & 0 \\ .5 & 0 & .5 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

We find that the rank of the coefficient matrix is 2, (therefore n-2=1 independent vector solution). The constraints on the components v_i are:

$$v_1 = -v_3 \quad \text{and} \quad v_2 = 0.$$

We choose

$$\mathbf{v}_3 = \begin{bmatrix} 1 \\ 0 \\ -1 \end{bmatrix}$$

(3) Modal matrix: $\mathbf{P} = [\ \mathbf{v}_1 \ \ \mathbf{v}_2 \ \ \mathbf{v}_3\ ]$:

$$\mathbf{P} = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & -1 \end{bmatrix}, \quad \mathbf{P}^{-1} = \begin{bmatrix} 0 & 1 & 0 \\ .5 & -1 & .5 \\ .5 & 0 & -.5 \end{bmatrix}$$

(4) From (3) it is easy to show that

$$\mathbf{J} = \mathbf{P}^{-1}\mathbf{A}\mathbf{P} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix} = \mathrm{diag}\{\ \lambda_1,\ \lambda_2,\ \lambda_3\ \}$$

A.3 Rank and Null Space

Two important concepts in dealing with matrices are: (1) the *range space* of the matrix, and (2) the *null space* of the matrix. We think of an $m \times n$ matrix as a transformation of vectors in an n-dimensional *domain* space into vectors in an m-dimensional *range* space, just as a mathematical function can map, or transform, values in x-space into values in y-space. Consider the following definitions:

Definition A.3: The *range space* of the $m \times n$ matrix $\mathbf{A}$ is the collection of m-dimensional vectors $\mathbf{y}$, such that $\mathbf{A}\,\mathbf{x} = \mathbf{y}$ for some n-dimensional vector $\mathbf{x}$.

Definition A.4: The *null space* of the $m \times n$ matrix $\mathbf{A}$ is the collection of n-dimensional vectors $\mathbf{x}$, such that $\mathbf{A}\,\mathbf{x} = \mathbf{0}$.

The first of the previous two definitions is directly related to the columns of $\mathbf{A}$ in that a vector $\mathbf{y}$ in the range space must be a linear combination of the columns of $\mathbf{A}$, with the components of $\mathbf{x}$ as coefficients. In the second of the two definitions, we see that the null space contains the vectors $\mathbf{x}$ that are mapped to $\mathbf{0}$. It may be shown that both the range and the null space are "subspaces," i.e. their vectors are closed under vector addition and scalar multiplication. It is useful to have special terms for the dimensions of these two spaces associated with the matrix $\mathbf{A}$. Thus, we have:

Definition A.5: The *rank* of the $m \times n$ matrix $\mathbf{A}$ is the dimension of its range space.

Definition A.6: The *nullity* of the $m \times n$ matrix $\mathbf{A}$ is the dimension of its null space.

Although complicated operations for matrices are implemented easily in *L-A-S*, it is instructive to apply the concepts "manually" to an example.

Example A.5: Range and Null Space Calculations

Determine the rank and nullity of the 3×5 matrix $\mathbf{A}$ given by

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 2 & 3 & 4 & 1 & 2 \\ 3 & 4 & 5 & 0 & 0 \end{bmatrix}$$

and calculate a set of basis vectors (a linearly independent set of vectors which span the space, i.e. any vector in the space can be written as a linear combination of the basis vectors) for both the range and the null spaces.

(1) Using elementary row operations, it can be shown that $\mathbf{A}$ can be reduced to

$$\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 2 & 7 & 8 \\ 0 & 0 & 0 & 2 & 1 \end{bmatrix}$$

Therefore, the rank is 3, the number of independent columns, say columns 1, 2 and 4. The nullity is the number of columns less the rank, 5 - 3 = 2.

(2) As mentioned in (1), columns 1, 2 and 4 are linearly independent; therefore, they could serve as a basis of the 3-dimensional range space. As a check on their linearly independence, let us calculate the non-zero determinant of these three columns:

$$\begin{vmatrix} 1 & 2 & 4 \\ 2 & 3 & 1 \\ 3 & 4 & 0 \end{vmatrix} = -2 \neq 0$$

(3) To calculate a set of 2 linearly independent vectors which can serve as a basis set for the null space, the row reduced version of **A** given above will be used:

$$\begin{bmatrix} 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 2 & 7 & 8 \\ 0 & 0 & 0 & 2 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Thus, the constraints on the components x_i, taken from these three scalar equations can be rewritten in terms of, say, x_3 and x_4 as follows:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} -x_3 - 3x_4 \\ -2x_3 + 9x_4 \\ x_3 \\ x_4 \\ -2x_4 \end{bmatrix}$$

Choosing [1 0] and [0 1] for [x_3 x_4] respectively, we obtain the following basis set for the null space:

$$\mathbf{x}_1 = \begin{bmatrix} -1 \\ -2 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad \text{and} \quad \mathbf{x}_2 = \begin{bmatrix} -3 \\ 9 \\ 0 \\ 1 \\ -2 \end{bmatrix}$$

A.4 Singular Value Decomposition (SVD)

Singular value decomposition can be thought of as a generalization of the eigensystem calculation for matrices which are rectangular. Consider a matrix **A** which is $(m \times n)$ and has rank r. We will assume that $m \geq n$, although the development applies equally well if the reverse is true. The objective is to represent **A** as

$$\mathbf{A} = \mathbf{U}\,\mathbf{W}\,\mathbf{V}^T \tag{A.18}$$

where **W** is an $(m \times n)$ diagonal matrix, i.e. the elements w_{ij}, $i \neq j$, are zero; and **U** and **V** are $(m \times m)$ and $(n \times n)$ orthogonal matrices, respectively.

We first note that $\mathbf{A}\mathbf{A}^T$ is an $(m \times m)$ positive semi-definite (symmetric) matrix, and that $\mathbf{A}^T\mathbf{A}$ is likewise an $(n \times n)$ positive semi-definite (symmetric) matrix. Consequently, we can find a set of orthonormal "left singular vectors" $\mathbf{u}_i$, $i=[1, m]$ and a set of orthonormal "right singular vectors" $\mathbf{v}_i$, $i=[1, n]$ from these two symmetric matrices. Thus,

$$\mathbf{A}\,\mathbf{A}^T\,\mathbf{u}_i = \sigma_i^2\,\mathbf{u}_i \quad \text{and} \quad \mathbf{A}^T\mathbf{A}\,\mathbf{v}_j = \lambda_j^2\,\mathbf{v}_j \tag{A.19}$$

for $i=[1, m]$ and $j=[1, n]$. The use of "squares" of the singular vallues is justified since both products are positive semi-definite, i.e. they have only non-negative eigenvalues. It can be shown (See Section A.5) that the non-zero eigenvalues of **AB** and **BA** are equal, so that the (non-zero) $\{\sigma_i\}$ equal the (non-zero) $\{\lambda_i\}$. In addition, since $\mathbf{A}^T\mathbf{u}_i = \mathbf{z}_i$ can be shown to be an eigenvector of $\mathbf{A}^T\mathbf{A}$, we have from Eq.(A.19)

$$\mathbf{A}\,\mathbf{z}_i = \sigma_i^2\,\mathbf{u}_i \tag{A.20}$$

But, unlike $\mathbf{u}_i$, the vector $\mathbf{z}_i$ is not necessarily a unit vector. In fact, for a non-zero σ_i, using a symbolic scalar product notation, we find the length of $\mathbf{z}_i$ to be σ_i, i.e.

$$|\mathbf{z}_i|^2 = (\mathbf{A}^T\mathbf{u}_i, \mathbf{A}^T\mathbf{u}_i) = (\mathbf{A}\mathbf{A}^T\mathbf{u}_i, \mathbf{u}_i) = \sigma_i^2\,(\mathbf{u}_i, \mathbf{u}_i) = \sigma_i^2$$

Since from Eq.(A.19) $\mathbf{A}^T\mathbf{A}\,\mathbf{v}_i = \sigma_i^2\,\mathbf{v}_i$, $(\lambda_i = \sigma_i)$, and $\mathbf{A}^T\mathbf{A}\,\mathbf{z}_i = \sigma_i^2\,\mathbf{z}_i$, it follows that $\mathbf{z}_i = \sigma_i\mathbf{v}_i$. Thus, dividing Eq.(A.20) by σ_i, and concatenating all the resulting equations,

$$\mathbf{A}\,[\,\mathbf{v}_1\;\;\mathbf{v}_2\;\cdots\;\mathbf{v}_n\,] = [\,\mathbf{u}_1\;\;\mathbf{u}_2\;\cdots\;\mathbf{u}_m\,]\left[\begin{array}{cccc} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_n \\ \hline & & \mathbf{0} & \end{array}\right] \tag{A.21}$$

Finally, since $\mathbf{V}^{-1} = \mathbf{V}^T$, because its columns $\{\mathbf{v}_j\}$ form an orthonormal set, the objective of Eq.(A.18) is attained. The last columns of **U**, and the corresponding zeros of **W** can be deleted, in this case **U**, **W** and $\mathbf{V}^T$ become $(m \times n)$, $(n \times n)$ and $(n \times n)$, matrices, respectively. Also, when **A** is less than full rank, the final rows of $\mathbf{V}^T$ and the corresponding zero columns of **W** can be omitted leaving **U**, **W** and $\mathbf{V}^T$ as $(m \times r)$, $(r \times r)$ and $(r \times n)$, matrices, respectively.

We will now present two related algorithms. The first, *NRS*, calculates the range and null spaces of a rectangular matrix. The second, *INOU*, provides a decomposition of a matrix **Q** into two subspaces: **Qr**, the projection of **Q** *into* the range space of **R**; and **Qou**, the projection of **Q** to the subspace *outside* of the range space of **R**.

Algorithm: *NRS*

Syntax: $\mathbf{A}$, *eps* (*NRS*) $\Rightarrow \mathbf{N}, \mathbf{R}, r$

Purpose: The calculation of the rank and the range and null space matrices of an $(m \times n)$ matrix **A**.

Input/Output Arguments:

- $\mathbf{A}$ = $(m \times n)$ matrix.
- *eps* = small positive scalar, suggested value; $eps = 10^5$.
- $\mathbf{N}$ = $(n \times n\text{-}r)$ null space matrix of **A**, where $\mathbf{AN} = \mathbf{0}_{m,(n-r)}$.
- $\mathbf{R}$ = $(m \times r)$ range space matrix of **A**, where $\rho(\mathbf{R}) = \rho(\mathbf{A})$.
- r = rank of the matrix **A**, $r = \rho(\mathbf{A})$, where $r \leq \min(m,n)$.

Description:

The rank of an $(m \times n)$ matrix **A** is defined as:

(i) the size of the largest non-vanishing determinant that can be formed from **A**, or
(ii) the maximum number of linearly independent columns, or rows, in **A**.

The range and null space matrices, **R** and **N** satisfy:

$$r = \rho(\mathbf{R}) = \rho(\mathbf{A})\ , \quad \mathbf{AN} = \mathbf{0}_{m,(n-r)} \tag{A.22}$$

The easiest and computationally most reliable way of calculating r, **R** and **N** consists of performing the singular-value decomposition (SVD) of the matrix **A**, i.e. decomposing **A** into:

$$\mathbf{A} = \mathbf{UWV}^T \tag{A.23}$$

where $(m \times n)$, $(n \times n)$ and $(n \times n)$ matrices **U**, **W** and **V** are given by:

$$\mathbf{U} = [\,\mathbf{u}_1 \;\cdots\; \mathbf{u}_r \;\cdots\; \mathbf{u}_n\,]\,,\quad \mathbf{V} = [\,\mathbf{v}_1 \;\cdots\; \mathbf{v}_r \;\cdots\; \mathbf{v}_n\,] \qquad \text{and} \quad \mathbf{W} = \mathrm{diag}\{\,\sigma_1, \cdots, \sigma_r, \cdots, \sigma_n\} \tag{A.24}$$

Matrices **U** and **V** are "unitary," i.e. columns $\mathbf{u}_i$ and $\mathbf{v}_i$, $i=[1,n]$, are orthonormal, i.e.:

$$\mathbf{U}^T\mathbf{U} = \mathbf{V}^T\mathbf{V} = \mathbf{I}_n \tag{A.25}$$

The positive scalars σ_i, $i=[1,n]$ are referred to as the *singular values of* **A**.

All SVD calculation procedures arrange the scalars σ_i in decreasing order, i.e.:

$$\sigma_i \geq \sigma_{i+1}\,, \quad i=[1,\, n-1] \tag{A.26}$$

If the positive scalar *eps* satisfies: *eps* $<<$ 1, then for all practical purposes the rank of **A** may be defined as the index r of the singular value σ_{r+1} satisfying:

$$\sigma_{r+1} \leq eps \tag{A.27}$$

or, the total number of σ_i satisfying $\sigma_i > eps$, $i=[1,r]$.

Thus, the range and null space matrices **R** and **N** could be defined by partitioning **U** and **V** as follows:

$$\mathbf{U} = [\,\mathbf{R} \mid \mathbf{X}\,] \quad \text{and} \quad \mathbf{V} = [\,\mathbf{Y} \mid \mathbf{N}\,] \tag{A.28}$$

where:

- **R** contains the first r columns $\mathbf{u}_i$ from **U**, $i = [1,r]$, while
- **N** contains the last n-r columns $\mathbf{v}_i$ from **V**, $i=[r+1,n\text{-}r]$.

The integer $\nu = n\text{-}r$, representing the dimension of the null space of **A**, is referred to as the *nullity of* **A**. The SVD of **A** is performed by the algorithm *SVD*:

Syntax: **A** (*SVD*) $\Rightarrow$ **w**, **U**, **V**

where the $(1 \times n)$ row **w** contains the singular values σ_i, $i=[1,n]$.

Algorithm:

```
1. Define the $(m \times n)$ matrix $\mathbf{A}$ and the scalar $eps$
2. Set $\mathbf{A}$ ($SVD$) $\Rightarrow \mathbf{w}, \mathbf{U}, \mathbf{V}$
3. If $\sigma_i > eps$, set $\sigma_i/\sigma_i = 1 \Rightarrow x_i$ ; else, set $0 \Rightarrow x_i$
4. Set $\mathbf{x}\,\mathbf{x}^T \Rightarrow r$
5. Set $\mathbf{U} \Rightarrow [\,\mathbf{R} \mid \mathbf{X}\,]$
6. Set $\mathbf{V} \Rightarrow [\,\mathbf{Y} \mid \mathbf{N}\,]$
7. Stop
```

Algorithm Implementation:

The listing of Algorithm *NRS*, implemented using the *L-A-S* language is given in Appendix C. Algorithm *SVD* is performed using the *L-A-S* operator SVD. The *m*-dimensional row **x** containing *r* unities and *m-r* zeros is calculated by the operator F/. The rank of **A** is then obviously given by $r = \mathbf{x}\,\mathbf{x}^T$. The partitioning of **U** and **V** in Steps 5 and 6 is done using the *L-A-S* operator CTC.

Algorithm: *INOU*

Syntax: **R**, **Q**, *eps* (*INOU*) = **Qr**, **Qou**

Purpose: To decompose the matrix **Q** into two subspaces **Qr** (the projection of **Q** into **R**) and **Qou** (the subspace outside the range space of **R**).

Input/Output Arguments:

- **R** = $(n \times m)$ matrix
- **Q** = $(n \times k)$ full column rank matrix, i.e. $\rho(\mathbf{Q}) = k$
- *eps* = small positive scalar, suggested value: $eps = 10^{-5}$
- **Qr** = $(n \times r)$ matrix; projection of **Q** into **R**
- **Qou** = $(n \times s)$ matrix; part of **Q** outside the range space of **R**

Description:

Matrices **Qr** and **Qou** satisfy:

$$\rho[\ \mathbf{R}\ |\ \mathbf{Qr}\ ] = \rho[\ \mathbf{R}\ ]\ ,\quad \rho[\ \mathbf{Q}\ |\ \mathbf{Qr}\ ] = \rho[\ \mathbf{Q}\ ]\ ,\quad \rho[\ \mathbf{Qr}\ ] = r$$

and (A.29)

$$\rho[\ \mathbf{R}\ |\ \mathbf{Qou}\ ] = \rho[\ \mathbf{R}\ ] + s\ ,\quad \rho[\ \mathbf{Q}\ |\ \mathbf{Qou}\ ] = \rho[\ \mathbf{Q}\ ]\ ,\quad \rho[\ \mathbf{Qou}\ ] = s$$

Since $\rho(\mathbf{Q}) = k$, then $r + s = k$.

The matrices **Qr** and **Qou** are obtained by postmultipying **Q** with $(k \times r)$ and $(k \times s)$ matrices $\mathbf{N}_r$ and $\mathbf{N}_{qn}$, i.e.:

$$\mathbf{Qr} = \mathbf{Q}\,\mathbf{N}_q \quad \text{and} \quad \mathbf{Qou} = \mathbf{Q}\,\mathbf{N}_{qn} \tag{A.30}$$

where $\mathbf{N}_q$ and $\mathbf{N}_{qn}$ are calculated from null space matrices:

$$[\ \mathbf{Q}\ \ |\ \ \mathbf{R}\ ]\begin{bmatrix}\mathbf{N}_q\\ \mathbf{N}_r\end{bmatrix} = \mathbf{0} \quad \text{and} \quad \mathbf{N}_q^T\mathbf{N}_{qn} = \mathbf{0} \tag{A.31}$$

From Eq.(A.31) it is evident that **Qr** = **Q** $\mathbf{N}_q$ is in the range space of **R** since:

$$\mathbf{Q}\mathbf{N}_q = -\mathbf{R}\mathbf{N}_r$$

Similarly, since [$\mathbf{N}_q$ | $\mathbf{N}_{qn}$] is a $(k \times k)$ nonsingular matrix, it follows that **Qou** = **Q** $\mathbf{N}_{qn}$ is outside the range space of **R**.

The null space matrices required in Eq.(A.31) are calculated by the algorithm *NRS* discussed above.

Algorithm:

```
1. Define $(n \times m)$ and $(n \times k)$ matrices R and Q and the scalar eps.
2. Set number of columns in Q ⇒ $k$
3. Set [ Q | R ] ⇒ QR
4. Set QR, eps (NRS) ⇒ $N_{qr}$, Y, x
5. Set the first $k$ rows from $N_{qr}$ ⇒ $N_q$
6. Set $N_q^T$, eps (NRS) ⇒ $N_{qn}$, Y, x
7. Set Q $N_q$ ⇒ Qr
8. Set Q $N_{qn}$ ⇒ Qou
9. Stop
```

Algorithm Implementation

The listing of Algorithm *INOU*, implemented using the *L-A-S* language is given in Appendix C. Algorithm *NRS* is performed using the *L-A-S* operator NRS. The subroutine version is given in Appendix C. Matrix transposition is performed by the *L-A-S* operator T.

A.5 Useful Results with Matrices

Consider the two matrices

$$\mathbf{C} = \mathbf{I} + \mathbf{AB} \quad \text{and} \quad \mathbf{D} = \mathbf{I} + \mathbf{BA} \tag{A.32}$$

where **C** has dimensions $n \times n$ and **D**, $m \times m$. It follows from Eq.(A.15) that there is a one-to-one correspondence between the eigenvalues of **C** and those of **AB**. Thus, if λ is an eigenvalue of **AB**, then $(1+\lambda)$ is an eigenvalue of **C**, and similarly for **D** and **BA**.

If we assume that $n > m$, then a non-zero eigenvalue λ of **BA** satisfies that

$$\mathbf{BAx} = \lambda\mathbf{x} \tag{A.33}$$

for some corresponding (non-zero) eigenvector **x**. Premultiplying by **A**,

$$\mathbf{AB}(\mathbf{Ax}) = \lambda(\mathbf{Ax}) \tag{A.34}$$

which shows that λ is automatically an eigenvalue of **AB** (with corresponding eigenvector **Ax**). Consequently, the m eigenvalues of **BA** are also eigenvalues of **AB** and vice versa, so that the remaining $(n - m)$ eigenvalues of **AB** must be zero.

Applying this result to **C** and **D** of Eq.(A.32), the m eigenvalues of **D** are also eigenvalues of **C**, and the remaining eigenvalues of **C** are unity. Therefore, the product of the eigenvalues of **C** equals the product of the eigenvalues of **D**; or, in other words, the determinants are equal:

$$\det(\mathbf{I} + \mathbf{AB}) = \det(\mathbf{I} + \mathbf{BA}) \quad \blacksquare \tag{A.35}$$

Partitioned Matrices: Using the Laplace expansion of a determinant, it is readily shown that:

$$\det(\mathbf{AB}) = (\det \mathbf{A})(\det \mathbf{B}) \quad \blacksquare \tag{A.36}$$

$$\det(\mathbf{A}) = \det(\mathbf{A}^T) \quad \blacksquare \tag{A.37}$$

and

$$\det\begin{bmatrix} \mathbf{A} & \mathbf{0} \\ \mathbf{B} & \mathbf{C} \end{bmatrix} = \det(\mathbf{A})\det(\mathbf{C}) \quad \blacksquare \tag{A.38}$$

The fact that partitioned matrices obey the same rules as ordinary matrices with respect to multiplication and addition permits us to generate some interesting expressions for inverse matrices. Suppose that $\mathbf{B} = \mathbf{A}^{-1}$, then, assuming compatible partitions:

$$\begin{bmatrix} \mathbf{A}_1 & \mathbf{A}_2 \\ \mathbf{A}_3 & \mathbf{A}_4 \end{bmatrix}\begin{bmatrix} \mathbf{B}_1 & \mathbf{B}_2 \\ \mathbf{B}_3 & \mathbf{B}_4 \end{bmatrix} = \begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} \end{bmatrix} \tag{A.39}$$

Thus,

$$\mathbf{A}_1\mathbf{B}_1 + \mathbf{A}_2\mathbf{B}_3 = \mathbf{I}$$

and (A.40)

$$\mathbf{A}_3\mathbf{B}_1 + \mathbf{A}_4\mathbf{B}_3 = \mathbf{0}$$

are the (1,1) and (2,1) elements of the product. Solving for $\mathbf{B}_1$ and $\mathbf{B}_3$, we obtain:

$$\mathbf{B}_1 = (\mathbf{A}_1 - \mathbf{A}_2\mathbf{A}_4^{-1}\mathbf{A}_3)^{-1} \tag{A.41}$$

and

$$\mathbf{B}_3 = -\mathbf{A}_4^{-1}\mathbf{A}_3(\mathbf{A}_1 - \mathbf{A}_2\mathbf{A}_4^{-1}\mathbf{A}_3)^{-1} \tag{A.42}$$

Similarly, the remaining equations permit solving for $\mathbf{B}_2$ and $\mathbf{B}_4$:

$$\mathbf{B}_2 = -\mathbf{A}_1^{-1}\mathbf{A}_2(\mathbf{A}_4 - \mathbf{A}_3\mathbf{A}_1^{-1}\mathbf{A}_2)^{-1} \tag{A.43}$$

and

$$\mathbf{B}_4 = (\mathbf{A}_4 - \mathbf{A}_3\mathbf{A}_1^{-1}\mathbf{A}_2)^{-1} \tag{A.44}$$

completing $\mathbf{B} = \mathbf{A}^{-1}$.

Many matrix identities can be developed by repeating the above process with a reversed product order, $\mathbf{BA} = \mathbf{I}$, and equating the two expressions for $\mathbf{B}$, since $\mathbf{A}^{-1}$ is unique for any non-singular matrix $\mathbf{A}$. In particular, the *matrix inversion lemma* is

$$(\mathbf{A}^{-1} + \mathbf{C}^T\mathbf{B}\mathbf{H})^{-1} = \mathbf{A} - \mathbf{A}\mathbf{C}^T(\mathbf{C}\mathbf{A}\mathbf{C}^T + \mathbf{B}^{-1})^{-1}\mathbf{C}\mathbf{A} \qquad \blacksquare\text{(A.45)}$$

A final result can be realized using the fact that for any matrix $\mathbf{M}$

$$\det\begin{bmatrix} \mathbf{I} & \mathbf{0} \\ \mathbf{M} & \mathbf{I} \end{bmatrix} = 1 \tag{A.46}$$

Therefore, using the property of Eq.(A.36), and assuming that $\mathbf{A}$ is nonsingular,

$$\det\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} = \det\left(\begin{bmatrix} \mathbf{I} & \mathbf{0} \\ -\mathbf{C}\mathbf{A}^{-1} & \mathbf{I} \end{bmatrix}\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}\right) \tag{A.47}$$

Multiplying out the previous expression,

$$\det\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} = \det\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{0} & \mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B} \end{bmatrix} \tag{A.48}$$

Finally, using properties from Eqs.(A.37) and (A.38),

$$\det\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} = \det(\mathbf{A})\det(\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B}) \qquad \blacksquare\text{(A.49)}$$

A.6 The Cayley-Hamilton Theorem

We know from Section A.2 that a matrix $\mathbf{A}$ with distinct eigenvalues is similar to a diagonal matrix. Recall that

$$\mathbf{\Lambda} = \operatorname{diag}[\lambda_1, \lambda_2, \cdots, \lambda_n] = \mathbf{E}^{-1}\mathbf{A}\mathbf{E} \tag{A.50}$$

where $\{\lambda_i\}$, $i=[1,n]$, are the (assumed distinct) eigenvalues of the $n \times n$ matrix $\mathbf{A}$, and $\mathbf{E}$ is the modal matrix of $\mathbf{A}$ whose columns are the eigenvectors corresponding to the eigenvalues in $\mathbf{\Lambda}$.

Note that an integer power of Λ takes the form

$$\Lambda^m = (\mathbf{E}^{-1}\mathbf{A}\mathbf{E})_1(\mathbf{E}^{-1}\mathbf{A}\mathbf{E})_2 \cdots (\mathbf{E}^{-1}\mathbf{A}\mathbf{E})_m = \mathbf{E}^{-1}\mathbf{A}^m\mathbf{E} \qquad (A.51)$$

It follow that for any polynomial $p(\lambda)$, the corresponding matrix polynomial is

$$p(\mathbf{A}) = \mathbf{E}\,p(\Lambda)\,\mathbf{E}^{-1} \qquad (A.52)$$

And since Λ is a diagonal matrix,

$$p(\Lambda) = \mathrm{diag}\{\,p(\lambda_1),\, p(\lambda_2),\, \cdots,\, p(\lambda_n)\,\} \qquad (A.53)$$

For the particular polynomial which is the characteristic polynomial of **A**, we have that

$$a(\mathbf{A}) = \mathbf{E}\,a(\Lambda)\,\mathbf{E}^{-1} = \mathbf{0} \qquad (A.54)$$

since $a(\lambda_i) = 0$ for $i=[1,n]$ by the definition of eigenvalues. This result may be summarized in the statement that "the matrix **A** satisfies its own characteristic equation."

Cayley-Hamilton Theorem: If $a(\lambda) = \det[\lambda\mathbf{I} - \mathbf{A}]$ is the characteristic polynomial of the (square) matrix **A**, then $a(\mathbf{A})$ is the zero matrix.

Our development assumed distinct eigenvalues for **A**, but it can be shown that the Cayley-Hamilton Theorem is valid for any square matrix. See also Algorithm *POLR*, which is discussed at the end of the examples in Section 2.4.

A.7 References

Brogan, W.L. (1991), *Modern Control Theory, 3rd Edition*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Chen, C-T. (1984), *Linear System Theory and Design*, Holt, Rinehart and Winston, Inc., New York.

Kailath, T. (1980), *Linear Systems*, Prentice-Hall Inc., Englewood Cliffs, NJ.

VanLandingham, H.F. (1985), *Introduction to Digital Control Systems*, Macmillan Publishing Co., New York.

Appendix B Special Topics

In this appendix we discuss several items of theoretical, as well as practical, interest. Some of the topics continue to build on the review material of Appendix A. For example, the first section in this appendix concerns the problem of linear algebraic equations. Other topics include the three minimal realization techniques, Hessenberg, Kalman and Jordan. One section discusses a useful, relatively simple method of measuring the relative controllability and observability of MIMO system poles.

B.1 Linear Algebraic Equations

Consider the following system of linear algebraic equations:

$$\mathbf{A}\mathbf{X} = \mathbf{B} \tag{B.1}$$

where the $[n(m+1) \times nm]$ matrix $\mathbf{A}$ has rank $= n$, and $\mathbf{B}$ is a given $[n(m+1) \times m]$ matrix. It is known that one among many solutions for the $[nm \times m]$ matrix $\mathbf{X}$ of Eq.(B.1) has the form:

$$\mathbf{X} = \begin{bmatrix} d_0 \mathbf{I}_m \\ d_1 \mathbf{I}_m \\ \vdots \\ d_{n-1} \mathbf{I}_m \end{bmatrix} \tag{B.2}$$

Recall that this problem was encountered in Algorithm *HTF*, Section 4.3.5, Eq.(4.74). We are interested in obtaining the scalars d_i, $i=[0,n-1]$, satisfying Eqs.(B.1) and (B.2).

Procedure:

The general solution $\mathbf{X}$ of Eq.(B.1) may be written as:

$$\mathbf{X} = \mathbf{Y} + \mathbf{N}\mathbf{T} \tag{B.3}$$

where $\mathbf{Y}$ is any nontrivial solution of Eq.(B.1), $\mathbf{N}$ is the $[nm \times n(n-1)]$ null space matrix of $\mathbf{A}$ satisfying:

$$\mathbf{AN} = \mathbf{0} \tag{B.4}$$

while $\mathbf{T}$ is an arbitrary $[n(n\text{-}1) \times m]$ matrix.

For additional notation let:

$$\begin{aligned} \mathbf{I}_m &= [\ \mathbf{e}_1 \ \cdots \ \mathbf{e}_m\ ] \\ \mathbf{Y} &= [\ \mathbf{y}_1 \ \cdots \ \mathbf{y}_m\ ] \\ \mathbf{T} &= [\ \mathbf{t}_1 \ \cdots \ \mathbf{t}_m\ ] \end{aligned} \tag{B.5}$$

then, taking into account Eq.(B.2) the i^{th} column $\mathbf{x}_i$ of the general solution $\mathbf{X}$ in Eq.(B.3) may be expressed as:

$$\mathbf{x}_i = \begin{bmatrix} d_0\,\mathbf{e}_i \\ d_1\,\mathbf{e}_i \\ \vdots \\ d_{n-1}\,\mathbf{e}_i \end{bmatrix} = \mathbf{y}_i + \mathbf{N}\mathbf{t}_i \tag{B.6}$$

Eliminating from Eq.(B.6) elements with the indices:

$$i,\ i+m,\ i+2m,\ \ldots,\ i+jm,\ \ldots,\ i+(n\text{-}1)m \tag{B.7}$$

Eq.(B.6) yields a system of $(m\text{-}1)m$ equations of the form:

$$\mathbf{N}_i\,\mathbf{t}_i = -y_{ii} \tag{B.8}$$

where $\mathbf{N}_i$ and y_{ii} represent an $[(m\text{-}1)m \times (m\text{-}1)m]$ matrix and $(m\text{-}1)m$ columns obtained from $\mathbf{N}$ and y_i in Eq.(B.6) by eliminating the rows with indices given by Eq.(B.7). By definition, $\mathbf{N}$ is a full column rank matrix. However, the square matrices $\mathbf{N}_i$ obtained from $\mathbf{N}$ by eliminating n rows are not necessarily of full rank for each i. But, according to the assumption of Eq.(B.2), it may be concluded that there is some index i, $i=[1,m]$, for which the matrix $\mathbf{N}_i$ is nonsingular. Using this $\mathbf{N}_i$, a column $\mathbf{t}_i$ of the unknown matrix $\mathbf{T}$ may be calculated from Eq.(B.8) as:

$$\mathbf{t}_i = -\mathbf{N}_i^{-1}\,y_{ii} \tag{B.9}$$

Having $\mathbf{t}_i$ from Eq.(B.9), the scalars d_j, $j=[0,n\text{-}1]$, may be readily obtained from Eq.(B.6).

B.2 Hessenberg Transformations

The basic idea of the Hessenberg transformation is to apply a sequence of n-p similarity transformations by which a given unobservable pair $\{\mathbf{A},\mathbf{C}\}$ with a full rank $\mathbf{C}$ is transformed into the form:

$$\mathbf{A} = \left[\begin{array}{c|c} \mathbf{A}_o & \mathbf{0} \\ \hline \mathbf{A}_{21} & \mathbf{A}_{no} \end{array}\right] \begin{array}{l} \} \, n_o \\ \\ \end{array}, \quad \mathbf{B} = \left[\begin{array}{c} \mathbf{B}_o \\ \hline \mathbf{B}_{no} \end{array}\right] \tag{B.10}$$
$$\mathbf{C} = [\, \mathbf{C}_o \mid \mathbf{0} \,]$$

In other words, after this sequence of similarity transformations, the matrix $\mathbf{C}$ is "column" reduced, i.e. only the first p columns have elements different from zero. The matrix $\mathbf{A}$ has an $(n_o \times n\text{-}n_o)$ block of zeros in the "upper right" corner, while $\mathbf{B}$ has no special structure. Obviously the modes, i.e. eigenvalues of the block $\mathbf{A}_{no}$ are not observable since, due to the structure of $\{\mathbf{A},\mathbf{C}\}$, they do not contribute to the output $\mathbf{y}(t) = \mathbf{C}\,\mathbf{x}(t)$.

Thus, the observable part of a given $\{\mathbf{A},\mathbf{B},\mathbf{C}\}$ becomes an n_o-th order representation given by:

$$R_o = \{\mathbf{A}_o, \mathbf{B}_o, \mathbf{C}_o\}$$

To check the controllability of $\{\mathbf{A}_o, \mathbf{B}_o, \mathbf{C}_o\}$, i.e. to eliminate possible uncontrollable modes, it is recommended that one consider the dual system

$$R_D = \{\mathbf{A}_D, \mathbf{B}_D, \mathbf{C}_D\} = \{\mathbf{A}_o^T, \mathbf{C}_o^T, \mathbf{B}_o^T\}$$

of the obtained R_o, and eliminate its unobservable modes, which are, of course, the uncotrollable modes of R_o, to obtain:

$$R_{Do} = \{\mathbf{A}_{Do}, \mathbf{B}_{Do}, \mathbf{C}_{Do}\}$$

Finally, a desired minimal realization corresponding to the given $\{\mathbf{A},\mathbf{B},\mathbf{C}\}$ is equal to the dual of R_{Do}, i.e.

$$R_m = \{\mathbf{A}_{Do}^T, \mathbf{C}_{Do}^T, \mathbf{B}_{Do}^T\}$$

The algorithm *MIN* given below performs only the elimination of the unobservable part, i.e. the transformation:

$$R \rightarrow R_o$$

Algorithm *MIN*:

```
1.  Set $\mathbf{C} \Rightarrow \mathbf{R}$, $n \Rightarrow k$
2.  Set $\mathbf{R}$ (SVD) $\Rightarrow \mathbf{s},\mathbf{U},\mathbf{V}$ ; where $\mathbf{R} = \mathbf{U}\mathbf{S}\mathbf{V}^T$ , $\mathbf{S} = \mathrm{diag}\{ s_1,\ldots, s_k \}$
3.  Set $\mathbf{V} \Rightarrow \mathbf{T}_h$, $k - p \Rightarrow k$, $1 \Rightarrow i$, $p+1 \Rightarrow j$
4.  Set $\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{V}$(STR) $\Rightarrow \mathbf{A}',\mathbf{B}',\mathbf{C}'$
5.  Partition $\mathbf{A}' \Rightarrow \left[\begin{array}{c|c} & \mathbf{Y} \\ \mathbf{X} & \mathbf{R} \\ & \mathbf{Z} \end{array}\right] \begin{array}{l} \}\ i-1 \\ \}\ p \\ \\ \end{array}$ , $\mathbf{R}$ is $(p \times k)$ such that $r_{11} = a'_{ij}$
6.  Set $\mathbf{R}$ (SVD) $\Rightarrow \mathbf{s},\mathbf{U},\mathbf{V}$
7.  Set $\mathrm{diag}\{ \mathbf{I}_{j-1}, \mathbf{V} \} \Rightarrow \mathbf{T}$, $\mathbf{T}_h\mathbf{T} \Rightarrow \mathbf{T}_h$
8.  Set $\mathbf{A}',\mathbf{B}',\mathbf{C}',\mathbf{T}$(STR) $\Rightarrow \mathbf{A}',\mathbf{B}',\mathbf{C}'$
9.  Set $k-1 \Rightarrow k$, $j+1 \Rightarrow j$, $i+1 \Rightarrow i$
10. If $j \le n$, go to 5; else, go to 11
11. Set $\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{T}_h$(STR) $\Rightarrow \mathbf{A}_h,\mathbf{B}_h,\mathbf{C}_h$
12. Set $n \Rightarrow n_o$, $0 \Rightarrow i$
13. Set $i+1 \Rightarrow i$, $n-i \Rightarrow k$
14. Partition $\mathbf{A}_h \Rightarrow \left[\begin{array}{c|c} \mathbf{X} & \mathbf{Z} \\ \hline \mathbf{Y} & \mathbf{W} \end{array}\right]$; $\mathbf{Z}$ is the $(k \times i)$ upper right block of $\mathbf{A}_h$
15. If $\| \mathbf{Z} \| > eps$, go to 17; else, go to 16
16. Set $k \Rightarrow n_o$
17. If $i < n-p$, go to 13; else, go to 18
18. Paritition $\mathbf{A}_h \Rightarrow \left[\begin{array}{c|c} \mathbf{A}_o & \mathbf{0} \\ \hline \mathbf{Y} & \mathbf{Z} \end{array}\right]$; $\mathbf{B}_h \Rightarrow \left[\begin{array}{c} \mathbf{B}_o \\ \hline \mathbf{X} \end{array}\right]$; $\mathbf{C}_h \Rightarrow [\ \mathbf{C}_o \mid \mathbf{0}\ ]$

        $R_o = \{\mathbf{A}_o,\mathbf{B}_o,\mathbf{C}_o\}$ is an $n_o$th order observable representation.
```

Note that $R' = \{\mathbf{A}',\mathbf{B}',\mathbf{C}'\}$, obtained in the last passage through Step 8, is the same as $R_h = \{\mathbf{A}_h,\mathbf{B}_h,\mathbf{C}_h\}$, obtained in Step 11, since the transformation matrix $\mathbf{T}_h$ accumulates all transformations performed in the loop, Steps 5 to 10.

Singular Value Deconoposition: For a given $(p \times k)$ matrix $\mathbf{R}$, Algorithm *SVD*, singular value decomposition, in Steps 2 and 6, calculates arrays $\mathbf{s}$, $\mathbf{U}$ and $\mathbf{V}$ where:

- $\mathbf{s}$ is a k-dimensional row array containing the singular values s_i, $i=[1,k]$, $s_i \ge 0$.
- $\mathbf{U}$ is a $(p \times k)$ matrix containing the "left" singular vectors.
- $\mathbf{V}$ is a $(k \times k)$ matrix containing the "right" singular vectors.

The arrays s, U and V satisfy:

$$\mathbf{R} = \mathbf{USV}^T , \text{ where } \mathbf{S} = \mathrm{diag}\{ s_1, \ldots, s_k \}$$

and $s_1 \geq s_2 \geq \ldots \geq s_k \geq 0$. Matrices **U** and **V** are "unitary" i.e. $\mathbf{V}^T\mathbf{V} = \mathbf{I}_k$ and $\mathbf{UU}^T = \mathbf{I}_p$, if $p \leq k$, or in other words, the columns of **U** and **V** are orthonormal.

The main property of the Hessenberg similarity transformation matrix $\mathbf{T}_k$ is that in the transformed triple $R_k = \{\mathbf{A}_h,\mathbf{B}_h,\mathbf{C}_h\}$ the $(n_o \times n\text{-}n_o)$ upper right submatrix of $\mathbf{A}_h$ and the last $n\text{-}n_o$ columns in $\mathbf{C}_k$ contain zero elements. This is a consequence of the fact that the last $k\text{-}r$ columns of **V**, where $r = \mathrm{rank}(\mathbf{R})$, are in the null space of **R**, i.e. in the product $\mathbf{X} = \mathbf{RV}$ all non-zero elements are "concentrated" in the first r columns of **X**. Thus, it is clear that the last $n\text{-}n_o$ modes of R_k are unobservable, and in order to obtain an observable part, these modes should be eliminated from R_k. Similarly, to eliminate possible uncontrollable modes, the same Hessenberg transformation should be applied to the representation which is dual to the above obtained R_o.

For more details on the Hessenberg trasnformation readers are referred to the reference section in Chapter 4.

B.3 The Kalman Decomposition

Consider a not necessarily minimal state space representation {**A**,**B**,**C**}, with order n, $m \leq n$ inputs and $p \leq n$ outputs. The Kalman Canonical decomposition is defined as the procedure of decomposing a given state space representation {**A**,**B**,**C**} into the following four coupled subsystems referred to as the:

1. Controllable and unobservable subsystem, denoted by $c\bar{o}$,
2. Controllable and observable, denoted by co,
3. Uncontrollable and unobservable, denoted by $\bar{c}\bar{o}$, and
4. Uncontrollable and observable, denoted by $\bar{c}o$.

The problem is to find a similarity transformation matrix **T** which will transform the given {**A**,**B**,**C**} into a form $\{\mathbf{A}_d,\mathbf{B}_d,\mathbf{C}_d\}$ where:

$$\begin{aligned} \mathbf{A}_d &= \mathbf{T}^{-1}\mathbf{AT} \\ \mathbf{B}_d &= \mathbf{T}^{-1}\mathbf{B} \\ \mathbf{C}_d &= \mathbf{CT} \end{aligned} \tag{B.11}$$

The structure of the matrices in Eq.(B.11) is as follows:

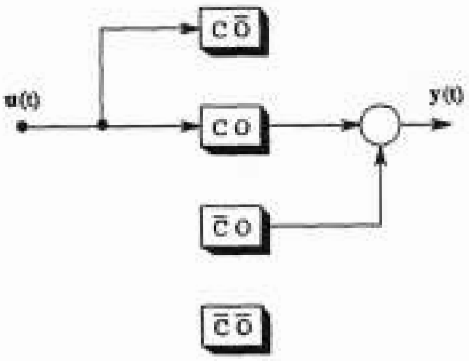


FIGURE B.1 Kalman Canonical Decomposition

$$\mathbf{A}_d = \begin{bmatrix} \mathbf{A}_{c\bar{o}} & \mathbf{A}_{12} & \mathbf{A}_{13} & \mathbf{A}_{14} \\ 0 & \mathbf{A}_{co} & 0 & \mathbf{A}_{24} \\ 0 & 0 & \mathbf{A}_{\bar{c}\bar{o}} & \mathbf{A}_{34} \\ 0 & 0 & 0 & \mathbf{A}_{\bar{c}o} \end{bmatrix}, \quad \mathbf{B}_d = \begin{bmatrix} \mathbf{B}_{\bar{o}} \\ \mathbf{B}_o \\ 0 \\ 0 \end{bmatrix} \tag{B.12}$$
$$\mathbf{C}_d = \begin{bmatrix} 0 & \mathbf{C}_c & 0 & \mathbf{C}_{\bar{c}} \end{bmatrix}$$

The structures given by Eq.(B.12) can be represented by the block diagram given in Fig. B.1. In Fig.B.1, as well as in Eq.(B.12), c and $\bar{c}$ stand for controllable and uncontrollable subsystems, while o and $\bar{o}$ stand for observable and unobservable subsystems, respectively. However, Fig. B.1 does not show all internal connections between the four subsystems. A more detailed block diagram is given in Fig. B.2.

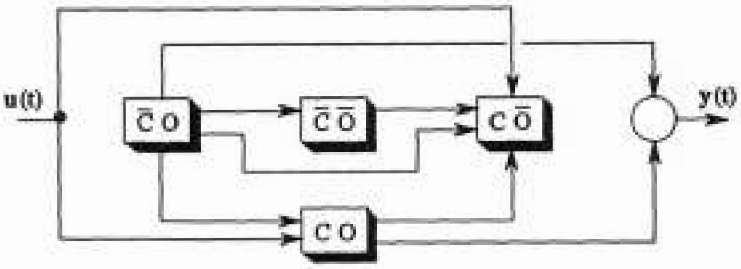


FIGURE B.2 Detailed Kalman Decomposition

Normally, only procedures for the determination of the *co* subsystem are considered; however, our purpose here is to suggest a possible algorithm for performing a Kalman canonical decomposition, i.e. a constructive procedure for calculating a similarity transformation matrix $\mathbf{T}$ performing the decomposition in Eq.(B.12).

Decomposition Procedure

The transformation matrix $\mathbf{T}$ should be given by the concatenation of the following submatrices:

$$\mathbf{T} = \left[\, \mathbf{T}_{c\bar{o}} \quad \mathbf{T}_{co} \quad \mathbf{T}_{\bar{c}\bar{o}} \quad \mathbf{T}_{\bar{c}o} \,\right] \tag{B.13}$$

where:

$$\mathbf{T}_{c} = \left[\, \mathbf{T}_{co} \quad \mathbf{T}_{c\bar{o}} \,\right] \tag{B.14}$$

should span the controllable subspace of the pair $\{\mathbf{A},\mathbf{B}\}$, while:

$$\mathbf{T}_{\bar{o}} = \left[\, \mathbf{T}_{c\bar{o}} \quad \mathbf{T}_{\bar{c}\bar{o}} \,\right] \tag{B.15}$$

should span the unobservable subspace of the pair $\{\mathbf{A},\mathbf{C}\}$.

Thus, the matrix $\mathbf{T}_c$ could be obtained by calculating the range space of the controllability matrix of the pair $\{\mathbf{A},\mathbf{B}\}$:

$$\mathbf{Q}_c = \left[\, \mathbf{B} \quad \mathbf{A}\mathbf{B} \quad \mathbf{A}^2\mathbf{B} \quad \cdots \quad \mathbf{A}^{n-m}\mathbf{B} \,\right] \tag{B.16}$$

i.e.

$$\mathbf{T}_c = \mathbf{R}(\mathbf{Q}_c) \tag{B.17}$$

Similarly, the matrix $\mathbf{T}_{\bar{o}}$ could be calculated from the observability matrix of the pair $\{\mathbf{A},\mathbf{C}\}$:

$$\mathbf{Q}_o = \begin{bmatrix} \mathbf{C} \\ \mathbf{C}\mathbf{A} \\ \mathbf{C}\mathbf{A}^2 \\ \vdots \\ \mathbf{C}\mathbf{A}^{n-p} \end{bmatrix} \tag{B.18}$$

using

$$\mathbf{T}_{\bar{o}} = \mathbf{N}(\mathbf{R}_{Qo}^T) \tag{B.19}$$

where $\mathbf{R}_{Qo}$ is the range space of $\mathbf{Q}_o$, i.e.

$$\mathbf{R}_{Qo} = \mathbf{R}(\mathbf{Q}_o) \tag{B.20}$$

The matrices $\mathbf{Q}_c$ and $\mathbf{Q}_o$ could be calculated by Algorithms *Qc* and *Qo* given in

Chapter 1 and Appendix C. In Eqs.(B.17), (B.19) and (B.20) the symbols $\mathbf{R}(\mathbf{X})$ and $\mathbf{N}(\mathbf{X})$ denote the range and null spaces of $\mathbf{X}$, respectively, while $\mathbf{X}^T$ denotes matrix transposition. The range and null spaces of a given matrix can be calculated by Algorithm *NRS* given in Appendix C.

Having matrices $\mathbf{T}_c$ and $\mathbf{T}_{\bar{o}}$, defined by Eqs.(B.14) and (B.15), the sub-matrices of $\mathbf{T}_c$ entering into the similarity transformation matrix $\mathbf{T}$, Eq.(B.13), can be obtained by decomposing $\mathbf{T}_c$ into a part spanned by the columns in $\mathbf{T}_{\bar{o}}$ and a part which is not. Similarly, the sub-matrices in Eq.(B.15) can be obtained by decomposing $\mathbf{T}_{\bar{o}}$ into a part spanned by the columns in $\mathbf{T}_c$ and a part which is not.

The above decompositions into the desired sub-matrices can be obtained by Algorithm *InOu* given in Appendix C, i.e.:

$$\mathbf{T}_{\bar{o}}\ ,\ \mathbf{T}_c\ ,\ \epsilon\ (\mathit{InOu}) \rightarrow \mathbf{T}_{c\bar{o}}\ ,\ \mathbf{T}_{co}$$

$$\mathbf{T}_c\ ,\ \mathbf{T}_{\bar{o}}\ ,\ \epsilon\ (\mathit{InOu}) \rightarrow \hat{\mathbf{T}}_{c\bar{o}}\ ,\ \mathbf{T}_{\bar{c}\bar{o}}$$

(Note: $\mathbf{T}_{c\bar{o}}$ and $\hat{\mathbf{T}}_{c\bar{o}}$ are not unique, but both versions span the same sub-space.)

Finally, the sub-matrix $\mathbf{T}_{\bar{c}o}$, which has not yet been determined, can be obtained as the null space of $\mathbf{T}_1{}^T$, i.e.

$$\mathbf{T}_{\bar{c}o} = \mathbf{N}(\mathbf{T}_1^T) \tag{B.21}$$

where

$$\mathbf{T}_1 = [\ \mathbf{T}_{c\bar{o}}\quad \mathbf{T}_{co}\quad \mathbf{T}_{\bar{c}\bar{o}}\ ] \tag{B.22}$$

Using the above mentioned algorithms, we are now ready to formulate the algorithm for Kalman canonical decomposition.

Kalman Canonical Decomposition Algorithm

```
1.  Define a state space representation $\{\mathbf{A},\mathbf{B},\mathbf{C}\}$ and the scalar eps satisfying:
    $0 < eps << 1$.
2.  Set $\mathbf{A}$, $\mathbf{B}$(Qc) $\Rightarrow \mathbf{Q}_c$, Eq.(B.16)
3.  Set $\mathbf{A}$, $\mathbf{C}$(Qo) $\Rightarrow \mathbf{Q}_o$, Eq.(B.18)
4.  Set $\mathbf{Q}_c$, eps (NRS) $\Rightarrow \mathbf{X}$, $\mathbf{T}_c$, $r_c$;  $\mathbf{T}_c$ is the controllability subspace
5.  Set $\mathbf{Q}_o{}^T$, eps (NRS) $\Rightarrow \mathbf{X}$, Rqot, $x$; Eq.(B.20)
6.  Set Rqot$^T$, eps (NRS) $\Rightarrow \mathbf{T}_{\bar{o}}$, $\mathbf{X}$, $r_o$; $\mathbf{T}_{\bar{o}}$ is the unobservability subspace
7.  Set $\mathbf{T}_{\bar{o}}$, $\mathbf{T}_c$, eps (InOu) $\Rightarrow \mathbf{T}_{c\bar{o}}$, $\mathbf{T}_{co}$; Eq.(B.14)
8.  Set $\mathbf{T}_c$, $\mathbf{T}_{\bar{o}}$, eps (InOu) $\Rightarrow \mathbf{T}_{c\bar{o}}$, $\mathbf{T}_{\bar{c}\bar{o}}$; Eq.(B.15)
9.  Set $[\ \mathbf{T}_{c\bar{o}}\quad \mathbf{T}_{co}\quad \mathbf{T}_{\bar{c}\bar{o}}\ ] \Rightarrow \mathbf{T}_1$; Eq.(B.24)
10. Set $\mathbf{T}_1{}^T$, eps (NRS) $\Rightarrow \mathbf{T}_{\bar{c}o}$, $\mathbf{X}$, $x$
```

```
11. Set $[\ \mathbf{T}_1 \quad \mathbf{T}_{\bar{c}o}\ ] \Rightarrow \mathbf{T}$, Eq.(B.13)
12. Set $[\ \dim(\mathbf{T}_{c\bar{o}}) \quad \dim(\mathbf{T}_{co}) \quad \dim(\mathbf{T}_{\bar{c}\bar{o}}) \quad \dim(\mathbf{T}_{\bar{c}o})\ ] \Rightarrow \mathbf{d}$
13. Set $\mathbf{A}, \mathbf{B}, \mathbf{C}, T$ $(STR) \Rightarrow \mathbf{A}_d, \mathbf{B}_d, \mathbf{C}_d$; Eq.(B.11)
14. Stop
```

Algorithm Implementation

The listing of Algorithm *KALD* implemented using the *L-A-S* language is given in Appendix C. Algorithms *Qc*, *Qo* and *NRS* are implemented using corresponding *L-A-S* operators, while *InOu* is implemented by the subroutine InOu. The matrix denoted by **Rqot**, determined in Step 5 and used in Step 6, is the range space of $\mathbf{Q}_o^T$. Algorithm *STR*, used in Step 13 is performed using the *L-A-S* operator STR. The four dimensional row **d** calculated in Step 12, containing subspace dimensions, is calculated using the *L-A-S* operators CDI and CTI.

B.4 Computation of Generalized Eigenvectors

In this section a method is suggested for computing the eigenvectors and generalized eigenvectors of a matrix, given that the eigenvalues are already known. The algorithm is straightforward, and an *L-A-S* implementation is provided. The following algorithms are associated with the general problem. Each algorithm is described in detail.

Algorithm: *MODM*

Syntax: **A**, **Egv**, *eps* $(MODM) \Rightarrow \mathbf{P}$

Purpose: Calculation of the eigenvector (modal) matrix of a matrix with multiple real or complex eigenvalues.

Input/Output arguments:

- $\mathbf{A} = (n \times n)$ real matrix with multiple real or complex-conjugate eigenvalues.
- $\mathbf{Egv} = (m \times 2)$ matrix, $m \leq n$, containing distinct eigenvalues, $\lambda_j = \sigma_j + j\omega_j$, $j=[1,m]$, of **A**; first column contains real parts, second column contains imaginary parts. In the case of a complex-conjugate pair of eigenvalues, **Egv** contains only one eigenvalue of the pair.
- *eps*, a sufficiently small positive scalar; suggested value: $eps = 10^{-5}$.

- $\mathbf{P}$ = $(n \times n)$ eigenvector matrix transforming $\mathbf{A}$ into the "real number" Jordan form.

Description:

The $(n \times n)$ modal matrix:

$$\mathbf{P} = [\, \mathbf{P}_1 \quad \cdots \quad \mathbf{P}_j \quad \cdots \quad \mathbf{P}_m \,] \tag{B.23}$$

is related to a given $\mathbf{A}$ by:

$$\mathbf{A}\mathbf{P} = \mathbf{P}\mathbf{A}_J \qquad \text{or} \qquad \mathbf{A}_J = \mathbf{P}^{-1}\mathbf{A}\mathbf{P} \tag{B.24}$$

where $\mathbf{A}_J$ is an $(n \times n)$ block diagonal Jordan form matrix given by:

$$\mathbf{A}_J = \mathrm{diag}\{\, \mathbf{A}_{J1}, \cdots, \mathbf{A}_{Jj}, \cdots, \mathbf{A}_{Jm} \,\} \tag{B.25}$$

The $(n \times n_j)$ and $(n_j \times n_j)$, $j=[1,m]$, matrices $\mathbf{P}_j$ and $\mathbf{A}_{Jj}$ in Eqs.(B.23) and (B.25) satisfy:

$$\mathbf{A}\mathbf{P}_j = \mathbf{P}_j\mathbf{A}_{Jj} \quad \text{where} \quad \sum_{j=1}^{m} n_j = n \tag{B.26}$$

Integers n_j are the *algebraic multiplicities* of the eigenvalues λ_j, $j=[1,m]$, of the matrix $\mathbf{A}$.

Matrices $\mathbf{P}_j$ and $\mathbf{A}_j$ could be partitioned as:

$$\begin{aligned} \mathbf{P}_j &= [\, \mathbf{P}_{j1} \quad \cdots \quad \mathbf{P}_{jk} \quad \cdots \quad \mathbf{P}_{j\nu_j} \,] \\ \mathbf{A}_{Jj} &= \mathrm{diag}\{\, \mathbf{A}_{Jj1}, \cdots, \mathbf{A}_{Jjk}, \cdots, \mathbf{A}_{Jj\nu_j} \,\} \end{aligned} \tag{B.27}$$

Integer ν_j, referred to as the *geometric multiplicity* of (multiple) eigenvalue λ_j, is given by the nullity of the matrix $\mathbf{B}_j = \mathbf{A} - \lambda_j\mathbf{I}$, i.e. ν_j is the dimension of the null space of $\mathbf{B}_j$.

The $(n \times n_{jk})$ matrices $\mathbf{P}_{jk}$, $k=[1,\nu_j]$, contain n_{jk} eigenvectors belonging to the k^{th} eigenvector chain of length n_{jk} associated with the eigenvalue λ_j,

$$\mathbf{P}_{jk} = [\, \mathbf{p}_{jk1} \quad \cdots \quad \mathbf{p}_{jkr} \quad \cdots \quad \mathbf{p}_{jkn} \,] \quad \text{where} \quad n_j = \sum_{k=1}^{\nu_j} n_{jk} \tag{B.28}$$

The $(n_{jk} \times n_{jk})$ matrix $\mathbf{A}_{Jjk}$, referred to as the k^{th} Jordan block associated with λ_j, satisfies:

$$\mathbf{A}\mathbf{P}_{jk} = \mathbf{P}_{jk}\mathbf{A}_{Jjk} \tag{B.29}$$

In the case of a real eigenvalue $\lambda_j = \sigma_j$, the Jordan block $\mathbf{A}_{Jjk}$ is an upper triangular matrix consisting of σ_j on the diagonal and a super diagonal of unities of the form

$$\mathbf{A}_{Jjk} = \begin{bmatrix} \sigma_j & 1 & & & & \\ & \sigma_j & 1 & & & \\ & & \sigma_j & \ddots & & \\ & & & \ddots & 1 & \\ & & & & \sigma_j & 1 \\ & & & & & \sigma_j \end{bmatrix} \tag{B.30}$$

In the case of a complex-conjugate pair of eigenvalues $\lambda_j = \sigma_j \pm j\omega_j$, the $(2n_{jk} \times 2n_{jk})$ Jordan block $\mathbf{A}_{Jjk}$ is associated with both eigenvalues of the pair and is in "real number" form given by:

$$\mathbf{A}_{Jjk} = \begin{bmatrix} \sigma_j & \omega_j & 1 & & & & & & & \\ -\omega_j & \sigma_j & & 1 & & & & & & \\ & & \sigma_j & \omega_j & 1 & & & & & \\ & & -\omega_j & \sigma_j & & 1 & & & & \\ & & & \ddots & \ddots & \ddots & \ddots & & & \\ & & & & & & \sigma_j & \omega_j & 1 & \\ & & & & & & -\omega_j & \sigma_j & & 1 \\ & & & & & & & & \sigma_j & \omega_j \\ & & & & & & & & -\omega_j & \sigma_j \end{bmatrix} \tag{B.31}$$

In order to satisfy Eq.(B.29) with $\mathbf{A}_{Jjk}$ in the real number form given by Eq.(B.31), the $(n \times 2n_{jk})$ eigenvector matrix $\mathbf{P}_{jk}$ is associated with both eigenvalues in the complex-conjugate pair $\sigma_j \pm j\omega_j$. The first two columns of $\mathbf{P}_{jk}$ are given by the real and imaginary parts of the eigenvector corresponding to the eigenvalue $\sigma_j + j\omega_j$. Similarly, the next two columns are the real and imaginary parts of the first generalized eigenvector corresponding to $\sigma_j + j\omega_j$.

The eigenvector chains $\mathbf{P}_{jk}$ in the $(n \times n_j)$ matrix $\mathbf{P}_j$ corresponding to a real eigenvalue $\lambda_j = \sigma_j$ with a multiplicity n_j are calculated by Algorithm *CHAR*. The eigenvector chains $\mathbf{P}_{jk}$ in the $(n \times 2n_j)$ matrix $\mathbf{P}_j$ corresponding to both eigenvalues of the complex conjugate pair $\sigma_j \pm j\omega_j$ with multiplicity n_j are calculated by Algorithm *CHAC*.

Algorithm:

```
1.  Define $(n \times n)$ matrix $\mathbf{A}$, distinct eigenvalues $\mathbf{Egv}$ and the scalar $eps$.
2.  Set number of columns in $\mathbf{A} \Rightarrow n$
3.  Set number of rows in $\mathbf{Egv} \Rightarrow m$
4.  Set $\mathbf{0}_{n,0} \Rightarrow \mathbf{P}$
5.  Set $0 \Rightarrow j$
6.  Set $j+1 \Rightarrow j$
7.  Extract $j^{th}$ row of $\mathbf{Egv} \Rightarrow \lambda\mathbf{j}$
8.  Partition $\lambda\mathbf{j} \Rightarrow [\ \sigma_j \mid \omega_j\ ]$
9.  If $\omega_j = 0$, go to 12; else, go to 10
10. Set $\mathbf{A}$, $\sigma_j$, $\omega_j$, $eps$ $(CHAC) \Rightarrow \mathbf{P}_j$
11. Go to 13
12. Set $\mathbf{A}$, $\sigma_j$, $eps$ $(CHAR) \Rightarrow \mathbf{P}_j$
13. Set $[\ \mathbf{P} \mid \mathbf{P}_j\ ] \Rightarrow \mathbf{P}$
14. If $j < m$, go to 6; else, stop.
```

Algorithm Implementation

The listing of Algorithm *MODM* implemented using the *L-A-S* language is given in Appendix C. Algorithms *CHAR* and *CHAC* are implemented using the *L-A-S* subroutines CHAR and CHAC, respectively. These two algorithms are presented next.

Algorithm: CHAR

Syntax: $\mathbf{A}$, σ_j , eps $(CHAR) \Rightarrow \mathbf{P}_j$

Purpose:

The calculation of the eigenvector chains corresponding to a real eigenvalue σ_j with multiplicity n_j, $n_j \geq 1$.

Input/Output Arguments:

- $\mathbf{A}$ = $(n \times n)$ matrix.
- σ_j = scalar corresponding to a real eigenvalue of $\mathbf{A}$ with

multiplicity n_j, $n_j \geq 1$.

- *eps* = sufficiently small positive scalar; suggested value: $eps = 10^{-5}$.
- $\mathbf{P}_j$ = $(n \times n_j)$ matrix containing all eigenvector chains corresponding to σ_j.

Description:

Let σ_j be a real eigenvalue of $\mathbf{A}$ with multiplicity n_j. Then v_j, the *nullity* of $\mathbf{B}_j = \mathbf{A} - \sigma_j\mathbf{I}$, i.e. the dimension of the null space of the matrix $\mathbf{B}_j$, satisfies:

$$1 \leq v_j \leq n_j \tag{B.32}$$

In the matrix $\mathbf{P}_j$ there are v_j proper and $(n_j - v_j)$ generalized eigenvectors corresponding to σ_j. According to Eqs.(B.27) and (B.28), these vectors are arranged in $\mathbf{P}_j$ as v_j eigenvector chains $\mathbf{P}_{jk}$ of lengths n_{jk}. Without loss of generality it may be assumed that:

$$n_{j1} \leq n_{j2} \leq \cdots \leq n_{jv_j} \tag{B.33}$$

The vectors $\mathbf{p}_{jkr}$, $r=[1,v_j]$, in the chains $\mathbf{P}_{jk}$ satisfy:

$$\mathbf{B}_j\mathbf{p}_{jk1} = \mathbf{0} \;\; \text{and} \;\; \mathbf{B}_j\mathbf{p}_{jkr} = \mathbf{p}_{jk(r-1)} \quad \text{for } r=[2,v_j] \tag{B.34}$$

or

$$(\mathbf{B}_j)^r\mathbf{p}_{jkr} = \mathbf{0} \quad \text{for } r=[1,v_j] \tag{B.35}$$

From Eq.(B.34) it follows that the vectors $\mathbf{p}_{jkr}$ for $r=[1,v_j\text{-}1]$ are in the range space of $\mathbf{B}_j$, while the vectors $\mathbf{p}_{jkn_{jk}}$, which are last in the chains are outside the range space of $\mathbf{B}_j$, i.e.:

$$\begin{gathered}\operatorname{rank}\left[\mathbf{B}_j \mid \mathbf{p}_{jkr}\right] = \operatorname{rank}\left[\mathbf{B}_j\right] \;\; \text{for} \;\; k = [1,\, n_{jk}-1] \;\; \text{and} \\ \operatorname{rank}\left[\mathbf{B}_j \mid \mathbf{p}_{jkn_{jk}}\right] = \operatorname{rank}[\mathbf{B}_j] + 1\end{gathered} \tag{B.36}$$

From Eq.(B.35) it follows that all of the vectors $\mathbf{p}_{jkr}$ are in null space of $(\mathbf{B}_j)^k$.

In Algorithm *CHAR* first the matrix $\mathbf{B}_j$ is built and its nullity v_j and range space $\mathbf{R}$ are calculated. Then the vector $\mathbf{p}_{j1n_{j1}}$, i.e. the last vector in the shortest eigenvector chain, is determined. This is done by determining the smallest integer k such that the null space $\mathbf{N}_k$ of the matrix $(\mathbf{B}_j)^k$ contains vectors $\mathbf{m}_i$, $i=[1,q]$, which are outside $\mathbf{R}$. If $q > 1$, i.e. if there is more than one vector satisfying this condition, then there are q chains with length $k = n_{j1}$. These vectors are then used as the last vector in their respective chains, and the other vectors in those chains are calculated using Eq.(B.34) by a simple premultiplication of $\mathbf{m}_i$ with $\mathbf{B}_j$.

These eigenvector chains are of the form:

$$\left[\, (\mathbf{B}_j)^{n_{j1}-1}\mathbf{m}_i \;\mid\; \cdots \;\mid\; \mathbf{B}_j\mathbf{m}_i \;\mid\; \mathbf{m}_i \,\right],\; i=[1,q] \qquad \text{(B.37)}$$

If $q < \nu_j$, more chains, longer than n_{j1}, are needed. Again, the last vectors in these chains are obtained by detecting the next smallest integer k such that the null space of the matrix $(\mathbf{B}_j)^k$ contains vectors outside the range space of the matrix obtained by concatenating $\mathbf{R}$ and the matrix $\mathbf{M}$ consisting of the q vectors $\mathbf{m}_i$, $i=[1,q]$, used in Eq.(B.37) for building the q eigenvector chains of length n_{j1}.

Calculation of the range and null spaces is done using Algorithm *NRS*. Calculation of the vectors which are outside the range space of a given matrix is done using Algorithm *INOU*.

Algorithm:

```
1. Define $(n \times n)$ matrix $\mathbf{A}$, real eigenvalue $\sigma_j$ and the scalar $eps$.
2. Set number of columns in $\mathbf{A} \Rightarrow n$
3. Set $\mathbf{I}_{n,n} \Rightarrow \mathbf{I}$
4. Set $\mathbf{0}_{n,0} \Rightarrow \mathbf{P}_j$
5. Set $\mathbf{A} - \sigma_j\mathbf{I} \Rightarrow \mathbf{B}_j$
6. Set $\mathbf{B}_j$, $eps$ $(NRS) \Rightarrow \mathbf{N}, \mathbf{R}, x$
7. Set number of columns in $\mathbf{N} \Rightarrow \nu_j$
8. Set $\mathbf{I} \Rightarrow \mathbf{B}_k$, $0 \Rightarrow k$, $0 \Rightarrow r$
9. Set $k+1 \Rightarrow k$
10. Set $\mathbf{B}_k\,\mathbf{B}_j \Rightarrow \mathbf{B}_k$
11. Set $\mathbf{B}_k$, $eps$ $(NRS) \Rightarrow \mathbf{N}_k, \mathbf{Y}, x$
12. Set $\mathbf{R}, \mathbf{N}_k$, $eps$ $(INOU) \Rightarrow \mathbf{Y}, \mathbf{M}$
13. Set number of columns in $\mathbf{M} \Rightarrow q$
14. Set $\mathbf{M} \Rightarrow \mathbf{M}_t$
15. If $q > 0$, go to 16; else, go to 9
16. Set $\mathbf{M}_t \Rightarrow [\, \mathbf{m} \mid \mathbf{M}_t \,]$
17. Set $\mathbf{0}_{n,0} \Rightarrow \mathbf{P}_t$, Set $0 \Rightarrow i$
18. Set $i+1 \Rightarrow i$
19. Set $[\, \mathbf{m} \mid \mathbf{P}_t \,] \Rightarrow \mathbf{P}_t$
20. Set $\mathbf{B}_j\mathbf{m} \Rightarrow \mathbf{m}$
21. If $i < k$, go to 18; else, go to 22
22. Set $[\, \mathbf{P}_j \mid \mathbf{P}_t \,] \Rightarrow \mathbf{P}_j$
23. If number of columns in $\mathbf{M}_t > 0$, go to 16; else, go to 24
24. Set $[\, \mathbf{R} \mid \mathbf{M} \,] \Rightarrow \mathbf{R}$
25. Set $r+q \Rightarrow r$
26. If $r < \nu_j$, go to 9; else, stop
```

Algorithm Implementation:

The listing of Algorithm *CHAR*, implemented using the *L-A-S* language is given in Appendix C. Algorithms *NRS* and *INOU* are performed using the *L-A-S* operator NRS and subroutine INOU.SUB. The matrix partitioning in Step 16 is done by the *L-A-S* operator CTC. Matrix concatenation in Steps 19 and 22 is uses the operator CTI.

Algorithm: CHAC

Syntax: $\mathbf{A}$, σ_j, ω_j, *eps* (*CHAC*) $\Rightarrow \mathbf{P}_j$

Purpose:

Calculation of the eigenvector chains corresponding to a pair of complex-conjugate eigenvalues $\sigma_j \pm j\omega_j$, with multiplicity n_j, $n_j \geq 1$.

Input/Output Arguments:

- $\mathbf{A} = (n \times n)$ matrix.
- σ_j and ω_j = scalars representing real and imaginary parts of a complex-conjugate pair of eigenvalues, $\sigma_j \pm j\omega_j$, of $\mathbf{A}$ with multiplicity n_j, $n_j \geq 1$.
- *eps* = sufficiently small positive scalar; suggested value: $eps = 10^{-5}$.
- $\mathbf{P}_j = (n \times 2n_j)$ matrix containing all of the eigenvector chains corresponding to the pair of eigenvalues $\sigma_j \pm j\omega_j$.

Description:

Let a matrix $\mathbf{A}$ have a complex-conjugate pair of eigenvalues $\lambda_j = \sigma_j + j\omega_j$ and $\lambda_{j+1} = \sigma_j - j\omega_j$ with multiplicity n_j. Then ν_j, given by the nullity of either $\mathbf{B}_j = \mathbf{A} - \lambda_j\mathbf{I}$ or $\mathbf{B}_{j+1} = \mathbf{A} - \lambda_{j+1}\mathbf{I}$, satisfies:

$$1 \leq \nu_j \leq n_j \tag{B.38}$$

Proper complex-conjugate eigenvectors $\mathbf{p}_j = \mathbf{u}_j + j\mathbf{v}_j$ and $\mathbf{p}_{j+1} = \mathbf{u}_j - j\mathbf{v}_j$ associated with eigenvalues λ_j and λ_{j+1}, respectively, satisfy:

$$\begin{bmatrix} \mathbf{B}_j & \mathbf{0} \\ \mathbf{0} & \mathbf{B}_{j+1} \end{bmatrix} \begin{bmatrix} \mathbf{p}_j \\ \mathbf{p}_{j+1} \end{bmatrix} = \mathbf{0} \tag{B.39}$$

It may be verified by inspection that real vectors $\mathbf{u}_j$ and $\mathbf{v}_j$ defining the complex-conjugate eigenvectors $\mathbf{p}_j$ and $\mathbf{p}_{j+1}$ can be calculated from:

$$\begin{bmatrix} \mathbf{B}_{jr} & \mathbf{B}_{ji} \\ -\mathbf{B}_{ji} & \mathbf{B}_{jr} \end{bmatrix} \begin{bmatrix} \mathbf{u}_j \\ \mathbf{v}_j \end{bmatrix} = \mathbf{0} \tag{B.40}$$

where $\mathbf{B}_{jr} = \mathbf{A} - \sigma_j\mathbf{I}$ and $\mathbf{B}_{ji} = \omega_j\mathbf{I}$. By definition, the complex-conjugate eigenvectors $\mathbf{p}_j$ and $\mathbf{p}_{j+1}$ satisfy:

$$\mathbf{A}\begin{bmatrix} \mathbf{p}_j & \mathbf{p}_{j+1} \end{bmatrix} = \begin{bmatrix} \mathbf{p}_j & \mathbf{p}_{j+1} \end{bmatrix} \begin{bmatrix} \lambda_j & 0 \\ 0 & \lambda_{j+1} \end{bmatrix} \tag{B.41}$$

Again, it may be verified that the vectors $\mathbf{u}_j$ and $\mathbf{v}_j$ satisfy:

$$\mathbf{A}\begin{bmatrix} \mathbf{u}_j & \mathbf{v}_j \end{bmatrix} = \begin{bmatrix} \mathbf{u}_j & \mathbf{v}_j \end{bmatrix} \begin{bmatrix} \sigma_j & \omega_j \\ -\omega_j & \sigma_j \end{bmatrix} \tag{B.42}$$

Note that in Eq.(B.41) the (2×2) diagonal block contains in the main diagonal the complex-conjugate pair of eigenvalues λ_j and λ_{j+1}, while in Eq.(B.42) the (2×2) block consists of the real numbers σ_j and ω_j, i.e. the real and imaginary parts of both λ_j and λ_{j+1}. Similarly, in Eq.(B.41) $\mathbf{p}_j$ and $\mathbf{p}_{j+1}$ are complex vectors, while in Eq.(B.42) only the real vectors $\mathbf{u}_j$ and $\mathbf{v}_j$ are used.

The algorithm *CHAC* determines all eigenvector chains associated with both λ_j and λ_{j+1} in a similar manner as is done in *CHAR*. The only differences are:

(i) Instead of the $(n \times n)$ matrix $\mathbf{B}_j$, given by Eq.(B.32), the following $(2n \times 2n)$ matrix $\mathbf{B}_j$ is built:

$$\mathbf{B}_j = \begin{bmatrix} \mathbf{B}_{jr} & \mathbf{B}_{ji} \\ -\mathbf{B}_{ji} & \mathbf{B}_{jr} \end{bmatrix} \tag{B.43}$$

where $\mathbf{B}_{jr} = \mathbf{A} - \sigma_j\mathbf{I}$ and $\mathbf{B}_{ji} = \omega_j\mathbf{I}$.

(ii) The nullity of $\mathbf{B}_j$ is $2\nu_j$, and its null space consists of $2n$-dimensional vectors $\mathbf{m}_j$ which can be represented by:

$$\mathbf{m}_j = \begin{bmatrix} \mathbf{u}_j \\ \mathbf{v}_j \end{bmatrix} \begin{matrix} \} \, n \\ \} \, n \end{matrix} \tag{B.44}$$

where the n-dimensional vectors $\mathbf{u}_j$ and $\mathbf{v}_j$ are defined by Eqs.(B.41) and (B.42).

(iii) In the eigenvector chains, instead of complex-conjugate eigenvectors $\mathbf{p}_j$ and $\mathbf{p}_{j+1}$, only the real number vectors $\mathbf{u}_j$ and $\mathbf{v}_j$ are used.

Calculation of range and null spaces is done using Algorithm *NRS*. Calculation of vectors which are outside the range space of a given matrix is done using Algorithm *INOU*.

Algorithm:

```
1.  Define ($n \times n$) matrix $\mathbf{A}$, real and imaginary parts $\sigma_j + j\omega_j$ of a
    complex-conjugate pair of eigenvalues of $\mathbf{A}$ and the scalar eps.
2.  Set number of columns in $\mathbf{A} \Rightarrow n$.
3.  Set $n+n \Rightarrow n_2$
4.  Set $\mathbf{I}_{n,n} \Rightarrow \mathbf{I}$
5.  Set $\mathbf{I}_{n_2,n_2} \Rightarrow \mathbf{I}_2$
6.  Set $\mathbf{0}_{n,0} \Rightarrow \mathbf{P}_j$
7.  Set $\mathbf{A} - \sigma_j\mathbf{I} \Rightarrow \mathbf{B}_{jr}$ ,  Set $\omega_j\mathbf{I} \Rightarrow \mathbf{B}_{ji}$
8.  Set $\begin{bmatrix} \mathbf{B}_{jr} & \mathbf{B}_{ji} \\ -\mathbf{B}_{ji} & \mathbf{B}_{jr} \end{bmatrix} \Rightarrow \mathbf{B}_j$
9.  Set $\mathbf{B}_j$, eps (NRS) $\Rightarrow \mathbf{N}, \mathbf{R}, x$
10. Set number of columns in $\mathbf{N} \Rightarrow v_j$
11. Set $\mathbf{I}_2 \Rightarrow \mathbf{B}_k$, $0 \Rightarrow k$, $0 \Rightarrow r$
12. Set $k+1 \Rightarrow k$
13. Set $\mathbf{B}_k \mathbf{B}_j \Rightarrow \mathbf{B}_k$
14. Set $\mathbf{B}_k$, eps (NRS) $\Rightarrow \mathbf{N}_k, \mathbf{Y}, x$
15. Set $\mathbf{R}, \mathbf{N}_k$, eps (INOU) $\Rightarrow \mathbf{Y}, \mathbf{M}$
16. Set number of columns in $\mathbf{M} \Rightarrow q$
17. Set the first $q/2$ columns from $\mathbf{M} \Rightarrow \mathbf{M}_r$
18. If $q > 0$, go to 19; else, go to 12
19. Set $\mathbf{M}_r \Rightarrow [\ \mathbf{m} \mid \mathbf{M}_r\ ]$
20. Set $\mathbf{m} \Rightarrow \begin{bmatrix} \mathbf{m}_r \\ \mathbf{m}_i \end{bmatrix} \begin{matrix} \}\ n \\ \}\ n \end{matrix}$
21. Set $\mathbf{0}_{n,0} \Rightarrow \mathbf{P}_i$,  Set $0 \Rightarrow i$
22. Set $i+1 \Rightarrow i$
23. Set $[\ \mathbf{m}_r \mid \mathbf{m}_i \mid \mathbf{P}_i\ ] \Rightarrow \mathbf{P}_i$
24. Set $\mathbf{B}_j\ \mathbf{m} \Rightarrow \mathbf{m}$
25. If $i < k$, go to 22; else, go to 26
26. Set $[\ \mathbf{P}_j \mid \mathbf{P}_i\ ] \Rightarrow \mathbf{P}_j$
27. If number of columns in $\mathbf{M}_r > 0$, go to 19; else, go to 28
28. Set $[\ \mathbf{R} \mid \mathbf{M}\ ] \Rightarrow \mathbf{R}$
25. Set $r+q \Rightarrow r$
26. If $r < v_j$, go to 12; else, stop
```

Algorithm Implementation:

The listing of Algorithm *CHAC*, implemented using the *L-A-S* language is given in Appendix C. The algorithms *NRS* and *INOU* are performed using the *L-A-S* operator NRS and subroutine INOU. Matrix partitioning in Steps 19 and 20 is done by the *L-A-S* operators CTC and CTR. Matrix concatenation in Steps 23, 26 and 28 is done using the operator CTI.

In Example 1 of Section 2.4 the real-number Jordan form, $\mathbf{A}_J$, and the corresponding modal matrix, $\mathbf{Q}$, was calculated for a (5×5) matrix $\mathbf{A}$ having both repeated roots and complex-conjugate roots. These two matrices can be obtained using the subroutine MODM as follows:

$$\mathbf{A}, \mathbf{Egv}, \epsilon, (MODM) \rightarrow \mathbf{Q}$$
$$\mathbf{Q}(-1), \mathbf{A}, \mathbf{Q}(*)(*) \rightarrow \mathbf{A}_J$$

B.5 Modal Controllability/Observability Tests

This section differs from the tests presented in Chapter 1 in that the method provides a "degree of controllability and observability" that goes beyond the "yes" or "no" tests studied there.

Introduction

It is well known that there are numerous procedures for checking the controllability and observability of state space representations of linear MIMO dynamic systems. Among the most popular are:

(1) Calculation of the ranks of controllability and observability matrices,
(2) Similarity transformation into the Jordan form state space representation,
(3) Kalman canonical decomposition,
(4) Transformation into the Hessenberg form, and the
(5) Popov-Belevitch-Hautus (PBH) test.

These procedures, being of different natures, have their own properties, advantages and disadvantages. Some are computationally ill-conditioned, some require extensive computation. And some procedures are not well suited in the case of multiple eigenvalues since they then require additional extensive computation. On the other hand, some procedures do not give information about the "degree" of controllability and/or observability, which is important in practical applications for numerical reasons.

In this section a simplified controllability/observability test is suggested. It is based on the PBH test, mentioned above, but it does not require calculation of the rank of n^{th} order matrices, n being the system order. Instead, it reduces to the calculation of the eigenvalues of a single matrix having an order less than the system order n. The calculation of this matrix is computationally straightforward. In addition to the information about controllability and observability, the present test also gives information about the *degree of controllability and observability*. The purpose of this section is to suggest a simplification of the PBH test.

A Simplified Observability/Controllability Test

Consider a sequence of equivalent n^{th} order state space representations $\{\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D}\}$, corresponding to a given linear MIMO dynamic system with m inputs and p outputs defined by a not necessarily minimal state space representation $R_o = \{\mathbf{A}_o, \mathbf{B}_o, \mathbf{C}_o, \mathbf{D}_o\}$ where:

$$\{\mathbf{A}, \mathbf{B}, \mathbf{C}\} = \{\mathbf{T}^{-1}\mathbf{A}_o\mathbf{T},\ \ \mathbf{T}^{-1}\mathbf{B}_o,\ \ \mathbf{C}_o\mathbf{T}\} \tag{B.45}$$

In Eq.(B.45) $\mathbf{T}$ is an arbitrary, random $(n \times n)$ non-singular matrix.

Without loss of generality, it may be assumed that there are no redundant inputs and outputs, i.e. that the rank of $\mathbf{B}$ is m and that the rank of $\mathbf{C}$ is p. The eigenvalues of $\mathbf{A}$, which are, of course, equal to those of $\mathbf{A}_o$, will be denoted by the set:

$$\lambda(\mathbf{A}_o) = \lambda(\mathbf{A}) = \Lambda = \{\lambda_i\} \quad , \quad i=[1,n] \tag{B.46}$$

Let $\mathbf{N}_b$ and $\mathbf{N}_c$ be orthonormal $(n \times n\text{-}m)$ and $(n \times n\text{-}p)$ matrices satisfying:

$$\begin{aligned} &\mathbf{N}_b^T\mathbf{B} = \mathbf{0}\,, \quad \mathbf{N}_b^T\mathbf{N}_b = \mathbf{I}_k\,, \quad k=n-m \quad \text{and} \\ &\mathbf{C}\mathbf{N}_c = \mathbf{0}\,, \quad \mathbf{N}_c^T\mathbf{N}_c = \mathbf{I}_r\,, \quad r=n-p \end{aligned} \tag{B.47}$$

The easiest way of calculating the orthonormal null space matrices $\mathbf{N}_b$ and $\mathbf{N}_c$ in Eq.(B.47) consists of performing the SVD of $\mathbf{B}^T$ and $\mathbf{C}$, respectively, i.e.:

$$\begin{aligned} \mathbf{B}^T &= [\,\mathbf{U}_b\,]\begin{bmatrix} \mathbf{S}_b & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}\begin{bmatrix} \mathbf{X}_b \\ \mathbf{N}_b^T \end{bmatrix}\begin{matrix} \}\ m \\ \}\ k=n-m \end{matrix} \\ \mathbf{C} &= [\,\mathbf{U}_c\,]\begin{bmatrix} \mathbf{S}_c & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}\begin{bmatrix} \mathbf{X}_c \\ \mathbf{N}_c^T \end{bmatrix}\begin{matrix} \}\ p \\ \}\ r=n-p \end{matrix} \end{aligned} \tag{B.48}$$

It should be pointed out that all matrices in Eqs.(B.47) and (B.48) depend on a similarity transformation $\mathbf{T}$, i.e.:

$$\mathbf{B} = \mathbf{B}(\mathbf{T}) \ , \ \ \mathbf{C} = \mathbf{C}(\mathbf{T}) \ , \ \ \mathbf{N}_b = \mathbf{N}_b(\mathbf{T}) \ , \ \ \mathbf{N}_c = \mathbf{N}_c(\mathbf{T}) \ , \ \ \text{etc.} \tag{B.49}$$

For simplicity of notation, however, the explicit dependence on $\mathbf{T}$ will be dropped.

Before stating the main result we need the following definition:

Definition: Let $\mathbf{P} = \mathbf{P}(\mathbf{T})$ be a $(k \times k)$ matrix-valued function of the $(n \times n)$ matrix $\mathbf{T}$ defined for almost every matrix $\mathbf{T}$. A set of fixed eigenvalues p_j, $j = [1, k^*]$, $k^* \le k$, of $\mathbf{P}(\mathbf{T})$ is a set of eigenvalues of $\mathbf{P}(\mathbf{T})$ that is invariant with respect to arbitrary variations of the matrix $\mathbf{T}$.

We can now prove the following results.

Theorem 1: The pair $\{\mathbf{A}_o, \mathbf{B}_o\}$ is controllable if the following $(k \times k)$, $k = n\text{-}m$ matrix:

$$\mathbf{P} = \mathbf{N}_b^T \mathbf{A} \mathbf{N}_b \tag{B.50}$$

depending on the similarity transformation $\mathbf{T}$, has no fixed eigenvalues with respect to arbitrary variations of matrix $\mathbf{T}$.

Theorem 2 (dual to Theorem 1): The pair $\{\mathbf{A}_o, \mathbf{C}_o\}$ is observable if the following matrix which has dimensions $(r \times r)$, $r = n\text{-}p$:

$$\mathbf{R} = \mathbf{N}_c^T \mathbf{A} \mathbf{N}_c \tag{B.51}$$

has no fixed eigenvalues with respect to arbitrary variations of matrix $\mathbf{T}$.

Proof of Theorem 1:

The PBH controllability test requires that the following $[n \times (n+m)]$ matrices $\mathbf{Q}_{ci}$, $i = [1, n]$, are of full rank, where

$$\mathbf{Q}_{ci} = \left[\ \mathbf{A} - \lambda_i \mathbf{I} \ | \ \mathbf{B} \ \right] \tag{B.52}$$

Premultiplying $\mathbf{Q}_{ci}$ with the $(k \times n)$, $k = n\text{-}m$, orthonormal null space matrix $\mathbf{N}_b^T$, defined in Eq.(B.47), yields:

$$\mathbf{N}_b^T \mathbf{Q}_{ci} = \left[\ \mathbf{N}_b^T \mathbf{A} - \lambda_i \mathbf{N}_b^T \ | \ \mathbf{0} \ \right] \tag{B.53}$$

From Sylvester's inequality, and for a controllable eigenvalue λ_i, it follows that:

$$\text{rank} \left[\ \mathbf{N}_b^T \mathbf{Q}_{ci} \ \right] = k \tag{B.54}$$

leading to:

$$\text{rank}[\mathbf{W}] = k\ , \quad \text{where} \quad \mathbf{W} = \mathbf{N}_b^T\mathbf{A} - \lambda_i\mathbf{N}_b^T \tag{B.55}$$

Now, postmultipying the $(n \times k)$ matrix $\mathbf{W}$ in Eq.(B.55) with $\mathbf{N}_b$ and taking into account Eqs.(B.47) and (B.50), one obtains:

$$\mathbf{W}\mathbf{N}_b = \mathbf{P} - \mathbf{I}_k\lambda_i \tag{B.56}$$

Using the Sylvester inequality again, since $k < n$, it follows that in the general case:

$$\text{rank}[\mathbf{P} - \mathbf{I}_k\lambda_i] \le k \tag{B.57}$$

Of course, the equality in Eq.(B.57) guarantees that no eigenvalue of the $(k \times k)$ matrix $\mathbf{P}$ is equal to λ_i, while in the case of strict inequality at least one eigenvalue of $\mathbf{P}$ is equal to λ_i.

Considering Eq.(B.56), it may be concluded that in Eq.(B.57), in the case of a controllable eigenvalue λ_i, the inequality will hold only when some column of $\mathbf{N}_b$ is in the null space of $\mathbf{W}$. However, since the similarity transformation matrix $\mathbf{T}$ influences matrices $\mathbf{W}$ and $\mathbf{N}_b$ in different ways, for an arbitrary matrix $\mathbf{T}$ we have:

$$\text{rank}[\mathbf{P} - \mathbf{I}_k\lambda_i] = k\ , \quad i = [1,n] \tag{B.58}$$

almost always, which proves Theorem 1, since from Eq.(B.58) it follows that no eigenvalue of $\mathbf{P} = \mathbf{P}(\mathbf{T})$ is equal to λ_i.

Equation (B.57) also indicates that, for a special selection of the matrix $\mathbf{T}$, it might happen that some of the eigenvalues of $\mathbf{P}$ are equal to some λ_i, even when this λ_i is a controllable mode of the pair $\{\mathbf{A},\mathbf{B}\}$. However, as was pointed out earlier, in the case of an arbitrary $\mathbf{T}$, it may be concluded that the condition:

$$\text{rank}[\mathbf{P}(\mathbf{T}) - \mathbf{I}_k\lambda_i] < k \tag{B.59}$$

will hold only for fixed eigenvalues of $\mathbf{P}(\mathbf{T})$, which are exactly the uncontrollable eigenvalues in the pair $\{\mathbf{A},\mathbf{B}\}$. This proves Theorem 1 as well as the following corollary. The proof of Theorem 2 is the dual of this proof.

Corollary: All fixed eigenvalues of matrices $\mathbf{P}$ or $\mathbf{R}$ are equal to some eigenvalues λ_i of $\mathbf{A}$, and they represent uncontrollable or unobservable eigenvalues of the pairs $\{\mathbf{A}_o,\mathbf{B}_o\}$ or $\{\mathbf{A}_o,\mathbf{C}_o\}$, respectively.

Degree of Controllability/Observability:

In the case that an eigenvalue λ_i is "almost" uncontrollable, it is natural to expect that some eigenvalue p_j of $\mathbf{P}(\mathbf{T})$ will be "almost" fixed, i.e. for various matrices $\mathbf{T}$ the eigenvalue p_j will be located in the s-plane within a small circle around λ_i. Therefore, as the degree r_i of controllability of λ_i, one may define the radius of the smallest circle in the s-plane, centered at the eigenvalue λ_i, encompassing all locations where, for various arbitrary matrices $\mathbf{T}$, the corresponding almost-fixed eigenvalue p_j falls. Thus, *the degree r_i of controllability of* λ_i can be written as:

$$r_i = \max_{\mathbf{T}} \{ \min_{j=[1,k]} \{ | \lambda_i - p_j(\mathbf{T}) | \} \} \tag{B.60}$$

In other words, the "maximum" operation is taken for that p_j that is closest to λ_i, i.e. only those p_j which correspond to the mode λ_i.

Since the concepts of controllability and observability were introduced in Chapter 1, an example is included with the end-of-chapter exercises there to illustrate the application of this method.

Algorithm Implementation

The *L-A-S* listing of Algorithm *COTS*, which performs the calculations for this method, can be found in Appendix C.

B.6 References

Brogan, W.L. (1991), *Modern Control Theory, 3rd Edition*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

Chen, C-T. (1984), *Linear System Theory and Design*, Holt, Rinehart and Winston, Inc., New York.

Kailath, T. (1980), *Linear Systems*, Prentice-Hall Inc., Englewood Cliffs, NJ.

Ackermann, J. (1985), *Sampled-Data Control Systems*, Springer-Verlag, Berlin.

Bingulac, S. and W. Luse (1989), "Calculation of generalized eigenvectors," *Journal of Computers and Electrical Engineering*, **15**, 1, 29-32.

Bingulac, S. and W. Luse (1990), "Computational simplification in controllability and observability tests," *Proceedings of the 28th Allerton Conference*, University of Illinois, October 3-5, 1990, pp. 527-528.

Appendix C   Introduction to *L-A-S*

In this appendix a detailed introduction to the *L-A-S* language is given, followed by the *L-A-S* code listings for the algorithms discussed in the text. This is a good place to begin for a serious study of this text. To help locate a particular algorithm once you are familiar with its application, a list of the algorithms generally in the order of their presentation is provided after the following Table of Contents.

Table of Contents for Appendix C

List of Algorithms

CHAPTER 1

```
par,zo,dz(Lin,sbr)=A,B,diff
par,z(gz,sub)=g
A,B(qc,sub)=Qc
A,C(qo,sub)=Qo
A,eps(nrs,sub)=N,R,r
A,B,C,D,xo,u,T(cdsr,sub)=y
A(Reso,sub)=p,Rr,R
A,B,C,D(Lalg,sub)=p,Rr,R
A,B,C,D(sstf,sub)=p,W
```

CHAPTER 2

```
A,B,C,D,T,Eps,Isrb(CTDT,sbr)=A1,B1,C1,D1
Ac,Bc,Cc,Dc,T,Eps(SRcd,sbr)=Ad,Bds,Bdr,Cd,Dds,Ddr
Ad,Bd,Cd,Dd,T,Eps(SRdc,sbr)=Ac,Bcs,Bcr,Cc,Dcs,Dcr
Ac,Bc,Cc,Dc,T,Eps(BLcd,sbr)=Ad,Bd,Cd,Dd
Ad,Bd,Cd,Dd,T,Eps(BLdc,sbr)=Ac,Bc,Cc,Dc
T,Ac,Nrm,N(Eat,sbr)=Ad
T,Ac,Eps(Eatj,sub)=Ad
T,Ac,B,Nrm,N(SIcd,sbr)=Ad,Bd
T,Ac,B,Nrm,N(RIcd,sbr)=Ad,Bdo,Bd1
T,Ac,Nrm,N(EATF,sbr)=Ad,E,F
T,Ad,Egm,N,eps(Lnm,sbr)=Ac
T,Ad,Eps(Lnmj,sub)=Ac
```

```
r,A(pom,sub)=R
p,A(polr,sub)=r
n(fact,sub)=f
N(fln,sub)=f
A4,B4,T,iodc(Bcdc,sub)=A5,B5o,B51,P
A,Bo,P,C,D,Eps(r5r4,sub)=Be,De
A,Be,P,C,De,Eps(r4r5,sub)=Bon,Dn
A,Eps(sqm,sub)=X
```

CHAPTER 3

```
A,B,C,T(str,sub)=At,Bt,Ct
Ind(SMat,sub)=Sa,Si,Sli,Sld
Q,mp,cut,Eps(Ind,sub)=Ind
A(C#,sub)=Adeg
vli,m(cind,sub)=Ind
A,B,Eps(cfpp,sbr)=Tc,Ind
A,B,Eps(cfns,sbr)=Tc,Ind
```

CHAPTER 4

Section 4.1

```
A,B,C,D,no,Eps(SSRo,sub)=Ao,Bo,Co,Do,Deg
A,B,C,D,nc,Eps(SSRc,sub)=Ac,Bc,Cc,Dc,Deg
A,B,C,D,M(SSH,sub)=H,hM
Ao,Bo,Co,Do,no(RoDN,sub)=D,N
Ac,Bc,Cc,Dc,nc(RcND,sub)=N,D
```

Section 4.2

```
d,W,Eps,nos(TFRo,sbr)=Ao,Bo,Co,Do,no,Cond
d,W,Eps,ncs(TfRc,sbr)=Ac,Bc,Cc,Dc,nc,Cond
d,W(TRon,sbr)=Ao,Bo,Co
d,W(TRcn,sbr)=Ac,Bc,Cc
d,W,Eps,nos(TFDN,sbr)=D,N,no,Cond
d,W,Eps,ncs(TFND,sbr)=N,D,nc,Cond
d,W,u,T(cdtr,sbr)=y
d,W,M(TFH,sbr)=H,hM
A,B,C(getd,sub)=n,m,p
p,m(dpm,sub)=P
m,L(ImL,sub)=ImL
den,num(ccf,sub)=A,b,c,d
d,GD(exD,sub)=G,D
d,GD(exD,sbr)=G,D
d,G,D(fgd,sbr)=GD
```

Section 4.3

```
H,Eps,nos(HRo,sbr)=Ao,Bo,Co,Do,no,Cond
H,Eps,ncs(HRc,sbr)=Ac,Bc,Cc,Dc,nc,Cond
H,Eps(HTF,sbr)=d,W
H,Eps(HTFp,sbr)=d,W
H,Eps(HTFm,sbr)=d,W
H,Eps,nos(HDN,sbr)=D,N,no,Cond
H,Eps,ncs(HND,sbr)=N,D,nc,Cond
u,H(uhy,sub)=y
H,f(Hf,sub)=Hf
```

Section 4.4

```
D,N,Eps(DNRo,sub)=Ao,Bo,Co,Do,no
N,D,Eps(NDRc,sub)=Ac,Bc,Cc,Dc,nc
D,N,M(DNH,sub)=H,hM
N,D,M(NDH,sub)=H,hM
D,N,Eps(DNTf,sbr)=d,W
N,D,Eps(NDTf,sbr)=d,W
no,n,f(Tscl,sub)=S
D,N,Eps,ncs(DNRc,sbr)=Ac,Bc,Cc,Dc,nc,Cond
N,D,Eps,nos(NDRo,sbr)=Ao,Bo,Co,Do,no,Cond
Dl,Nl,Eps,ncs(DNND,sbr)=Nr,Dr,nc,Cond
Nr,Dr,Eps,nos(NDDN,sbr)=Dl,Nl,no,Cond
```

CHAPTER 5

```
u,y,Eps,nos(uyRo,sbr)=Ao,Bo,Co,Do,no,xo,Cond
u,y,Eps,nos(uyDN,sbr)=D,N,no,Cond
u,y,Eps,nos(uyTF,sbr)=dtt,Wt,no,C#
N(pmt,sub)=Nt
G,Eps(Elzc,sub)=Gr
u,y,M(uyh,sub)=H,hM
Do,Eps(ComD,sbr)=comd,F
```

APPENDIX B

```
A,B,C,Eps(Min,sub)=Ao,Bo,Co,Tt
A,B,C,Eps(Min,sbr)=Am,Bm,Cm
A,B,C,Eps(Kald,sbr)=Ad,Bd,Cd,T,dim
R,Q,Eps(InOu,sub)=Qr,Qou
A,Egv,Eps(ModM,sbr)=P
A,sj,oj,Eps(ChaC,sbr)=Pj
A,sig,Eps(ChaR,sbr)=Pj
A,B,C,im,Eps(COts,sbr)=Resc,Reso,xxc,xxo
```

C.1 Introduction

The *Linear Algebra and Systems* (*L-A-S*) language is a high level interactive conversational language useful for the analysis and design of linear control systems. *L-A-S* is intended to be a handy, easy-to-use tool for verifying an analysis technique or a control design. Some of its unique features are:

1. As the user types and executes *L-A-S* commands, they are stored in the *L-A-S* interpreter memory, allowing them to be reexecuted within the same session using the same or different input.

2. Within the same session, the stored sequence of commands can be:
   a. —saved to disk for future execution,
   b. —reexecuted sequentially from the first command to the last,
   c. —reexecuted starting from any command in the sequence,
   d. The user can enter the TRACE mode where commands are executed one at a time.
   e. During the reexecution, it is possible to stop the sequential execution at any point in the sequence.

3. When the execution of an *L-A-S* program is stopped, it is possible to:
   a. —display and/or change any variable previously defined,
   b. —type and execute any additional command using previously defined variables,
   c. —define new variables,
   d. —modify any existing command,
   e. —delete any existing command,
   f. —include new commands,
   g. —reexecute existing commands individually,
   h. —resume normal sequential execution of the modified sequence of commands,
   i. —save the modified sequence to disk,
   j. —declare the sequence as an *L-A-S* subroutine, which allows the sequence to be invoked in a later session simply by specifying the name of the subroutine as well as the names of input/output variables to be used/defined by the subroutine, and
   k. —obtain on-line help on any aspect of *L-A-S* usage.

These features make the *L-A-S* software/language a unique computational environment for quick and user-friendly development and testing of a wide variety of algorithms in control, systems and signals areas. Once the algorithm has been tested and developed using *L-A-S*, it could later be easily implemented and reprogrammed using any programming language or CAD package. Also noteworthy is the availability of interface programs to exchange data between *L-A-S* and other engineering design/analysis packages. This is discussed later in this appendix.

All of the more than 200 existing *L-A-S* commands and subroutines are based on reliable public domain software packages, such as *Eispack* and *Linpack*, or on numerically proven algorithms published in technical journals. These commands and subroutines include:

- standard matrix manipulation
- array definition and plotting (including 3-dimensional plotting)
- classical SISO system analysis and design procedures (Bode, Nyquist, Root-Locus)
- solution of differential and difference equations
- calculation of system responses in frequency domain
- Fast Fourier transforms
- digital filter design
- optimal control
- solution of Riccati and Lyapunov matrix equations
- controllability and observability tests
- state and output feedback pole-placement in MIMO systems
- singular value decomposition
- similarity transformation
- eigenvalue and eigenvector calculation
- full or reduced order observer design
- LQR, LQG, and LQG/LTR design
- minimal realization
- system identification
- system linearization
- transformation from continuous-time system representation to an equivalent discrete-time representation and vice versa
- polynomial matrix manipulation
- operations with linear spaces and subspaces

L-A-S lends itself to simple modification of existing subroutines or developement and inclusion of user written subroutines. The on-line help facility contains quick information about the syntax and semantics of all *L-A-S* commands and subroutines. This appendix contains more in-depth descriptions of the *L-A-S* commands as well as helpful examples of their use.

***L-A-S* Language:** The *L-A-S* language is similar to reverse Polish notation in that inputs to a function are entered first, then the operator or function followed by an equals sign and, then, the output variables. As each statement is entered, the operation defined by that statement is performed prior to allowing the user to input the next statement.

Consider the following simple example program consisting of one comment line and 7 *L-A-S* statements:

```
1       _ExampC1
2       ,1/,,1/2,,-3(dma)=A
3       /,1/2,3,,(dma,t)=B
4       A,B(+)=C
5       C(-1,t)=Ci,det
6       Ci,C(*)=D
7       Ci,B(-),A(+,t)=Res
8       C,Ci,D,Res(out)=
```

The first line, Statement 1, is a comment. It usually contains a program name, which, in turn, corresponds to the file name with the extension ".DPF" containing this program. The next two statements, 2 and 3, define matrices A and B given by:

$$A = \begin{vmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 2 & 0 & -3 \end{vmatrix} \quad ; \quad B = \begin{vmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 2 & 3 & 0 \end{vmatrix}$$

The above matrices A and B could also be defined by the following more obvious, but more involved statements:

```
0,1,0/0,0,1/2,0,-3(dma)=A
0,0,0/0,1,0/2,3,0(dma)=B
```

It is our feeling that the versions which avoid entering of both leading and trailing zeros are more convenient, particularly for more experienced users. We believe that the readers will soon become proficient in *L-A-S* and that they will prefer to use a more concise version of the DMA (define matrix) operator. Note that in the suggested version of the DMA operator, instead of entering zeros explicitly, it suffice to enter a comma "," as an element delimiter and the slash "/" as a row delimiter.

In Statement 4 the matrices A and B are added to form matrix C. Statement 5 calculates the inverse of matrix C and its determinant. The results are assigned to Ci and det, respectively. Multiplying Ci with C, Statement 6 places the result, the identity matrix, in D. Statement 7 subtracts matrix B from Ci and then adds matrix A to the difference. The result is placed in matrix Res. Finally, Statement 8 types matrices C, Ci, D and Res to the screen.

The results obtained on the screen are as follows:

```
           C
  .000   1.000    .000
  .000   1.000   1.000
 4.000   3.000  -3.000
```

```
          Ci
 -1.500    .750    .250
  1.000    .000    .000
 -1.000   1.000    .000

          D
  1.000    .000    .000
   .000   1.000    .000
   .000    .000   1.000

          Res
 -1.500   1.750    .250
  1.000  -1.000   1.000
 -1.000  -2.000  -3.000
```

Organization of *L-A-S*: *L-A-S* consists of two different types of functions. The first, called *interpreter commands* (IC) are usually initiated by typing the three, or four, letter command, or its abbreviated version consisting of one, or two, characters. *L-A-S* then performs some task which allows the user to view, change or otherwise manipulate the current *L-A-S* program and data. The second type of statement is called an *operator statement* (OS). Operator statements have the following structure:

```
<label>:<inp-field>( <op-field> )=<out-field>
```

The terminal symbols ":", "(", ")" and "=" are used as field delimiters.

The **label**, (`<label>`) which is optional, is used in conjunction with program control operators for iterative and recursive calculations.

The **input-field**, (`<inp-field>`), contains variable names to be used by the operator.

The **operator-field**, (`<op-field>`), contains the mnemonic name of the function to be performed. An operator field is always enclosed in parentheses.

The **output-field**, (`<out-field>`), contains the variable names to which the outputs of the operator are assigned.

Statements 2 through 6 and 8 in the previous example program are examples of *single operator statements* (SOS), i.e. each statement contains only one operator. Statement 7 shows the use of *multiple operator statements* (MOS) since it contains more than one operator.

Note that the operator fields in Statements 3, 5 and 7, in addition to the operator name, contain the operator flag "t", separated by comma. If used, the flag "t" instructs the *L-A-S* interpreter to display on screen the results of this operator. In addition to the flag "t", it is also possible to use either:

"E" , "L" or "L,E" (or "e" , "l" or "l,e")

The functions of these operator flags are, respectively:

- —to display results on screen in "E" scientific format with 5 —significant digits
- —to print results on the specified print file
- —to print results on the specified print file in "E" format

Other options are described in the on-line Help file.

Multiple Operator Statements: To emphasize the usefulness of the multiple operator statement, (MOS), and to illustrate the use of some other *L-A-S* operators, consider the task of building the $(n \times n)$ matrix $\mathbf{A}_c$, Eq.(3.13) defined by:

$$\mathrm{Ac} = \left| \begin{array}{ccccc} 0 & 0 & 1 & \ldots & 0 \\ . & & & & . \\ . & & & & . \\ . & & & & . \\ 0 & 0 & 0 & \ldots & 1 \\ -a_0 & -a_1 & & \ldots & -a_{n-1} \end{array} \right|$$

given an $(n+1)$-dimensional row $\mathbf{a} = |\ a_0\ a_1\ \ldots\ a_{n-1}\ a_n\ |$ containing the coefficients a_i of the characteristic polynomial $a(s)$, $a(s) = \det(s\mathbf{I}\text{-}\mathbf{Ac})$, of $\mathbf{Ac}$.

Thus, given the row **a**, the matrix **Ac** may be built by the following two *L-A-S* multiple operator statements, see the *L-A-S* subroutine CCF.SUB:

```
1    a(cdi)(dec)=n
2    n(dec),n(dim)(shr),a,n(ctc),-1(s*)(rti,t)=Ac
```

Instead of these two MOS, the matrix Ac may also be built by the following sequence of *single operator statements* (SOS):

1	`a(cdi)=T1`	Extract column dimension of a ⇒ T1
2	`T1(dec)=n`	Decrement T1 ⇒ n
3	`n(dec)=T1`	Decrement n ⇒ T1
4	`T1,n(dim)=T2`	Define Identity matrix $I_{T1,n}$ ⇒ T2
5	`T2(shr)=T3`	Shift T2 by one column to right ⇒ T3
6	`a,n(ctc)=T4`	Cut (partition) by column a ⇒ T4; T4 has n columns
7	`T4,-1(s*)=T5`	Multiply T4 by the scalar -1 ⇒ T5
8	`T3,T5(rti,t)=Ac`	Concatenate T3 and T5 (row tie) by rows ⇒ Ac and display result on screen

Each MOS defines a number of temporary variables Ti, i=[1,k-1], k being equal to the number of operators in a MOS. After completion of a MOS, these variables are deleted and they are not available for further use. Up to ten operators may be combined in a MOS. Only variables appearing in the operator fields may be used in input fields of subsequent statements.

To better understand the implementation and use of MOS, consider once more the MOS 2 discussed above. In accordance with the algorithm representation given in the Preface, each operator may be represented by a block which performs a specific calculation. In other words, the MOS 2 from before may be interpreted by the sequence of calculations represented in Fig. C.1. The variables T_i, i=[1,5], appearing in Fig. C.1, are referred to as *generalized variables* `<gen-var>`. The syntax of a generalized variable may be represented by the following recursive definition:

```
<gen-var>    :=   <inp-field> ( <op-field> )
<inp-field>  :=  <blank>|<inp-arg>|<gen-var>|<inp-arg>
                 <inp-field>|<gen-var> , <inp-field>
```

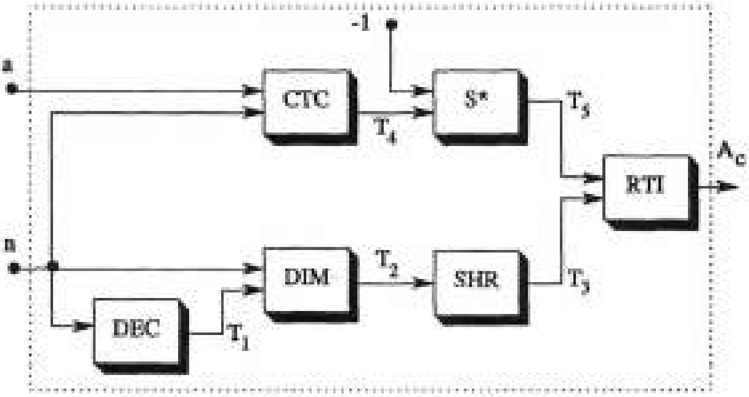


Figure C.1. Calculation Sequence for MOS *AC*

where: `<blank>` indicates a blank space,
`<inp-arg>` indicates an input argument, and
" | " represents a possible option.

In other words, an input field "through" an operator, i.e.

```
( <op-field> )
```

defines a generalized variable `<gen-var>`, consisting of a "list" of both

```
<inp-arg>  and  <gen-var>
```

separated by commas.

Note that in building a MOS the user should know the required "length" of the input field for each operator used in defining a generalized variable. This information may be obtained by the IC:

```
*  ope,<operator-name>
```

Also, recall that some of operators used in defining a generalized variable may have more than one output argument, as is the case in our example with the CTC operator. By the very definition of the MOS and the concept of a generalized variable, it should be realized that a generalized variable always corresponds to the first output argument of the operator used.

Post-Fix Notation: To get full benefit from the *L-A-S* software, users are urged to master the reverse Polish (post-fix) notation and the structure and operation of the MOS.

To assist readers in this task, let us review briefly the basic characteristics of the conventional "in-fix" notation.

If it is desired to add (or multiply) two numbers (or matrices), say **A** and **B**, and to place the result in **C**, the conventional in-fix notation is:

$$\mathbf{C} = \mathbf{A} + \mathbf{B}$$

where the opertor "+", addition, is placed in between the input arguments **A** and **B**. The result of the opertion, variable **C**, is on the left, separated from the structure "**A** + **B**" by the delimiter "=", the equal sign.

This in-fix notation works fine for "binary" operators, i.e. for operators requiring two input arguments. The operator can easily be placed between these two input arguments. In the case of "unary" operators, operators requiring only one input argument, say matrix transposition or inversion, it is customary to use superscripts as "*T*" and "-1", i.e.

$$\mathbf{A}^T \text{ and } \mathbf{B}^{-1}$$

leading, for instance, to:

$$\mathbf{D} = \mathbf{A}^T + \mathbf{B}^{-1} \tag{C.1}$$

The real problem occurs in scientific calculations where one is faced with "ternary" and, more generally, "*n*-nary" operators (algorithms) requiring three or more input arguments. As an example of a ternary operator consider building of the controllability matrix $\mathbf{Q}_c$, Definition 1.4, Section 1.3.4, which, in general, depends on three input arguments $\{\mathbf{A},\mathbf{B},k\}$, i.e.:

$$\mathbf{Q}_c = \left[\ \mathbf{B}\ \mid\ \mathbf{AB}\ \mid\ \cdots\ \mid\ \mathbf{A}^{k-1}\mathbf{B}\ \right] \tag{C.2}$$

where the integer k satisfies $1 \leq k \leq n$.

A good example of an "n-nary" operator, for n=4, is the algorithm *LNM*, natural log of a square matrix $\mathbf{A}$, Eq.(2.63), Section 2.3.1, which, as is explained there, depends on the following four input arguments: $\{\mathbf{A},\ T,\ N,\ \lambda_m\}$ and is given by:

$$\ln(\mathbf{A}) = -\frac{r}{T}\sum_{i=1}^{N}\frac{(\mathbf{I}-\mathbf{A}^{1/r})^i}{i} \tag{C.3}$$

where r is related to $\mathbf{A}$ and λ_m by Eq.(2.64).

The in-fix notation "followers" attempted to resolve these notational problems by resorting to the concept of the "subroutine" or "macro" widely used by various computer languages. Thus, the extensions of the in-fix notation to n-ary operators is:

$$Q_c = \mathrm{Qc}(\mathbf{A},\mathbf{B},k) \quad \text{and} \quad \ln \mathbf{A} = \mathrm{LNM}(\mathbf{A},T,N,\lambda_m)$$

in the cases of Eqs.(C.2) and (C.3), respectively.

Consider now the unlikely situation for which it is required to calculate either:

- —the product of Qc and ln **A**, or even
- —just ln **A** where **A** contains the first n columns of Qc in Eq.(C.2).

Then, the in-fix approach, provided that all input arguments are already defined, would be, for example:

```
Qc   = Qc(A,B,k)
ln A = LNM(A,T,N,λm)
Res1 = Qc * ln A
```

in the first case, and

```
Qc = Qc(A,B,k)
Qc1 = first n columns from Qc
Res2 = LNM(Qc1,T,N,λm)
```

in the second case. However, the post-fix, or reverse Polish, notation offers the following unified notation:

```
            A,B,k(Qc),A,T,N,λm(LNM)(*)=Res1
or          A,B,k(qc),n(ctc),T,N,λm(LNM)=Res2
```

in the above two considered cases, provided, of course, that:

ternary operator	A,B,k(Qc)=T1	or A,B,k(Qc) $\Rightarrow$ Qc
binary operator	T1,n(etc)=T2	or X,n(etc) $\Rightarrow$ Xn and
n-nary, for n=4, operator	T2,N,T,λ_m(LNM)=Res	or A,N,T,λ(LNM) $\Rightarrow$ Ac

are defined and available. Similarly, a version of the in-fix expression for **D**, given by Eq.(C.1), is:

$$\mathbf{D} = \mathbf{A}^T + \mathrm{inv}(\mathbf{A})$$

while the post-fix notation becomes:

$$\mathbf{A}(t),\mathbf{B}(-1)(+)=\mathbf{D}$$

where symbols (t), (-1) and (+) denote:

—unary operator of matrix transposition:	(t)
—unary operator of matrix inversion	(-1)
—binary operator of matrix addition	(+)

It is worth mentioning at this point that the way the algorithms are described in this book (see the Glossary) "mimics" the post-fix notation discussed here. This is the same notation adopted by Hewlett-Packard for their calculators.

Output Operator Options: Some of the useful options of the operator (out)= will be briefly reviewed here. The versions of the operator (out)= given below:

```
Res(out)=
Res(out,t,0)=
Res(out,t,1)=
Res(out,t,2)=
Res(out,e)=
```

display the previously mentioned matrix Res on the screen in the following forms:

```
          Res
  -1.500  1.750   .250
   1.000 -1.000  1.000
  -1.000 -2.000 -3.000


          Res
      -2.     2.      0.
       1.    -1.      1.
      -1.    -2.     -3.
```

```
          Res
  -1.5     1.8       .3
   1.0    -1.0      1.0
  -1.0    -2.0     -3.0


          Res
 -1.50    1.75      .25
  1.00   -1.00     1.00
 -1.00   -2.00    -3.00


          Res
-.15000E+01  .17500E+01  .25000E+00
 .10000E+01 -.10000E+01  .10000E+01
-.10000E+01 -.20000E+01 -.30000E+01
```

On the other hand, the versions:

```
A(out,L)=
A(out,L,0)=
A(out,L,1)=
A(out,L,2)=
A(out,L,e)=
```

print the matrix Res on the specified print file in the same forms as given above.

The versions:

Res(out,t,3)= and Res(out)=

as well as

Res(out,L,3)= and Res(out,L)=

are equivalent. More details on these and other options are available in the Help file.

Getting Started: *L-A-S* software may be accessed in any subdirectory provided that in this subdirectory the file DEFDSK has been copied from the master subdirectory, e.g. C:\LAS\. To begin an *L-A-S* session, simply type **LAS**, then, after two screens, the *L-A-S* interpreter issues a prompt: "*", informing the user that it is ready to accept *L-A-S* commands. Both interpreter commands and operator statements may be typed either by upper or lower case letters. However, the *L-A-S* interpreter makes the distinction between upper and lower cases in variable names and statement labels.

During an *L-A-S* session, DOS commands, such as changing directories, etc., may be sent to the command processor by first typing an exclaimation mark "!" and then the system command. This allows file manipulation without exiting *L-A-S*.

On-line help is available on every aspect of *L-A-S* by typing:

HELP (for help subjects)
HELP,ALL (for general help on the *L-A-S* interpreter)
HELP,OPE (for a brief description of all operators)
HELP,IC (for a brief description of all interpreter commands)
HELP,SUB (for a brief description of available subroutines)
HELP,EXA (for example programs)
HELP,LIM (for a description of the limitations of *L-A-S*)

To wipe out all existing variables and program statements and reset the program controls to their default values during an *L-A-S* session, simply type **BEG** **(B)** (for BEGIN).

To end the current session of *L-A-S* type either **END** or **QUIT (Q)**.

Program Creation: An *L-A-S* program may be either read in from disk or typed in interactively on the keyboard by typing a sequence of operator statements. All operator statements which are typed are memorized and will remain in the interpreter memory until they are manually removed or the session is ended. Therefore, the user may type in a sequence of statements to solve a problem with particular input data, recieve the output, change or modify the sequence of statements if desired, and reexecute the same sequence of operator statements with the same input data or different input data. Note that statements which evoke error messages are not saved in the current program.

Another aspect of program creation is the echo feature in *L-A-S*. During the current *L-A-S* session, all user input is written to a file named **ECHO.DAT**. This file is erased at the beginning of each new *L-A-S* session. The content of the **ECHO.DAT** file can be extremely useful by allowing the user to review the entire *L-A-S* session after the session has ended. Also, the content of this file may be used in conjunction with the interpreter command **FILE** to repeat the previously executed *L-A-S* session exactly.

Also note that all printer output from *L-A-S* goes to a specified "print file" rather than directly to the printer. The default print file name is **LASR** but can be changed. The user can access the print file after the current *L-A-S* session to modify or view the contents using any ASCII text editor. This feature also enables the user to write some output to the print file, rename the default print file, and then view the previously specified print file from within the current *L-A-S* session. Be forewarned that the current default print file may not be viewed while it is the default. This is inherently obvious since a file cannot be opened if it is already opened.

Data Types: Scalars, vectors, matrices, polynomials, polynomial matrices and character variables may be defined in *L-A-S*. All variable names however are required to be four characters long or less.

Simple Example: A simple example is included next to illustrate the use of *L-A-S*. Each OS and IC is reviewed. The following *L-A-S* program, named Sim-Ex (Simple Example), illustrates some basic *L-A-S* features.

```
 1        _Sim_Ex
 2        0,1,0/0,0,1/-4,-6,-4(dma)=Ao
 3        _Matrix_definition_Inversion
 4        _Eigenvalues_&_Display
 5     a:,1/,,1/-4,-6,-4(dma,t)=A
 6        Ao,A(-,t)=difA
 7        A(-1)=Ai
 8        A(egv,t)=eg
 9        eg(out,t,1)=
10        _Enter_nli_&_nty
11        _and_j,a
12        (sto)=
13        A(t),A(*,e)=AtA
14        AtA(out,L,2)=
15        A,eg,(Data)(wbf)=
```

- Statements 1, 3, 4, 10 and 11 are comments containing various information and suggestions.
- Statements 2 and 5 define equal (3 x 3) matrices **Ao** and **A**.
- Statement 5 has the *label* "a".
- Statement 6 calculates **Ao** - **A** ⇒ **difA**; which is a zero matrix.
- Statement 7 calculates the inverse of **A**; $\mathbf{A}^{-1}$ ⇒ **Ai**.
- Statement 8 calculates and displays eigenvalues of **A**; **A**(*Egv*) ⇒ **eg**.
- Statement 9 displays **eg** with only one decimal digit.
- Statement 12 is a "dummy," but a very useful, "STOP" statement.
- Statement 13 calculates $\mathbf{A}^T\mathbf{A}$ ⇒ **AtA**, and displays result in the E format.
- Statement 14 writes the array **AtA** on the print file LASR with two decimal digits.
- Statement 15 stores arrays **A** and **eg** on the "Disk Binary File:" Data.DBF. These arrays could be used as input data in any subsequent *L-A-S* session.

After initiating an *L-A-S* session and typing the above statements one at a time, the following should be obtained on the screen:

```
*   _Sim_Ex
*   0,1,0/0,0,1/-4,-6,-4(dma)=Ao

*   _Matrix_definition_Inversion

*   _Eigenvalues_&_Display

*   a:,1/,,1/-4,-6,-4(dma,t)=A

           A
   .000  1.000   .000
   .000   .000  1.000
 -4.000 -6.000 -4.000

*   Ao,A(-,t)=difA

  <Ao   >- <A    > =<difA>
   .000   .000   .000
   .000   .000   .000
   .000   .000   .000

*   A(-1)=Ai

*   A(eqv,t)=eg
Real & Imag. parts of eigenvalues of <A    >

           eg
 -2.000   .000
 -1.000  1.000
 -1.000 -1.000

*   eg(out,t,1)=
           eg
   -2.0      .0
   -1.0     1.0
   -1.0    -1.0

*   _Enter_nli_&_nty

*   _and_j,a

*   (sto)=

*   A(t),A(*,e)=AtA

     <.Ti  >   *   <A    >    =   <AtA >
   .16000E+02   .24000E+02   .16000E+02
   .24000E+02   .37000E+02   .24000E+02
   .16000E+02   .24000E+02   .17000E+02

*   AtA(out,L,2)=
        *   A,eg,{Data}(wbf)=
```

At this point it is suggested that the reader enter the following interpreter commands and monitor their effects on the program execution, although it may seem boring and time consuming. After that you will grasp the basic features which will enable you to effectively use the software in solving more complicated problems, even without the necessity of the rest of this appendix!

```
*   p,2,10      (or: pro,2,10;  or: program,2,10)          1
*   n           (or: names)                                 2
*   s           (or: status)                                3
*   pr          (or: prlisting)                             4
*   j,a         (or:m jump,a; or: jump,5)                   5
*   nl          (or: nlist)                                 6
*   j,a                                                     7
*   l           (or: list)                                  8
*   nt          (or: ntype)                                 9
*   s                                                      10
*   j,a                                                    11
*   nl                                                     12
*   j,a                                                    13
*   t           (or: type)                                 14
*   l                                                      15
*   c           (or: con;  or: continue)                   16
*   tra         (or: trace)                                17
*   j,a                                                    18
*   c            <three times>                             19
*   s                                                      20
*   ntr         (or: ntrace)                               21
*   w,prg1      (or: wpf,prg1)                             22
*   b           (or: begin)                                23
*   s                                                      24
*   n                                                      25
*   r,prg1      (or: rpf,prg1)                             26
*   e,10,12                                                27
*   e,3,4                                                  28
*   p                                                      29
*   inc,2,_New_version  (or: include,2,_xxx)               30
*   cha,1,Ex\Example  (or: change,1,Ex\New_Vers.)          31
*   pr                                                     32
*   w,prg2      (or: wpf,pgr2)                             33
*   p                                                      34
*   q           (or: quit;  or: end)                       35
```

These interpreter commands perform the following:

(1) —display on screen listing of Statements 2 to 10 of current *L-A-S* program,

(2) —display names and dimensions of currently defined arrays (matrices), i.e.:

```
   Ao        A         difA        Ai          eg          AtA
< 3, 3 > < 3, 3 >  < 3, 3 >  < 3, 3 >  < 3, 2 >  < 3, 3 >
```

(3) —display information about the *L-A-S* interpreter "status," i.e.:

```
Instr,PrgC,Char,Mem,List,Type,Trace,F-key,Test,#Data,
 15     15  246    0    0    0     0     0     0    51
                        #Matr
                          6
```

This information includes:

Instr	=	15; Program in *L-A-S* interpreter working memory has 15 statements
PrgC	=	15; Program counter is at Statement 15, i.e. at the end
Char	=	246; Program has 246 characters
Mem	=	0; Interpreter is in the "MEMORIZE" mode, i.e. any newly typed operator statement will be added to the program
List	=	0; Interpreter is in the "LIST" mode
Type	=	0; Interpreter is in the "TYPE" mode
Trace	=	0; Interpreter is in the "NO TRACE" mode
F-key	=	0; Interpreter is in the "NO Function-KEY" mode
Test	=	0; Interpreter is in the "NO TEST" mode
#Data	=	51; Total number of elements in defined arrays is = 51
#Matr	=	6; Total number of defined arrays is = 6

The above interpreter "modes" are the "default" modes. More details about the modes will be given later. To continue with the above listing of ICs, number:

(4) —writes all program statements on the print file LASR,

(5) —jumps back to the statement with the label "a", and reexecute sequentially all statements up to the first encountered (sto)= statement (At that point interpreter is ready to accept any operator statement or interpreter command.),

(6) —sets interpreter status to the "NO LIST" mode, i.e. during subsequent program reexecution the statements will not be displayed on screen,

(7) —"jumps" to the statement with the label "a" and reexecutes down to the (sto)= statement, but no statement will be displayed,

(8) —sets interpreter status back to the default "LIST" mode,

(9) —sets interpreter status to the "NO TYPE" mode, i.e. during reexecution the operator field flags "t", "e" or "L" will be ignored,

(10) Interpreter "status" has been changed: PrgC = 12, Mem = 1, Type = 1, i.e. it is in "NO MEMORIZE" and "NO TYPE" modes. (Any typed statement will only be executed, but it will not be added to the program.)

(11) —same as (5), but now during the reexecution, only the statements will be displayed (All operator field flags are suppressed.),

(12) —sets interpreter to the "NO LIST" mode,

(13) —same as (7), but now neither of the operator flags are in effect, nor the will any statements be displayed during reexection (Note that in this case only the responses to (out)= operators are shown on the screen.),
(14) —sets the interpreter back to the default TYPE mode,
(15) —sets the interpreter back to the default LIST mode,
(16) —reexecutes that part of the program after the (sto)= statement until eiither the last statement, or the next (sto)= statement,
(17) —sets the interpreter status to the "TRACE" mode,
(18) —jumps back to the statement with the label "a", but now since the status is "TRACE", only that statement will be reexecuted. (After that the interpreter "halts" in the same manner as if it encountered the (sto)= statement.) At this point the user may type any operator statement or interpreter command. It is suggested, as a response to prompt "*", to type, for instance:

```
*    eg(out)=
*    Ao(-1,t)=x,y
*    (rbf)=x,y,(Data)
*    x,y(out)=
*    s
*    n
```

and to observe changes in the interpreter status as well as number of arrays defined,
(19) —the three "c" (or: continue) interpreter commands permit execution of the three statements located below the statement labeled "a",
(20) The program counter is now at 5+3 = 8, since Statement 5 has label "a".
(21) —sets the interpreter back to the default "NO TRACE" mode (As a result, all statements up to the first (sto)= statement will be reexecuted sequentially.),
(22) The current *L-A-S* program is stored in the program file under the name Prg1.

The file name is Prg1.DPF. "DPF" (Disk Program File) is the standard file extension. The programs stored on DPF can be retrieved and reexecuted with the same or different input data. Also, if desired, as will be seen below, some of the statements may be changed, deleted, or new ones may be included.

(23) —deletes all existing arrays and statements, and all interpreter flags are reset to their default values (This is equivalent to ending the *L-A-S* session and initiating another one.),
(24) —the message:

```
The L-A-S symbol table is empty
Insrt,Prg.C,Char,Mem,List,Type,Trace,F-key,Test,#.Data,
  0     0     0    0    0    0     0     0     0     0
                           #.Matr
                             0
```

indicates that system table is empty; neither program nor arrays have yet been defined,

(25) —the message:

```
The L-A-S symbol table is empty
```

indicates that no arrays have yet been defined,

(26) —reads the program previously stored on the DPF and prepares for reexecution and/or modifications,
(27) —erases that part of the program between Statements 10 to 12 (Statement numbers are automatically resequenced.),
(28) —erases that part of the program between Statements 3 to 4,
(29) —displays the modified program on the screen,
(30) —adds a new comment after Statement 2 (This new statement has the statement number 3.),
(31) —changes the string of characters "Ex" in statement 1 to "Example",
(32) —prints the listing of the modified program on print file LASR,
(33) —stores the current program on DPF under the name Prg2,
(34) —displays the current program on screen, and
(35) —quits (ends) the current *L-A-S* session.

To get more insight into the *L-A-S* interpreter functioning, it is suggested that the reader use any text editor to examine the contents of:

	print file:	LASR
and	"ECHO" file:	ECHO.DAT

More details about the used operator statement and interpreter commands may be found in the Help file.

C.2 A List of *L-A-S* Operators

MNEMONIC NAME	DESCRIPTION
*	Matrix (array) multiplication
+	Matrix (array) addition
−	Matrix (array) subtraction
-1	Matrix (array) inversion (and determinant calculation)
ABS	Absolute value of an ($n \times m$) array or integer
ALT	Alternate polynomial matrix forms (PMF-c & PMF-r)
ATG	ArcTanGent of an ($n \times m$) array or integer
BEL	Activates computer bell

BOD	Calculation of the frequency (Bode) diagram
C*	Complex function multiplication
C/	Complex function division
C2R	Alternate PMF-c ⇒ PMF-r
CCON	Cascade connection of two subsystems
CDI	Define column dimension of matrix
CE3	Response of a linear continuous system in state space
CHD	Continuous roots of a discrete characteristic polynomial
CHE	Polynomial roots. (Char. poly. ⇒ Eigenvalues)
CLS	Closed loop SISO system
CMP	Copy matrix into the polynomial matrix form (PMF)
COIN	Copy integer into a scalar variable
COS	COS(x) of an array or integer
COT	Controllability and observability test
CPM	Copy polynomial matrix into a general matrix
CTC	Matrix (array) cut by columns (partition)
CTI	Matrix (array) column concatenation (columns tie)
CTR	Matrix (array) cut by row (partition)
CUR	Cube root of an array or integer
D2NV	$D(s)$ in PMF-r (monic) ⇒ PCI/POI
DCH	Define character string (variable)
DCV	Define column vector with integer entries
DDM	Define diagonal matrix
DE1-DE9	Simulation of nonlinear discrete systems
DEC	Decrement ($n \times m$) array elements by one
DFI	Define file with ASCII characters
DFT	Direct fast Fourier transform
DIIM	Definition of "inverted" identity matrix
DIM	Definition of an identity matrix
DIS	Time response plotting
DISD	Time response plot—only points are displayed
DISL	Plot with log (logarithmic) scale
DLD	Plot with log scale—only points are displayed
DMA	Define matrix with real number entries
DPM	Define pseudo-random matrix
DSC	Define scalars (input from terminal keyboard)
DSM	Define selector (permutation) matrix
DVC	Define vector (joins scalars into a row vector)
DZM	Definition of a zero matrix
EATF	Matrices; exp(AT), **E** and **F**; discretization
EFJF	Diag matrix of f(egi); f(x)=x|exp(x)|ln(x)|sq(x)|
EGC	Eigenvalues ⇒ characteristic polynomial
EGV	Eigenvalues of a square matrix
ELM	Eliminate matrices from *L-A-S* working memory

ELZ	Eliminate last zeros in a row
EMD	Extract main diagonal from a matrix
EXM	Extract matrix (or scalar) from a matrix
EXP	The exponential function element-by-element of an array
F*	Function multiplication
F/	Function division
FCON	Feedback connection of two subsystems
GS	Evaluation of $G(s)$; G = polynomial matrix; $s = a+jb$, a complex number
GTS	Generate time scale
IFJ	If jump (conditional jump)
IFT	Inverse fast Fourier transform
INC	Increment ($n \times m$) array elements by one
INP	Matrix/array input from terminal keyboard
INPM	Matrix/array input with specified dimensions
INT	Integer parts of ($n \times m$) array elements
INV	Pseudo matrix-inversion using singular value decomposition
JFR	Modal matrix and Jordan form of a square matrix without generalized eigenvectors
JMP	Unconditional jump
KRPR	Kronecker product of two matrices
LAP	Solution of the linear matrix Lyapunov equation
LIS	Enter list mode
LNM	Ln(A) of an ($n \times n$) array (square matrix)
LOG	Ln(x), element-by-element, of an ($n \times m$) array
LYP	Solution of the matrix eq. $\mathbf{AX} + \mathbf{XB} = \mathbf{C}$
MAX	Maximum element of an array
MCP	Matrix copy
MIN	Determination of the minimal realization (Hessenberg)
MTF	Calculation of the matrix transfer function
MTV	Matrix to vector transformation
NBE	Deactivates computer bell
NIK	Frequency (Nyquist) response plotting (x-y plot)
NINP	Define column dimension of a polynomial matrix in PMF, i.e. # of inputs
NLI	No list—exit list node
NOP	No operation
NRC	Matrix norm and norms of each column
NRR	Matrix norm and norms of each row
NRS	Null- ,range-space and rank
NTE	Exit test mode
NTR	Exit trace mode
NTY	No type—exit type mode
NYQ	Calculation of the frequency (Nyquist) diagram

ORD	Ordering of vector elements
OUT	Display the results on the terminal screen
OUT,L	Print the results to the print file
P*	Polynomial multiplication
P+	Polynomial addition
P-1	Inversion of polynomial matrix
P3D	Three dimensional (3-D) plot
PCH	Print character variables
PCON	Parallel connection of two subsystems
PFI	Print file (write a file to the print file)
PLL	Printer plot to print file
PLT	Printer plot on terminal
PMA	Polynomial matrix addition
PMFC	Alternate PMF-c ⇒ polynomial matrix form (PMF)
PMFR	Alternate PMF-r ⇒ polynomial matrix form (PMF)
PMI	Polynomial matrix input from terminal keyboard
PMM	Polynomial matrix multiplication
PNR	Polynomial normalization (reduction to monic form)
POI	Pseudo-observability/controllability indices
POLR	Polynomial reduction using C-H Theorem
POM	Polynomial of a square matrix $\mathbf{A}$; $c(\mathbf{A}) \Rightarrow \mathbf{R}$
PRD	Polynomial reduction to the *coprime* form
PRT	Polar to rectangular transformation
QC	Controllability matrix of the pair $\{\mathbf{A},\mathbf{B}\}$
QO	Observability matrix of the pair $\{\mathbf{A},\mathbf{C}\}$
R2C	Alternate PMF-r ⇒ PMF-c
RBF	Read binary file; read data from a binary data file
RCS	Response of a continuous system in state space
RCT	Response of a continuous system given by a transfer function matrix
RDF	Read data file; reading from an ASCII data file
RDI	Define row dimension of matrix
RDS	Response of a discrete system in state space
RDT	Response of a discrete system given by a transfer function matrix
RIC	Solution of the algebraic matrix Riccati equation using eigenvector Hamiltonian approach
RKC	Rank calculation and separation of linearly independent and dependent columns
RKR	Rank calculation and separation of linearly independent and dependent rows
RLC	Root-locus calculation
RMP	Replace matrix part
RPT	Rectangular to polar transformation

RTI	Matrix (array) row concatenation (tie by rows)
S*	Matrix (array) multiplication by a scalar
S/	Matrix (array) division by a scalar
S/X	(sinx)/x of an array or integer
SHD	Matrix (array) shift down one row
SHL	Matrix (array) shift left one column
SHR	Matrix (array) shift right one column
SHU	Matrix (array) shift up one row
SIN	sin(x) of an array or integer
SLE	Solution of linear equations
SQM	Square root of a square matrix
SQR	Square root of an array or integer
SSTF	Calculation of the matrix transfer function
STEP	Define an array with all entries equal to one
STO	Stop program execution
STR	State space transfomation
SVC	Singular-value decomposition of complex matrix
SVD	Singular-value decomposition
T	Matrix (array) transposition
TCH	Type character variables
TES	Enter test mode—not to be used by *L-A-S* users
TFI	Type file with ASCII characters
TFSS	Transfer function matrix ⇒ state space (Hessenberg)
TIME	Get time in seconds since beginning of *L-A-S* session
TOEP	Building a Toeplitz matrix
TR	Trace of a matrix (array)
TRA	Enter trace mode
TVC	Transforms (partitions) row vector into scalars
TXT,L,*text*	Writes arbitrary text to the print file
TXT,T,*text*	Displays arbitrary text on the terminal screen
TYP	Type—enter type mode
TZS	Transmission zeros (generalized eigenvalue problem)
VTM	Vector to matrix transformation
WBF	Write binary file; write data to a binary data file
WDF	Write data file; write to an ASCII data file
XLAB	Label *x*-axis of the plot
XYP	*x*-*y* Plotting (Nyquist and root-locus)
YLAB	Label *y*-axis of the plot
YXSC	Set scales for *y*- and *x*-axes of a plot

Detailed syntactical description of each operator statement may be obtained by typing **HELP,xyz**; where **xyz** stands for the mnemonic name of an operator statement.

C.3 *L-A-S* Subroutines

In addition to the *L-A-S* operators listed above, the *L-A-S* software contains large number of "macros," referred to as *L-A-S* subroutines. In the software there are two type of subroutine, namely:

- —subroutines of the type "SUB" and
- —subroutines of the type "SBR"

Both subroutine types have a syntax similar to the syntax of *L-A-S* operators. They consist of a sequence of *L-A-S* statements, but can be executed by referring to the subroutine name only, i.e.

```
<label>:<inp-field>( <sub-n>,SUB )=<out-field>        or
<label>:<inp-field>( (sbr-n>,SBR )=<out-field>
```

for "SUB" and "SBR" subroutine, respectively, where < sub-n > and < sbr-n > are subroutine names assigned during the subroutine definition.

Note that the only difference with respect to an *L-A-S* operator statement is that the "operator field," in addition to the name, contains also a specifier:

```
,SUB                    for a "SUB" subroutine  and
,SBR                    for a "SBR" subroutine,
```

separated by a comma ",".

All current *L-A-S* subroutines reside in *L-A-S* master subdirectories:

C:\LAS\SUB and C:\LAS\SBR

respectively.

Once in an *L-A-S* session, a subroutine may be checked by:

```
*  r,<sub-n>.SUB          or
*  r,(sbr-n>.SBR          and
*  p                  or  program,n1,n2
```

By checking listings of available subroutines, it is relatively easy to figure out how it is possible to define other subroutine solving a specific analysis/design problem.

The information about names and input/output arguments of currently available subroutines may be obtained by the IC:

```
*  h,sub
```

It is worth mentioning that subroutines of the "SUB" type can not call another subroutine. Also operators (STO)= and (DMA)= are not permitted. On the other hand, subroutines of the type "SBR" may contain calls to other defined subroutines of either type. A subroutine of type "SBR" can even call itself, thus is capable performing recursive calculations very effectively. In return, execution of subroutines of the type "SUB" is slightly shorter than that of "SBR" subroutines.

Omitting Input, Output and Operator Fields: The number of elements in the input and output fields is determined by the operator statement syntax, see the Help file for more details. If the input or output fields contain an insufficient number of array names, the *L-A-S* Interpreter issues an appropriate error message. However, if the *entire* field is omitted, then the *L-A-S* interpreter prompts the user to enter the desired array names. This is convenient for performing calculations with different input data and/or to define different data using the single statement. As an example, consider that an $(n \times n)$ matrix **A** and three $(n \times m)$ matrices **B1**, **B2** and **B3** are already defined. Then, for instance, the operator statement:

```
A,B1(Qc)=Qc1
```

calculates the controllability matrix $\mathbf{Q}_{c1}$ of the pair {**A**,**B1**}. However, if the following "incomplete" statement is typed:

```
(Qc)=
```

then, at execution time, the user has an opportunity to specify the arrays to be used by the operator, as well as the names of the arrays containing the results. In the above case, if:

```
A,B3        is typed for the input field, and
Q3          is typed for output field
```

then, of course, the operator QC calculates the controllability matrix $\mathbf{Q}_{c3}$ of the pair {**A**,**B3**}. This idea has been extended to the operator field as well. Thus, the following "completely incomplete" statement:

```
()=
```

may be considered as a "general" operator statement by which any operator using any input arrays and defining any output array may be executed. It may be said that in this case *L-A-S* enters into a "question and answer" mode, which experienced users tend to avoid. Note that whenever in the current *L-A-S* program an "incompletely" specified operator statement is to be executed, the user has to specify the missing field (input, operator or output).

Using Integers in Input Fields: Operators, as well as subroutines of the type "SUB" requiring scalars as input arguments, may, if the scalar is equal to an integer, be executed by specifying the integer directly in the input field. For example, if $n=2$ and $m=3$, then the statement:

```
n,m(Din)= I
```

defines the identity matrix $$\mathbf{I}_{n,m} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

In the cases of `2,3(Dim)=I1` , or `n,3(Dim)=I2` , etc., the same results are obtained. Similarly, the "SUB" subroutine:

```
*    n,m,p(Abcd,sub)=A,B,C,D
```

defines an arbitrary, random n^{th} order MIMO state space representation $R = \{\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D}\}$ with m inputs and p outputs. Thus, if the following statement is typed:

```
*    5,3,2(Abcd,sub)=A1,B1,C1,D1
```

then $R1 = \{\mathbf{A1},\mathbf{B1},\mathbf{C1},\mathbf{D1}\}$ corresponds to a 5th order MIMO system with 3 inputs and 2 outputs. Note that this is not applicable to subroutines of the type "SBR."

Interactive Data Definition: So far, as operators for defining data, only the operators DMA and RBF have been mentioned. In order to allow more flexibility in the *L-A-S* software, there are several *interactive* data definition operators. At execution time these operators prompt user to enter desired data, which is very useful for checking algorithms with different input data. An example illustrating the two most commonly used input operators is given below:

```
*    (dsc)=k
*    k,3(inpm)=A
```

The operator DSC (Define Scalar) types the name "k" on the screen and prompts user to specify the scalar k, while the INPM (Input Matrix) operator, in the above statement types the name "A" on the screen and prompts user to type elements of the $(k \times 3)$ matrix $\mathbf{A}$. Assuming that $k = 3$, then, if the following matrix $\mathbf{A}$ is to be defined:

$$\mathrm{A} = \begin{matrix} 1.0 & 0.0 & 2.5 \\ 1.2\times10^{-5} & 0.0 & 0.0 \\ 0.0 & -5.5 & 0.0 \end{matrix}$$

the user has to type:

```
1,,2.5          <return>
1.2e-5          <return>
,-5.5           <return>
```

or:

```
1,0,2.5         <return>
0.000012,0,0    <return>
0,-5.5,0        <return>
```

As was mentioned earlier, see Example C.1, Section C.1, the first version which avoids entering of both leading and trailing zeros is considered more convenient. For better readability arbitrary number of blank characters may be added, if desired.

Default Values for *L-A-S* Operators: Some *L-A-S* operators may be executed using a lesser number of input arguments than the maximum number defined in the Help file. In this case for unspecified input arguments the *default* values are used. The information about the default values is contained in the Help file. To clarify this feature consider some examples:

The statements:

```
*    A,B(Qc)=Qc
*    A,B,k(Qc)=Qc1
```

calculate:

$$\begin{aligned} Qc &= \mid B \mid A\,B \mid \ldots \mid A^{n-m}\,B \mid \\ Qc1 &= \mid B \mid A\,B \mid \ldots \mid A^{k-1}\,B \mid \end{aligned}$$

respectively. In other words, the default value of the third argument in the Qc operator is $k = n\text{-}m+1$.

The statements:

```
*    A(sstf) = d
*    A,B,C(sstf)=d,Wsp
*    A,B,C,D(sstf)=d,W
```

calculate row **d**, and matrices **Wsp** and **W** in PMF representing $d(z) = \det(z\mathbf{I}\text{-}\mathbf{A})$, $Wsp(z) = \mathbf{C}\ \mathrm{adj}(z\mathbf{I}\text{-}\mathbf{A})\ \mathbf{B}$ and $W(z) = \mathbf{C}\ \mathrm{adj}(z\mathbf{I}\text{-}\mathbf{A})\ \mathbf{B} + d(z)\ \mathbf{D}$.

In other words, the default value for the fourth argument in operator SSTF is a zero matrix. Also, if only one input argument is specified, then SSTF calculates only the row **d**.

The statement:

```
*    T,Ac,Nrm,N(Eatf)=Ad,E,F
```

calculates matrices **Ad**, **E** and **F**, see Chapter 2, using the specified values for *Nrm* and *N*, while the statement

```
*    T,Ac(Eatf) = Ad1,E1,F1
```

uses for the third and fourth arguments the default values given by: $Nrm = 0.5$ and $N = 16$. Let us mention that the statement:

```
*    T,Ac(Eatf)=Ad
```

defines only the matrix **Ad**.

The statement:

```
*    T,Ad,Egm,N,Eps(Lnm)=Ac
```

calculates $\mathbf{Ac} = Ln(\mathbf{Ad})/T$, see Chapter 2, using for *Egm*, *N* and *Eps* the specified values. The statement:

```
*    T,Ad(Lnm)=Ac1
```

calculates **Ac1** using for *Egm* and *N* the default values given by: $Egm=0.25$ and $N=36$, while *Eps* is either equal to 10^{-16} or to a value set previously by the IC *Eps*.

The statement:

```
*   x0,N,T,A,B,u(CSR)=x,tv
```

calculates a continuous system response to both initial condition **x0** and input **u**(t), i.e. a solution **x**(t) of the linear differential equation:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\,\mathbf{x}(t) + \mathbf{B}\,\mathbf{u}(t)\ ,\quad \mathbf{x}(0) = \mathbf{x0}$$

for $0 \le t \le T$, in N points, with the N dimensional column **tv** equal to:

$$\mathbf{tv} = [\,0 \quad dt \quad 2dt \quad \cdots \quad (N-1)dt\,]^T\ ,\quad dt = \frac{T}{(N-1)}$$

On the other hand:

```
*   x0,N,T,A,B(CSR)=xs
```

defines **xs**(t) as the system response to a step input and **x0**, while

```
* x0,N,T,A(CSR)=xi
```

calculates **xi**(t) as the response to initial conditions only, i.e. assumes that both **B** and **u** are zero. Specific details about default values are given in Help file.

Number of Output Arguments: If an operator statement has more than one output argument, then it not always necessary to specify all outputs arguments. The operator defines only the specified output variables.

Consider, for example:

```
*   A(Jfr)=M,Aj
*   A(Jfr)=M
```

In the first version both the modal matrix **M** and the corresponding diagonal Jordan form **Aj** of the *diagonalizable* square matrix **A** are defined, while the second version defines only the modal matrix **M**.

Similarly, given matrices **A** and **B**, the following statements:

```
*   A,k(ctc)=A1,A2
*   B,k(ctr)=B1,B2
```

partition (cut by columns/rows) the matrices **A** and **B** into

```
                                         | B1 |
 A  =>  | A1 | A2 |     and     B  =>    | -- | ,
                                         | B2 |
```

where **A1** and **B1** have k columns and rows, respectively, while the statements:

```
*   A,k(ctc)=A1
*   B,k(ctr)=B1
```

define only **A1** and **B1**. Note that if $k = 0$, then **A1** has *zero* columns and **B1** has *zero* rows. Also, if only one input argument is specified, both operators prompt the user to specify the value of k.

Defining Matrices in the PMF: Since this text deals extensively with polynomial matrices, possible ways of defining and manipulating matrices in the PMF are important.

Consider two polynomial matrices W and V given by:

$$W = \begin{bmatrix} s & 1+s & -1+s \\ s^2 & 2+3s & s+4s^2 \end{bmatrix}, \quad V = \begin{bmatrix} s & 2+3s \\ s^2 & -1+s \\ 1+s & s+4s^2 \end{bmatrix}$$

Obviously W is (2×3), while V is (3×2). Some polynomial elements $w_{ij}(s)$ in $W(s)$ have been selected to be equal to some elements in $V(s)$. The *non-standard*, but nevertheless convenient way of defining matrices W and V in PMF, which allows their manipulation in *L-A-S* is as follows:

Define a *general* (6×3) matrix **X**:

$$\mathbf{X} = \begin{vmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \\ 2 & 3 & 0 \\ -1 & 1 & 0 \\ 0 & 1 & 4 \end{vmatrix}$$

which, of course, can be done by:

```
*   ,1/,,1/1,,1/2,3/-1,1/,1,4(dma,t)=X
```

Then, using the CMP operator (Copy Matrix into PMF), i.e.

```
*   X,3(cmp)=W
*   X,2(cmp)=V
```

defines **W** and **V** to be in PMF. To realize what the operator CMP actually does, note that the following OUT statement:

```
*   X,W,V(out,t,2)=
```

displays on the screen:

```
           X
     .00   1.00    .00
     .00    .00   1.00
    1.00    .00   1.00
    2.00   3.00    .00
   -1.00   1.00    .00
     .00   1.00   4.00

           W
     .00   1.00    .00
     .00    .00   1.00
    1.00    .00   1.00
    2.00   3.00    .00
   -1.00   1.00    .00
     .00   1.00   4.00
 Polynomial matrix <W   > has   3 columns

           V
     .00   1.00    .00
     .00    .00   1.00
    1.00    .00   1.00
    2.00   3.00    .00
   -1.00   1.00    .00
     .00   1.00   4.00
 Polynomial matrix <V   > has   2 columns
```

In other words, the operator CMP "copies" **X** into **W** and **V** but, at the same time, declares them as matrices in PMF, i.e. matrices whose rows contain the coefficients of the polynomials $wij(s)$ and $vij(s)$ in (2×3) and (3×2) polynomial matrices $W(s)$ and $V(s)$, respectively. This allows matrices in PMF to be used as input arguments in the "polynomial matrix" manipulation operators and subroutines, such as:

```
  PMA, PMM, P-1, ALT, TFSS, CCON, FCON, PCON, RCT, RDT,
             PMT.SUB , DNRo.SUB, NDRo.SBR, etc.
```

For instance, the sequence:

```
*   W,V(pmm)=WV
*   W(pmt,sub)=Wt
*   WV(p-1)=WVad,det
*   Wt,V(pma)=WtV
*   WV,Wt,WVad,det,WtV(out,t,1)=
```

calculates polynomials matrices, all in the PMF, corresponding to:

$$\begin{aligned}
WV(s) &= W(s)*V(s)\\
Wt(s) &= W^T(s)\\
WVad(s) &= \mathrm{adj}(\ WV(s)\ )\\
det(s) &= \det(\ WV(s)\ )\\
WtV(s) &= W^T(s) + V(s)
\end{aligned}$$

and displays on the screen:

```
           WV
    -1.0     1.0     1.0     1.0     1.0
      .0     1.0     6.0     5.0     4.0
    -1.0     2.0    -1.0     5.0      .0
    -2.0    -1.0     6.0    11.0    16.0
 Polynomial matrix <WV  > has   2 columns

           Wt
      .0     1.0      .0
     1.0      .0     1.0
    -1.0     1.0      .0
      .0      .0     1.0
     2.0     3.0      .0
      .0     1.0     4.0
 Polynomial matrix <Wt  > has   2 columns

           WVad
    -2.0    -1.0     6.0    11.0    16.0
      .0    -1.0    -6.0    -5.0    -4.0
     1.0    -2.0     1.0    -5.0      .0
    -1.0     1.0     1.0     1.0     1.0
 Polynomial matrix <WVad> has   2 columns
           det
  2.0    .0   -5.0  -14.0   -7.0   -1.0   12.0   7.0   16.0

           WtV
      .0     2.0      .0
     1.0      .0     2.0
      .0     1.0     1.0
     2.0     3.0     1.0
     1.0     4.0      .0
      .0     2.0     8.0
 Polynomial matrix <WtV > has   2 columns
```

The *L-A-S* operator CPM (copy PMF into a matrix) may be considered as an "inverse" to the operator CMP, in the sense that the statement:

```
            *   W(cpm)=XX      as well as   *    V(cpm)=XXX
```

would produce (6 × 3) "general" matrices XX and XXX exactly equal to the matrix X mentioned above.

The so-called "standard way" of defining polynomial matrices, i.e. matrices in PMF, consists of using the PMI (polynomial matrix input) operator. To define **W** and **V** by the PMI operator, the following should be done. In an *L-A-S* session type:

```
            *   (pmi,m)=W,V
```

The PMI operator prompts the user to enter the dimensions and maximum orders of the polynomials, i.e. to enter values for:

$$p, m, n \ \{\text{for both } W(s) \text{ and } V(s)\}$$

If for **W** the values 2,3,2 are typed, and for **V** the values 3,2,2, see the example below, then the PMI operator expects (for both **W** and **V** in PMF):

`pm = 6 rows and n+1 = 3 columns.`

If it is desired to define **W** and **V** as before, then it is necessary to type exactly the same numbers as were used in defining the (6 x 3) matrix **X** above. The complete man-machine conversation is, after typing the OS: `(pmi,m)=W,V`

```
Enter dimensions <p,m> and max order <n> of polynomials
for [p*m x (n+1)] PMF <W    > :   2,3,2
Enter dimensions <p,m> and max order <n> of polynomials
for [p*m x (n+1)] PMF <V    > :   3,2,2
PMF Matrix <W    > has <    6> rows and <    3> columns
PMF Matrix <V    > has <    6> rows and <    3> columns
         W
 ,1
 ,,1
 1,,1
 2,3
 -1,1
 ,1,4
         V
 ,1
 ,,1
 1,,1
 2,3
 -1,1
 ,1,4
```

Building "SUB" and "SBR" Subroutines: In order to encourage users to build their own subroutines, consider the following two examples:

(a) Subroutine SMat.SUB

(b) Subroutine Exd.SBR

mentioned in Section 3.3.4. and 4.2.1., respectively.

The listing of these subroutines is given below:

```
no(SMat,sub)=nx,Sa,Si,Sli,Sld
(nli)=
no(poi)=n,nx,va,vi,vli,vld
va(dsm),vi(dsm)(mcp)=Sa,Si
vli(dsm),vld(dsm)(mcp)=Sli,Sld
(lis)=
f,GD(exD,sbr)=G,D
nli
nty
1,2(dzm)(tvc)=G,D
GD(ninp,t)=m
f(cdi)(dec,t)=n1
GD,n1(ctc,t)=G,D
f,n1(ctc),m(dpm,sub)=f1p
D,m(cmp),f1p(pmm),-1(s*),G,m(cmp)(pma,t)=G
D(t),m(vtm,t)=D
typ
lis
```

```
p,m(dpm,sub)=P
(nli)=
p(t),m,m(dim)(mtv)(*,t)=P
P(t),m(cmp)=P
(lis)=
```

Consulting the Help file, it may be concluded that in SMAT.SUB:

- —that for the operator POI:
  Given set of POI, or PCI, **no** = { no_i } POI generates:
  - n = the sum of { no_i }
  - nx = the max of { no_i }
  - **va**, **vi**, **vli**, **vld** = the selector vectors defined in Section 3.3.4, Eqs.(3.75)-(3.78)

- —while for the operator DSM:
  Using the previously obtained selector vectors, DSM generates the selector matrices **Sa**, **Si**, **Sli** and **Sld**, Eq. (3.79)

- —similarly, for the subroutine EXD.SBR:
  Given the $(n+1)$ dimensional row **d** containing the coefficients d_i of $d(z)$, and the $[pm \times (n+1)]$ matrix in PMF corresponding to a non-strictly proper polynomial matrix $W(z)$ EXD calculates:
  - —the strictly proper part **Wsp** in PMF and the corresponding $(p \times m)$ matrix **D**.

In doing this, another "SUB" subroutine, namely DPM.SUB (diagonal polynomial matrix) is executed within the EXD.SBR. This is, in fact, the only reason why it was necessary to define the subroutine EXD as a "SBR" type. The listing of the DPM.SUB is given above. By incorporating the code of DPM.SUB into subroutine EXD, it would be possible to define EXD as a subroutine of the "SUB" type, and in this way to speed up considerably its execution. The listing of the modified subroutine EXD, now of the "SUB" type, follows:

```
 1      f,GD(exD,sub)=G,D
 2      (nli)=
 3      GD(ninp,t)=m
 4      f(cdi)(dec,t)=n1
 5      GD,n1(ctc,t)=G,D
 6      f(t),n1(ctr),m,m(dim)(mtv)(*,t)=f1p
 7      f1p(t),m(cmp)=f1p
 8      D,m(cmp),f1p(pmm),-1(s*),G,m(cmp)(pma,t)=G
 9      D(t),m(vtm,t)=D
10      (lis)=
```

In the above subroutines the statements:

(NLI)= , NLI and NTY

are included at the beginning to transfer the *L-A-S* interpreter into **NO LIST** and **NO TYPE** modes. Similarly, the statements:

(LIS)= , TYP and LIS

are added at the end to return the interpreter to the default LIST and TYPE modes. The purpose of Statement 4 in EXD.SBR will be explained later.

To assess the faster execution of subroutines of the "SUB" type, a simple, self-explanatory program BUILDSB is given below.

```
_BuildSB
_Building_SUB_&_SBR_subroutines
_and_checking_execution_time
5,3,2(abcd,sub)=A,B,C,D
A,B,C,D(sstf)=d,W
A,B,C(sstf)=d,Wsp
(time)=t1
d,W(EXD,SBR)=Wsp1,D1
(time)=t2
d,W(exd,sub)=Wsp2,D2
(time)=t3
t3,t2(-),t2,t1(-)(mcp,t)=tsub,tsbr
Wsp,Wsp1(-),Wsp,Wsp2(-)(out)=
D,D1(-),D,D2(-)(out)=
tsub,tsbr(out,1,1)=
```

The program BUILDSB:

- —defines an arbitrary state space representation $R=\{\mathbf{A},\mathbf{B},\mathbf{C},\mathbf{D}\}$ with $n=5$, $m=3$ and $p=2$,
- —calculates arrays **d** and **W**, where $\mathbf{C}(z\mathbf{I}-\mathbf{A})^{-1}\mathbf{B}+\mathbf{D}=W(z)/d(z)$,
- —calculates arrays **d** and **Wsp** where $\mathbf{C}(z\mathbf{I}-\mathbf{A})^{-1}\mathbf{B}=Wsp(z)/d(z)$,
- —assigns current time in seconds to the scalar $t1$,
- —calls EXD.SBR, i.e. generates **Wsp1** and **D1**,
- —assigns current time to scalar $t2$,
- —calls EXD.SUB, i.e. generates **Wsp2** and **D2**,
- —assigns current time to the scalar $t3$, and
- —defines scalars $tsub = t3\text{-}t2$ and $tsbr = t2\text{-}t1$ and displays their values on the screen, etc.

In the above case *tsub* = 8 seconds, while *tsbr* = 26 seconds.

The purpose of giving this simple example is to show that names of "SUB" and "SBR" subroutine may be equal. Also, a subroutine name may be equal to the name of an existing operator statement. Another purpose is to illustrate the use of the operator TIME.

Hints for SBR Subroutine Execution: During the execution of an SBR subroutine, the interpreter commands *INCLUDE*, *ELIMINATE* or *CHANGE* should not be used, i.e. the total number of characters of the subroutine should not be altered during its execution. This is a consequence of the way the *L-A-S* Interpreter executes an SBR subroutine. In particular, when a call to an SBR subroutine is encountered in a calling program, the calling statement is replaced by all statements of the subroutine, and the execution of the subroutine begins. At that moment the total number of statements (as well as the number of characters) in the current program is increased. At completion of the subroutine execution, all subroutine statements are removed, the initial calling statement is replaced and execution is resumed. Although this process is usually transparent to the user, it may be observed when the SBR subroutine is executed in *TRACE* mode.

Function-KEY Mode: It is well known that the DOS operating system allows the user to type "in advance" several characters before they are actually processed. Similarly, in an *L-A-S* session it is possible to type in advance answers to anticipated *L-A-S* prompts " * ". In fact, this is possible only in the default NO Function-KEY mode. To explain the reasons why the Function-KEY mode has been implemented consider a simple recursive scheme intended to sum positive integers i for $1 \le i \le Imax$, i.e.

```
1.  Set 0 ⇒ Summ, 0 ⇒ i, Define Imax
2.  Set i+1 ⇒ i, Set Summ+i ⇒ Summ
3.  If i < Imax, go to 2; Else, go to 4
4.  Display Summ
```

The correct, and two incorrect *L-A-S* implementations of the above recursive scheme for *Imax* = 15, are as follows:

```
      _SummC_Correct_version
     0(coin),0(coin),15(coin)(mcp)=Summ,i,Imax
  i:i(inc)=i
     Summ,i(+,t)=Summ
     i,Imax(ifj)=i,k,k
  k:Summ(out,t,0)=
1      _SummI
2      _Incorrect_version
3      0(coin),0(coin),15(coin)(mcp)=Summ,i,Imax
4    i:i(inc)=i
5      Summ,i(+,t)=Summ
6      i,Imax(ifj)=i,i,i
7    k:Summ(out,t,0)=

1      _SummIS
2      _Incorrect_version;_with_(STO)=_statement
3      0(coin),0(coin),15(coin)(mcp)=Summ,i,Imax
4    i:i(inc)=i
5      Summ,i(+,t)=Summ
6      (sto)=
7      i,Imax(ifj)=i,i,i
8    k:Summ(out,t,0)=
```

Obviously, the version SummC is correct, since for $i < Imax$, the operator IFJ transfers control to the statement with the label "*i*", i.e. Statement 4, while for $i \geq Imax$, execution goes to the statement with the label "*k*". However, if, by accident, in Statement 6, the output field reads: *i,i,i* instead of: *i,k,k* then we have an "infinite" loop; see version SummI above. The only possibility to "exit" is to end the *L-A-S* session by "force," i.e. to type either:

```
<Ctrl>-<Break>  or
<Ctrl>-<Alt>-<Del>
```

A "conservative" (good) programming practice is to include a STO statement in all loops which may lead to infinite loops; see the version SummIS above. Note that, as was explained earlier, whenever in sequential execution, the statement STO is encountered, the execution "HALTS," and then by typing, for instance, `jump,k` (or `j,k`), it is possible to exit the loop. After that, of course, it is necessary to correct the output field in the IFJ statement. Once we are sure that loop works correctly, it is possible to eliminate the (STO)= statement.

Another way to avoid ending a *L-A-S* session by "force," once in an infinite loop, is to use the Function-KEY mode. This can be done by the interpreter command:

```
FKE  (or fk)
```

Then, during any sequential execution, by simply pressing the function key <F1> the *L-A-S* interpreter will enter into the TRACE mode, causing the sequential execution to "halt" immediately. Then, again, by typing any IC or OS the user may either exit a loop or verify what is happening in the program. However, unfortunately, when the interpreter is in the Function-KEY mode, it is not possible to type "in advance" characters as responses to anticipated future interpreter prompts, but it is necessary to wait until the prompt is actually issued. This is due to the implementation of this mode and its interaction with other parts of the *L-A-S* interpreter. Once in FKE mode, it is possible by pressing other function keys, F2 to F7, to enter/exit other *L-A-S* modes, such as LIST, TYPE, etc. More details are available in the HELP file under "Help,Fke."

Creation and Execution Modes of Operation: The above example program SUMMC, involving recursive calculation, i.e. the *L-A-S* statement IFJ (conditional jump) will be used to introduce the important concepts of the *Creation and Execution* modes of operation. The mode of operation of the *L-A-S* software corresponding to the interpreter being in the MEM mode, i.e. when all correctly typed and executed statements are stored in the interpreter working memory is referred to as the *Creation* mode of operation. The mode of operation in which the program residing in interpreter memory is reexecuted is referred to as the *Execution* mode.

Consider now that the interpreter is in the Creation mode and that the user types (one at a time) the statements of the above program SUMMC. If Statement 6 is typed as

```
*    i,Imax(ifj)=i,k,k
```

the *L-A-S* interpreter will issue a warning:

```
SELAB - Label = k does not exist
```

indicating that it expects that in the sequel, a statement with the label "*k*" will be entered. Also, note that although, for $i < Imax$, the above IFJ statement is supposed to transfer control to the statement with label "*i*", the interpreter, being currently in the Creation mode, will simply issue the prompt "*", expecting the next statetement to be typed. After typing all the statements of program SUMMC, including, of course, the statement with label "*k*", and once the user enters the IC:

```
*    jump,i     or
*    j,i        or even    j,4
```

since Statement 4 has the label "*i*", the interpreter will automatically enter into the Execution mode, i.e. sequential execution of existing statements. Then, when the IFJ statement is encountered, the control *will be transferred* either to:

—the statement with label "*i*"   for $i < Imax$ or to
—the statement with label "*k*"   for $i \geq Imax$.

This distinction between the Creation and Execution modes has been introduced solely to allow typing of IFJ and JMP operator statements, which in their output fields may refer to statement labels not currently existing in a program being typed. Note that as far as other "calculation oriented operator statements" are concerned, there is no distinction between the Creation mode and the Execution mode.

Recovery from Execution Errors: Assume that during a sequential reexecution of an *L-A-S* program an error occurs. This error might be due to an incompatibility of dimensions of arguments in the input field, or an insufficient number of arguments in the input/output fields. In this case an appropriate error message is issued and sequential execution halts immediately. The program counter corresponds to the number of the statement where the error occurred. To find out what caused the error, it is suggested to enter some interpreter commands such as:

```
Status  ,  Pro,n1,n2  ,  or  Names
```

to display the part of the program containing the statement where the error occurred, and to check the dimensions of the arrays used. Also OUT operator statements might help. Note that at that moment any operator statement may be typed, including a statement which will change some of the previously defined arrays or define new ones. Since the interpreter is in NO-MEM mode, operator statements will only perform required operations, but will not be stored in the interpreter memory. Of course, by using the IC:

```
ELI  ,  CHA   or  INC
```

it is possible to eliminate and/or change an existing statement, or to include new ones. Once the user is sure that the cause of error is eliminated, by using the IC:

```
*   j,-1       or    j,<st-#-er>
```

where `<st-#-er>` stands for the number of the statement which caused the error, sequential execution will be resumed starting from that statement. However, if the user is not completely sure that all causes of error have been removed, it is advisable first to enter in the TRACE mode and then to type `j,-1`. Then, of course, only one statement will be executed, and the user should enter:

```
*   con     (or  c)
```

after each executed statement. Once the user is convinced that the program works correctly, it is possible by using the IC NTRA to exit from TRACE mode and resume normal default sequential execution of all remaining statements.

Recovery from Errors in Subroutines: Note that the first four statements in all SBR type subroutines are as follows:

- The first statement defining the subroutine name and type, as well as input/output arguments, has the general form:

```
A1,...,An(XYZ,SBR)=B1,..,Bm
```

- The next two statements are the interpreter commands: NLI and NTY, mentioned under **Building "SUB" and "SBR" subroutines,** (see exD.sbr).

- The fourth statement is a MOS of the form:

```
1,m(dzm)(tvc)=B1,,,Bm
```

whose purpose is to define all output arguments "temporarily." It may be concluded that this MOS sets zeros in all output arguments, Bi's, i=[1,m]. It should be mentioned that in the case that during the subroutine execution there are no syntax or execution errors, this statement does not have an active role, since some other statements in the subroutine "body" again define all these output arguments. On the other hand, if an execution or syntax error occurs, then, in the case of a SBR subroutine, (which also happens in the case of a "main" program), the sequential executions "halts" immediately. Then, the user may follow a procedure explained under the subtitle **Recovery from Execution Errors**. However, if it is preferred to exit from the subroutine, the user may type the IC:

```
*  j,q      or
*  j,<st-#-first-q>
```

where `<st-#-first-q>` is the statement number of the *first* statement having the label "q". Then, since the last-but-one statement in the subroutine body usually has the label "q" (for quit), i.e.:

```
*  q:typ
```

the control will be transfered to the subroutine end. In order to exit from the subroutine "normally," all output arguments should be defined, which would not be the case if the above mentioned MOS, temporarily defining all output arguments, is not included in the begining of the subroutine body.

Since the label "q" is *reserved* for that purpose, a "good programming practice" is not to use the label "q" in main programs. If, in spite of that, the user "insists" on using the label "q" in a main program calling a SBR subroutine, then, when an error occurs within a SBR subroutine, instead of the above mentioned IC `j,q`, the user has to determine the `<st-#-first-q>`, which could be done by the IC:

```
*  f,q:     or  *  find,q:
```

In the case of an error within SUB type subroutine:

- an appropriate error message is issued,
- sequential execution halts, and
- control is transferred to the program calling this subroutine,
- the program counter indicates the statement containing the call to the SUB subroutine where the error occurred.

If it is desired to "override" the (NLI)= operator statement which is, as has been mentioned previously, usually used as the second statement in the subroutine body, the IC:

```
            *   l,sub     or   list,sub
```

may be typed before the SUB subroutine execution. If in the FKE mode, instead of the IC `l,sub` , the function key F3 may be pressed.

Changing Elements in Defined Arrays: Recall that the *L-A-S* software is well suited for reexecuting a sequence of operator statements residing in the Interpreter working memory with the same, or different, input data. Therefore, it is our feeling that at this point it is worthwhile to illustrate the possibilities of changing elements in arrays already defined. There are, of course, a number of ways that this can be accomplished, but here only two ways will be mentioned.

Assume that in a current *L-A-S* session matrix **A** with dimension (6×7) and matrix **B** with dimension (2×4) have been defined and used, and that the user has decided to modify some of their elements.

1. If single elements in A and B are to be replaced "interactively," say it is desired to substitute:

$$-2.5 \Rightarrow a_{45} \;\; ; \;\; 101.2 \Rightarrow a_{26} \;\; \text{and} \;\; 1.2*10^5 \Rightarrow b_{2,3}$$

then, the incompletely specified INP operator statement may be used, i.e.

```
                *   (inp,e)=
```

In this case the user-machine conversation is as follows:

```
Type matrix names you want to change via keyboard
or type either <N.> , <NAM.> or <ALL.> :   A,B
Enter indices (I,J) and value A(I,J) of matrix element
  I,J, A(I,J)  : 4,5,-2.5    <return>
  I,J, A(I,J)  : 2,6,101.2   <return>
  I,J, A(I,J)  :             <return>
Enter indices (I,J)  and value B(I,J) of matrix element
  I,J, B(I,J)  : 2,3,1.2e5   <return>
  I,J, B(I,J)  :             <return>

            *   A,B(out)=
```

The last statement, i.e. A,B(out)= , was used to verify if the desired substitutions were made.

2. The second way is "less" interactive, but it allows a complete block (submatrix)

in a defined matrix to be substituted into another matrix. Assume now that in the above mentioned (6 × 7) matrix **A** it is desired to replace the sub-array elements:

$$\begin{matrix} a_{23} & a_{24} & a_{25} & a_{26} \\ a_{33} & a_{34} & a_{35} & a_{36} \end{matrix}$$

with an already defined (2 × 4) matrix **B**. This can be done by typing the following operator statement:

```
*   A,B,2,3(rmp)=A
```

which places the whole matrix **B** into **A** starting at location (2,3), keeping other elements unchanged.

Using the operator RMP (Replace Matrix Part), the task of the previously mentioned incompletely specified operator statement (INP,e)= may be "non-interactively" performed by the sequence:

```
*    A,-2.5(dma),4,5(rmp)=A
*    A,101.2(dma),2,6(rmp)=A
*    1.2e5(dma),2,3(rmp)=B
```

If desired, the first two statements may be combined into the following single MOS:

```
*  A,-2.5(dma),4,5(rmp),101.2(dma),2,6(rmp)=A
```

Plotting Capabilities: To illustrate plotting capabilities of *L-A-S* software consider the *L-A-S* program given below:

```
  _Exerc
  _Plotting
  -.1,1/-1,-.1(dma,t)=A
  2,1(dim)=xo
  _Use_N=201_&_T=40
n:(dsc)=N,T
  xo,N,T,A(rcs)=x
  Time_Response(ylab)=
  Time[sec](xlab)=
  -1,1,,T(dma)=sc
  sc(yxsc)=
  x,T(dis)=
  -1,1,-2,2(dma)=sc2
  sc2(yxsc)=
  Phase_plane__x2[t](ylab)=
  x1[t]_(xlab)=
17      x(nik)=
18      N,T(out)=
19      A,xo(out,t,1)=
20      N,T(out,1,0)=
```

Its name is EXERC and it can be found in the directory C:\LAS\DPF\. The program defines a (2 × 2) matrix **A** and initial condition vector **x**(0) given by:

```
    | -0.1     1.0 |              | 1 |
A = |              |  ;   x(0) =  |   |
    | -1.0    -0.1 |              | 0 |
```

By Statement 6, with the label "n", the quantities N and T should be defined interactively. Suggested values are N=201 and T=40. The differential equation:

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{A}\,\mathbf{x}(t) \quad , \quad \mathbf{x}(0) = \mathbf{x}_0$$

for $0 \leq t \leq T$, given by the N data points is solved by Statement 6, operator RCS. Note that operator RCS is executed within the algorithm CDSR, i.e. *L-A-S* subroutine CDSR.SUB.

The two elements $x_i(t)$, $i=[1,2]$, of the state vector **x**(t) are displayed versus time by Statement 12, operator DIS, while Statement 17, operator NIK (Nyqist diagram), i.e. X-Y Plotting, displays $x_2(t)$ versus $x_1(t)$ in the "Phase-Plane" plot of Fig.C.3. Operators YLAB, XLAB and YXSC, executed before the "plotting" operators DIS and NIK, allow the user to label the y- and x-axes, as well as to set axis scales. For more details see the Help file. The plots obtained are shown in Figs.C.2 and C.3. For an illustration of three dimensional plotting capabilities the reader is referred to the *L-A-S* program PLTALL.DPF in the subdirectory C:\LAS\PLT\.

Interface with Other CAD Packages: This section pertains to using data generated by other programs or packages. In the master subdirecory C:\LAS\ there is the independent main program

INDAT.EXE

which may be used for reading data files containing arbitrary ASCII data and preparing them for inclusion in the *L-A-S* package. The use of INDAT will be

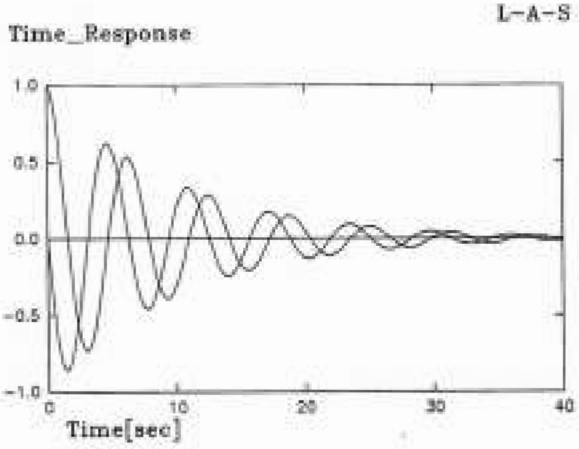


Figure C.2. *L-A-S* Time-Response Plots

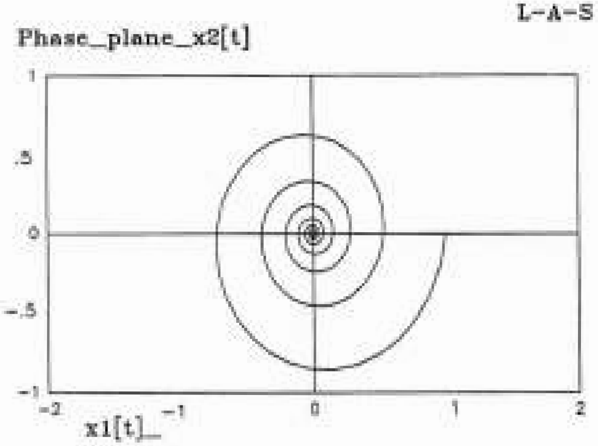


Figure C.3. *L-A-S* Phase-Plane Plot

illustrated by an example. Consider that we have a file INP.DAT of the form given below:

```
             A
-1.000  1.000   .000   .000
  .000 -2.000  1.000   .000
  .000 -1.000 -2.000  1.000
  .000   .000   .000 -2.000
```

```
          B
  1.000    .000    .000
   .001    .000    .000
   .000   1.000    .000
   .000    .000   1.000

          C
  .00000E+00   .10000E-04   .00000E+00   .10000E+01
  .10000E+01   .00000E+00   .10000E+07   .00000E+00

          D
   .000   1.000    .000
   .000    .000    .000
```

and we want to use this data in the *L-A-S* software. To this end, the following is suggested. Using any text editor, delete all blank lines and eliminate all non numeric characters. After this intervention, the file INP.DAT should have the following form:

```
 -1.000  1.000   .000   .000        Modified file INP.DAT
   .000 -2.000  1.000    .000
   .000 -1.000 -2.000  1.000
   .000    .000   .000 -2.000
  1.000    .000    .000
   .001    .000    .000
   .000   1.000    .000
   .000    .000   1.000
  .00000E+00   .10000E-04   .00000E+00   .10000E+01
  .10000E+01   .00000E+00   .10000E+07   .00000E+00
   .000   1.000    .000
   .000    .000    .000
```

Then, run the program INDAT, i.e. type: **INDAT**
The program INDAT prompts user for:

- file name containing data to be read, in our case the answer should be: INP.DAT
- file name were the modified data will be written, in our case answer might be: Temp
- the *L-A-S* variable name to which data to be read will be assigned, a possible name is: Abcd

The complete user - machine conversation is given below:

```
Please enter File name containing ASCII characters
corresponding to the array to be transferred into
the L-A-S package. To exit, Enter: STOP/stop or s

Inp.Dat
```

```
File <Inp.Dat > has <nr> =     12 non-blank rows

Please enter the file name where the modified array
to be read by the L-A-S package will be written.  You
may use any string of up to 8 characters.  Suggested
names are:  t1,  t2,  tmp1, ...

Temp

Please enter any string of up to 4 characters to be
used as the L-A-S variable name.  Suggested names
are: t1,  t2,  tmp1,  A,  ...    If you have used a
string of up to four characters for the file name
above, you may use the same string for the L-A-S
variable name

Abcd

The array <Abcd> has     12 rows and
                          4 columns

File <Temp    > created

Please enter file name containing ASCII characters
corresponding to the array to be transferred into the
L-A-S package.  To exit, enter: STOP/stop or s

s

Stop - Program terminated.
```

The file TEMP, created by the program INDAT, is given below:

```
(Inp,m)=Abcd
   12,    4
-1.000,1.000,.000,.000,
.000,-2.000,1.000,.000,
.000,-1.000,-2.000,1.000,
.000,.000,.000,-2.000,
1.000,.000,.000,
.001,.000,.000,
.000,1.000,.000,
.000,.000,1.000,
.00000E+00,.10000E-04,.00000E+00,.10000E+01,
.10000E+01,.00000E+00,.10000E+07,.00000E+00,
.000,1.000,.000,
.000,.000,.000,
Abcd(Out)=
```

After having the file TEMP created, enter *L-A-S* and as an answer to the *L-A-S* prompt "*", type:

```
                    *      file,Temp
```

According to the function of the interpreter command FILE, on the screen the following will appear:

```
Temp

*
 Read statement = (Inp,m)=Abcd

*
 Read statement = Abcd(Out)=
 -.10000E+01  .10000E+01  .00000E+00  .00000E+00
  .00000E+00 -.20000E+01  .10000E+01  .00000E+00
  .00000E+00 -.10000E+01 -.20000E+01  .10000E+01
  .00000E+00  .00000E+00  .00000E+00 -.20000E+01
  .10000E+01  .00000E+00  .00000E+00  .00000E+00
  .10000E-02  .00000E+00  .00000E+00  .00000E+00
  .00000E+00  .10000E+01  .00000E+00  .00000E+00
  .00000E+00  .00000E+00  .10000E+01  .00000E+00
  .00000E+00  .10000E-04  .00000E+00  .10000E+01
  .10000E+01  .00000E+00  .10000E+07  .00000E+00
  .00000E+00  .10000E+01  .00000E+00  .00000E+00
  .00000E+00  .00000E+00  .00000E+00  .00000E+00

*
```

Since in this way within the *L-A-S*, a (12 × 4) array **Abcd** has been created, it is now easy to extract matrices **A**,**B**,**C** and **D** from the obtained **Abcd**. This can be done, for instance, by the following sequence of *L-A-S* operators:

```
                *      Abcd,8(ctr)=AB,CD
                *      AB,4(ctr)=A,B
                *      B,3(ctc)=B
                *      CD,2(ctr)=C,D
                *      D,3(ctc)=D
```

After that, of course, arrays A,B,C and D of appropriate dimensions are available and can subsequently be used as input arguments in any *L-A-S* operator or subroutine.

Recovering from a "Crash:" It will occasionally happen that an inadvertent command will cause an *L-A-S* program to "crash" and send the user back to DOS. Fortunately, the *L-A-S* interpreter stores all data from the terminal keyboard to the file ECHO.DAT. In the case of a fatal execution error, for example a floating point overflow, the current *L-A-S* session "crashes" and the user is back at the DOS prompt. In order to recover the lost session, without retyping all the statements again, the following should be done:

- Rename the file ECHO.DAT to an arbitrary name, e.g. ECHO.DDD
- Edit this file and delete (or comment out) the last *L-A-S* statement which caused the "crash." Also, all other statements which caused the execution and/or syntax error should be deleted (or commented out).
- Save that modified file.
- Enter the *L-A-S* interpreter, and following the first interpreter prompt: " * ", enter the following Interpreter command:

```
*  file, ECHO.DDD
```

- In the case that in the edited file ECHO.DDD there are no *L-A-S* statements which cause an execution and/or syntax error, all statements from that file will be sequentially executed. After the execution of the last statement, the *L-A-S* Interpreter will issue the standard prompt " * ", and will be ready to accept any statement from the terminal keyboard. All executed statements will again be stored on the newly created ECHO.DAT file.

In order to gain experience in using the ECHO feature, three examples of ECHO files are presented. The first one is the non-edited version obtained after the program due to the last statement:

```
* ff,f(*,t)=ff
```

has returned to DOS. Note that in this example the following statements:

```
1,2,4,3(dam,t)=verr
ss
v,v(*,e)=vv
```

are incorrect. Therefore, these three statements, together with the last one should be either deleted, or commented out, as has been done in the second version, which was renamed to ECHO.DDD. The third version of the ECHO.DAT file was created during the reading of the edited and renamed ECHO file.

```
_    L - A - S   ECHO.DAT   file                   Version 1

_    created   7/31/1992   at   11:20    ECHO.DAT file created
 _Echo_Example                           by L-A-S Interpreter
 I,3,2(dma)=v                            when overflow occurred
 1,2,4,3(dam,t)=verr                    Operator DAM does not exist
 __Syntax error above "
 ss                                      I.C. SS does not exist
 __I.C. Syntax Error above ^ !
 s
 v(out,t,2)=
 v,v(*,e)=vv                              v is (1 x 3) row, multipli-
 __Syntax error above ^                    cation v*v is not permitted
 v,v(t)(*,t)=vv
 1e100(dma,t)=f                            -here f = 10^100
 f,f(*,t)=ff
 ff,f(*,t)=ff                              -here ff = 10^200
 ff,f(*,t)=ff                              This statement has created
                                           "overflow" and consequently
                                           this L-A-S session is ended
                                           "in a crash."

_     L - A - S   ECHO.DAT   file                    Version 2

_     created   7/31/1992   at   11:20
 _Echo_Example
 I,3,2(dma)=v                             File ECHO.DAT modified and
 _Comm_1,2,4,3(dam,t)=verr                 renamed  ECHO.DDD
 __Syntax error above ^
 _Comm_ss
 __I.C. Syntax Error above ^ !             Lines creating syntax or
 s                                         execution errors are com-
 v(out,t,2)=                               mented out by adding a
 _Comm_v,v(*,e)=vv                         leading "_Comm_".
 __Syntax error above ^
 v,v(t)(*,t)=vv
 1e100(dma,t)=f
 f,f(*,t)=ff
 ff,f(*,t)=ff
 _Comm_ff,f(*,t)=ff

                                                          Version 3
_     L - A - S   ECHO.DAT   file              New ECHO.DAT file created
                                           during the L-A-S session
_     created   7/31/1992   at   11:30     in which the I.C FILE was
 file,echo.ddd                             used and the file ECHO.DDD
                                           was read.

_     L - A - S   ECHO.DAT   file          Note that this ECHO file was
                                           created 10 minutes after the
_     created   7/31/1992   at   11:20     first version.
 _Echo_Example
 I,3,2(dma)=v
 _Comm_1,2,4,3(dam,t)=verr
 __Syntax error above ^
 _Comm_ss
 __I.C. Syntax Error above ^ !
 s
 v(out,t,2)=
 _Comm_v,v(*,e)=vv
 __Syntax error above ^
 v,v(t)(*,t)=vv
 1e100(dma,t)=f
 f,f(*,t)=ff
 ff,f(*,t)=ff
 _Comm_ff,f(*,t)=ff
 _Now_L-A-S_interpreter_is_back
 _in_terminal_Keyboard_mode
 _Some_additional_statements_are_typed
```

```
ff(out)=
p
v(out,t,0)=
_Now,_to_quit_"q"_should_be_typed
q
```

C.4 A List of *L-A-S* Interpreter Commands

MNEMONIC NAME	DESCRIPTION
!<*Sys. Com.*>	Execution of any DOS operating system command
BEG (B)	Begin; the current *L-A-S* program and variables are deleted
BEL	Activates the computer bell
CHA	Change any string of characters in an operator statement
CLE	Clear the terminal screen
CON (C)	Continue
COP (CO)	Copy part of the *L-A-S* program
DSP	Display status of the *L-A-S* interpreter
ELI (E)	Eliminate operator statements
ELM	Eliminate matrices
END	End; ends the *L-A-S* session
EPS	Definition of the default "machine zero"
FIL	All inputs to the *L-A-S* interpreter are from a specified file
FIN (F)	Find string of characters in *L-A-S* program
FKE (FK)	Enter Function-KEY mode
GRA	CGA high resolution graphics mode
HELP (H)	Syntactical description of *L-A-S* statements
INC	Include an operator statement or interpreter command
INF	Include a program file into the current *L-A-S* program
JUM (J)	Jump; jump to any statement in the current *L-A-S* program
LIS (L)	List operator statements - enter List mode
LOA (LO)	Loads arrays into the *L-A-S* Interpreter memory
MEM (M)	Memorize entered operator statements - enter Memorize mode
MOV (MO)	Move part of the *L-A-S* program
NAM (N)	Names; display of names and dimensions of all arrays
NBE	Deactivates the computer bell
NFK (NF)	Exit Function-KEY mode (default)
NLI (NL)	No listing of operator statements - exit List mode
NME	No memorizing of operator statements - exit Mem. mode
NTE	No test; exit test mode
NTR	No tracing of *L-A-S* program execution - exit Trace mode
NTY (NT)	No typing of operator statement results - exit Type mode
OPE (O)	Display of compatibility conditions for dimension of input arrays
PFI (PF)	Print file specification

PRL (PR)	Print *L-A-S* program listing
PRO (P)	Display *L-A-S* program on terminal screen
QUI (Q)	Quit; ends *L-A-S* session
REF	Read external file; read *L-A-S* program created by any text editor
RPF (R)	Read program file; *L-A-S* program from the DPF (Disk Program File) is read and added to the current *L-A-S* program
RSV	Restore all variables
SAV	All variables are stored (saved) on a file.
STA (S)	Status; displays status of *L-A-S* program
STV	Store all variables
TCH	Total change; global substitute of an old name by a new one in an *L-A-S* program
TES	Test; display of additional intermediate information during execution, *(used only in L-A-S software implementation)*
TIM	Time in [sec] for plots to stay on the screen
TRA	Trace; trace *L-A-S* program execution - enter Trace mode
TYP (T)	Type; cancels NTY - enter Type mode
WPF (W)	Write program file; the current *L-A-S* program is saved
XLAB	Label the X-axis of the plot
YLAB	Label the Y-axis of the plot

Detailed syntactical description of each interpreter command may be obtained by typing **HELP,xyz** ; where **xyz** stands for the mnemonic name of an interpreter command.

C.5 On-Line Help File

This section reviews "help" descriptions of some *L-A-S* operators. This type of information is available in the on-line Help file. Initially, three examples which indicate the type of help that is available are presented. Following these examples, a selection of actual Help file responses is given. Although not exhaustive, the few examples of this section should convey that the *L-A-S* Help file provides adequate information to make good use of the corresponding operators.

Example 1:

PURPOSE: Matrix inversion with optional determinant calculation.

USAGE: **A (-1 [,*fl*])= AI [, D].** The *fl* (flag) option is used to specify the output format: *t* corresponds to *xxx.xxx* and *e*, to an *exponential form* for a wider range of values.

DESCRIPTION: **AI** is the inverse of the $(n \times n)$ matrix **A**. **D** is the determinant of the matrix **A**.

See also: *INV, P-1*

EXAMPLE:

Given the matrix (previously defined in *L-A-S*):

$$\mathbf{A} = \begin{bmatrix} -2 & 1 & 0 & 0 \\ 0 & -1 & 1 & 0 \\ 0 & -1 & -1 & 1 \\ 0 & 0 & 0 & -3 \end{bmatrix}$$

The *L-A-S* statement: **A (-1, t) = AI, D** will yield

$$\mathbf{AI} = \begin{bmatrix} -0.5 & -0.25 & -0.25 & -0.083 \\ 0 & -0.5 & -0.5 & -0.167 \\ 0 & 0.5 & -0.5 & -0.167 \\ 0 & 0 & 0 & -0.333 \end{bmatrix}$$

with the determinant D = 12.000

Example 2:

PURPOSE: Complex function multiplication

USAGE: **X, Y (C* [,*fl*]) = Z**

DESCRIPTION: The matrix **X** of dimension $(n \times 2)$ has the complex form $x_k = real(x_k) + j\ imag(x_k)$, where $k = 1, \cdots, n$ and $j = (-1)^{1/2}$. The matrices **Y** and **Z** have the same form. The operation represents a term by term complex multiplication.

See also: *, C/, F*, P*, PMM, S*

EXAMPLE: Given the arrays (previously defined in *L-A-S*):

$$X = \begin{bmatrix} 1 & 2 \\ 2 & 1 \\ 1 & 0 \\ 2 & 0 \\ 0 & -2 \end{bmatrix} \quad , \quad Y = \begin{bmatrix} 1 & -2 \\ 2 & 1 \\ 0 & 2 \\ 1 & 0 \\ 0 & -2 \end{bmatrix}$$

The *L-A-S* statement **X, Y (C*) = Z** will yield:

$$Z = \begin{bmatrix} 5 & 0 \\ 3 & 4 \\ 0 & 2 \\ 2 & 0 \\ -4 & 0 \end{bmatrix}$$

Example 3:

PURPOSE: To calculate the eigenvalues of a square matrix.

USAGE: **A (EGV [, *fl*]) = EG**

DESCRIPTION: Given the $(n \times n)$ matrix **A**, the two columns of the $(n \times 2)$ matrix **EG** contain the real and imaginary parts of the eigenvalues of the matrix **A**.

See also: EGC, CHE, CHD

EXAMPLE: Given the matrix (previously defined in *L-A-S*):

$$A = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -2 & -4 & -3 \end{bmatrix}$$

The *L-A-S* statement **A (EGV) = EG** will yield:

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

The following is a list of several examples taken directly from the *L-A-S* Help file:

```
*   T,Ad[,Egm,N,Eps](Lnm[,f1])=Ac   Natural log of (n x n) matrix Ad
              Ln(Ad)/T ==> Ac
T(1 x 1) scalar or integer
Ad(n x n) given matrix
Egm(1 x 1) scaling factor < 1, default value = 0.25, leads to:
    |  w[ I-Ad^(1/r) ] | < Egm; r = 2^j ; j & r scaling coefficients
N(1 x 1) truncation coefficient, default values N=36.
Eps(1 x 1) << 1, sufficiently small positive scalar
   Preset default value: Eps = 10^(-16)
   Default value of Eps could be changed by the I.C. EPS
                           Algorithm, see Chapter 2:
       Given T,Ad,Egm ==> j,r; then
       Ac = -r/T* sum ( [ I - Ad^(1/r) ]^i * (1/i ) ; i=[1,N]

See also: EATF


*   c,A(POLR[,f1)=r    Polynomial reduction using the C-H-Theorem
c(1 x N); A(n x n); r(1 x n); N > n
r(s) and c(s) satisfy:   r(A) = c(A)
      Algorithm, see Chapter 2:
                 1.  c ==> r, N ==> k; det(Is-A) ==> f(s)
                 2.  k-1 ==> k
                 3.  r(k-n+j)-r(k)*f(j) ==> r(k-n+j), for j=[o,n-1]
                 4.  For k > n go to 2; Else, Stop.

See also: POM


*   A,B[,k](QC[,f1])=Qc   Controllability matrix
A(n x n); B(n x m)
k(1 x 1) specifies # of blocks in Qc; default value: k = n-m+1
           If k = 0, Operator uses k = n-m+1
Qc(n x h) = Controllability matrix = |B|A*B|..|A^(k-1)*B|, h=m*k

See also: QO, RKC, RKR, NRS


*   xo,N,Tt,A[,B,u](RCS[,f1])=x[,T] ; Continuous system response
  xo(n x 1) = intial condition vector
   N(1 x 1) = # of points, scalar or integer
  Tt(1 x 1) = total time, scalar or integer
  A (n x n) = system matrix
  B (n x m) = input matrix
  u (N x m) = input vector
  x (N x n) = state vector - solution
  The operator calculates the solution of the following diff. eq.

  dx(t)/dt = A * x(t) + B * u(t)   ;  x(o)=xo
  for:      t = [0,Tt]  in  N points.

  T(N x 1) = values of independent variable t - obtained
  from values N and Tt ; initial value = 0.

  If u omitted, calculates step response; assumes: u = step
  If both B and u omitted, calculates response to xo; assumes: B = 0

See also: CE1, CE2, CE3, CE4, CE5
*   v(DSM[,fl])=S ; Define Selector Matrix S - Oper. stmt.

v(1 x n);  S(n x m); where  m = # of non-zero elements in selector
vector v.
v = (vi);  S = (si))
                                            0 0 0      Non-zero elements
                                            1 0 0       in the row v are
Example:  for v = [ 0 1 3 0 2 ] ; S = 0 3 0      typically unity.
                                            0 0 0
                                            0 0 2

*   A[,Eps](NRS[,fl])=N[,R,r] - Null- , Range-space and Rank
A(n x m) given matrix
Eps = sufficiently small postive scalar. Preset default value:
      Eps = 10^(-16)
      Default value of Eps could be changed by the I.C. EPS
N(m x v)  Null space of A, satisfies: A * N = 0, v = m - r = Nullity
of A
R(n x r) Range space of A, satisfies: r = rank[A] = rank[A|R]
r(1 x 1) Rank of A,  satisfies: m = v + r

See also: RKC, RKR, QC, QO


*   Q[,eps](RKC[,fl])=[Qli][,Qld][,sv]  RanK of Q and separation of
linear independent Columns. Q[n x m], Qli[n x r] contains L.I.
columns,
Qld[n x (m-r)] contains L.D. columns. ;
sv[1 x m] ;  selector vector; elements are equal either to one or zero
The i-th element of sv equal to one signifies that the
i-th column of Q is linearly independent on previous
j columns of Q  ;  j = [1,i-1]. and that column qi is in Qli
eps = suff. small positive scalar. Preset default value: eps= 10^(-16)
The default value could be changed using the I.C. EPS

See also: RKR


*   A(NRR[,fl])=an[,Av]  NoRm & Row norms; Frobenious norm
A(n x m) = (aij) ; an = sqrt( sum of aij^2 ) ; Avi = sqrt( sum of
aij^2 )
an[1 x 1] = norm of A ; Av[1 x n] =  norms of rows of A

See also: NRC
```

It is our hope that with this brief introduction the reader will be able to strike out on his or her own, making ample use of the on-line help and the lists in the next section.

Authors' Remarks:

Although *L-A-S* might seem "complicated" to a new user, it does, in fact, follow the book's motto:

"Everything should be made as simple as possible, but not simpler."

L-A-S has more features than most existing CAD packages, some not found in any other CAD software. Some paticularly useful features are the "modes:"

MEM , TRACE , LIST , TYPE , Function-KEY

as well as the flexible multiple-operator statements (MOS) and subroutines.

To get full benefit from the software, it is necessary to invest time to get acquainted with all *L-A-S* features. On the other hand, the features required to use the software as a simple matrix "calculator," which is what the majority of other existing CAD packages basically offer, see examples ExampC1, Section C.1, and ExerC, Section C.3, may be learned in virtually no time.

C.6 *L-A-S* Code for Specific Algorithms

CHAPTER 1

LIN.SBR

```
   par,zo,dz(Lin,sbr)=A,B,diff
   nli
   nty
   _(A,B)_linearized_model_of
   _dx(t)/dt=g(x,u,p); z=|x|u|
   _in_the_vicinity_of_zo
   I,J(dzm)(tvc)=A,B,diff
   _Message_LIN_displayed
   _on_screen
   lin(tfi)=
   (sto)=
   zo(rdi)=h
   h,1(dzm)=zerv
   zerv(inc)=onv
    onv(t),-1(s*),h,h(dim)
    (rti,t)=T
   par,zo(gz,sub)=go
   go,go(rdi)(mcp)=H,n
   0(coin)=i
i:i(inc)=i
   dz,i,1(exm)=dzi
    zo,zo,i,1,1,1(exm),dzi
    (+),i,1(rmp,t)=zi
   par,zi(gz,sub)=gi
   H,gi(cti,t)=H
   i,n(ifj)=i,j,j
j:(nop)=
   dz(t)(ddm)=D
   H,T(*,t)=H
   H,D(-1)(*,t)=H
   H,n(ctc)=A,B
   go,H,dz(*)(+,t)=g1a
   par,zo,dz(+)(gz,sub)=g1
   g1,g1a(-,e)=diff
q:typ
   lis
```

GZ.SUB

```
   par,z(gz,sub)=g
   (nli)=
    _Generates_nonlinear
   _function_g(z,p)
   _z=|x|u|_defining
   _dx(t)/dt=g(x,u,p)
    _for_2_Deg_of_Freedom
    _Robot_Arm
   _Called_by_LIN.SBR
   par(tvc,t)=a,b,c,d,e
   z(t)(tvc,t)=
   z1,z2,z3,z4,z5,z6
   z4,2(s*)(sin)=s2z4
   z2,2(s*)(sin)=s2z2
   z4(cos)=cz4
   z4(sin)=sz4
    b,c,cz4(*),cz4(*)(+),d,
   sz4(*),sz4(*)(+,t)=den
    a,z1(*),z3(*),s2z2(*),
   z5(+),z6(-,t)=g1
   g1,den(s/,t)=g1
    z1,z1(*),s2z4(*),e(*),
   -1(s*,t)=g3
   g3,z6,e(s/)(+,t)=g3
   g1,z1,g3,z3(rti,t)=g
   (lis)=
```

QC.SUB

```
   A,B(qc,sub)=Qc
   (nli)=
    _Implemented_also_as
    _QC_operator
    _Qc=_Controllability
    _matrix_of_(A,B)
    _Qc_is_(n_x_h); h=(n-m+1)m
   X(cdi),B(cdi)(mcp)=n,m
   n,m(-)(inc)=im
   B(mcp)=X
```

```
  n,0(dzm)=Qc
  0(coin)=i
i:i(inc)=i
  Qc,X(cti,t)=Qc
  A,X(*)=X
  i,im(ifj)=i,j,j
j:(lis)=
```

QO.SUB

```
  A,C(qo,sub)=Qo
  (nli)=
  _Implemented_also_as
  _QO_operator
  _Qo=_Observability
  _matrix_of_(A,C)
  _Qo_is_(h_x_n);_h=(n-p+1)p
  A(cdi),C(rdi)(mcp)=n,k
  n,k(-)(inc)=im
  C(mcp)=X
  0,n(dzm)=Qo
  n,n(-)=i
i:i(inc)=i
  Qo,X(rti,t)=Qo
  X,A(*)=X
  i,im(ifj)=i,j,j
j:(lis)=
```

NRS.SUB

```
  A,eps(nrs,sub)=N,R,r
  (nli)=
  _Implemented_also_as
  _NRS_operator
 _N,_R_=_Null_and_Range_space
  _of_A;_r=rank(A)
  A(svd)=w,U,V
  w,w,eps(f/)=x
  x,x(t)(*,t)=r
  U,r(ctc)=R
  V,r(ctc)=x,N
  (lis)=
```

CDSR.SUB

```
  A,B,C,D,xo,u,T(cdsr,sub)=y
  (nli)=
  _General_SS_(C-T)_or
  _(D-T)==>
  _Response_to_u_&_xo
  u(t)=ut
  _For_D-T_case_use_T=
  _neg._integer
  T(ifj)=d,d,c
d:(nop)=
  _D-T_system_response
  A,B,C,ut,xo(rds)=yo
  yo,ut,D(t)(*)(+)=y
  (jmp)=f
c:(nop)=
  _C-T_system_response
  ut(rdi)=N
  xo,N,T,A,B,ut(rcs)=x
  x,C(t)(*),ut,D(t)(*)(+)=y
f:y(t)=y
  (lis)=
```

RESO.SUB

```
  A(Reso,sub)=p,Rr,R
  (nli)=
 _Implemented_by_SSTF_and_ALT
  _operators._Use:
  _A,I,I(SSTF)=p,R
  _R(alt)=Rc,Rr
  _I=Identity_matrix
  A(cdi)=n
  n,n(dim)=I
  n,0(dzm)=Rr
  n,n(dim)=Ri
  n,n(*),0(dzm)=R
  1(coin)=p
  0(coin)=i
i:i(inc)=i
  Ri,A(*)=E
  E(tr),i(s/),-1(s*)=pi
  pi,p(cti,t)=p
  Ri,Rr(cti,t)=Rr
  Ri(mtv)(t),R(cti,t)=R
  E,I,pi(s*)(+)=Ri
  i,n(ifj)=i,j,j
j:(nop)=
  R,n(cmp)=R
  (lis)=
```

LALG.SUB

```
  A,B,C,D(Lalg,sub)=p,Rr,R
  (nli)=
  _Implemented_by_SSTF_and
  _ALT_operators
  _Use_A,B,C,D(sstf)=p,R
  _R(alt)=Rc,Rr
  A(cdi)=n
  n,n(dim)=I
  D,I(mcp)=Rr,Ri
  D(mtv)(t)=R
  1(coin)=p
  0(coin)=i
i:i(inc)=i
  Ri,A(*)=E
  E(tr),i(s/),-1(s*)=pi
  C,Ri,B(*)(*),D,pi(s*)(+)=Ri
  pi,p(cti,t)=p
  Ri,Rr(cti,t)=Rr
  Ri(mtv)(t),R(cti,t)=R
  E,I,pi(s*)(+)=Ri
  i,n(ifj)=i,j,j
j:(nop)=
  R,B(cdi)(cmp)=R
  (lis)=
```

SSTF.SUB

```
   A,B,C,D(sstf,sub)=P,W
   (nli)=
   _Implemented_by_the
   _SSTF_operator
   _Calculates_Transfer
   _function_matrix
   _W(s)/P(s)=C(Is-A)(^-1)B+D
   _W_is_in_the_PMF
   _A(cdi),B(cdi),C(rdi)
   (mcp)=n,m,p
   n,1(dzm)=zv
   p,m(*),1(dzm)=zpm
   n,n(dim)=I
   0(coin)=z
   z(mcp)=j
   n,0(dzm)=WC
   0,n(dzm)=BWC
k:j(inc)=j
   C,j,1,1,n(exm)=ci
   0,n(dzm)=Wc
   z(mcp)=i
i:i(inc)=i
   A,-1(s*),ci,i,1(rmp)=Aci
   Aci(mtv)=Aciv
   I,zv(t),i,1(rmp)=Iei
   Iei(mtv)=Ieiv
    Aciv,Ieiv(rti)(t),n(cmp)
   =Wcpl
   Wcpl(p-1)=Adj,det
   _det(out)=
   _Wc,det,n(ctc)(rti,t)=Wc
   i,n(ifj)=i,j,j
j:(nop)=
   WC,Wc(cti,t)=WC
   j,p(ifj)=k,l,l
l:(nop)=
   z(mcp)=l
L:l(inc)=l
   B,l,1,n,1(exm)(t,t)=blt
   blt,WC(*,t)=btWC
   btWC,p(vtm)(t)=blWC
   BWC,blWC(rti,t)=BWC
   l,m(ifj)=L,K,K
K:(nop)=
   D(mtv)=Dv
   A(mtf)=P
    BWC,zpm(cti),Dv(t),P(*)
   (+,t)=W
   W,m(cmp)=W
   (lis)=
```

CHAPTER 2

CTDT.SBR

```
   A,B,C,D,T,eps,Isrb
   (CTDT,sbr)=A1,B1,C1,D1
   nli
   nty
   _General_(CT)_<==>_(DT)
   _for_(SI)_(RI)_and_(Bl_Tr.)
   I,4(dzm)(tvc)=A1,B1,C1,D1
   Isrb(ifj)=c,q,d
d:(nop)=
   _C-T==>D-T
   _Isrb,2(ifj)=s,r,b
s:(nop)=
   _(SI)
   _A,B,C,D,T,eps(SRcd,SBR)=
   A1,B1,x,C1,D1,y
   nli
   nty
   (jmp)=q
r:(nop)=
   _(RI)
   _A,B,C,D,T,eps(SRcd,SBR)
   =A1,x,B1,C1,y,D1
   nli
   nty
   (jmp)=q
b:(nop)=
   _(Bl.Tr.)
   _A,B,C,D,T,eps(BLcd,SBR)
   =A1,B1,C1,D1
   nli
   nty
   (jmp)=q
c:(nop)=
   _D-T==>C-T
   _Isrb,-2(ifj)=B,R,S
S:(nop)=
   _(SI)
   _A,B,C,D,T,eps(SRdc,SBR)
   =A1,B1,x,C1,D1,y
   nli
   nty
   (jmp)=q
R:(nop)=
   _(RI)
   _A,B,C,D,T,eps(SRdc,SBR)
   =A1,x,B1,C1,y,D1
   nli
   nty
   (jmp)=q
B:(nop)=
   _(Bl.Tr.)
   _A,B,C,D,T,eps(BLdc,SBR)
   =A1,B1,C1,D1
   nli
   nty
q:typ
   lis
```

SRCD.SBR

```
   Ac,Bc,Cc,Dc,T,eps(SRcd,sbr)
   =Ad,Bds,Bdr,Cd,Dds,Ddr
   nli
   nty
   _(CT)==>(DT)_Transformation
   _into_(SI)_and_(RI)
   _(Ad,Bds,Cd,Dds)=(SI)
```

```
    _(DT)_model
 6  _(Ad,Bdr,Cd,Ddr)=(RI)
    _(DT)_model
 7  I,6(dzm)(tvc)=
    Ad,Bds,Bdr,Cd,Dds,Ddr
 8  T,Ac(eatf,t)=Ad,E,F
 9  F,E,F(-)(-1)(*,t)=P
10  E,Bc(*),T(s*)=Bds
11  Bds,F,Bc(*),T(s*)(-,t)=Bdo
12  Ad,Bdo,P,Cc,Dc,eps
    (R5R4,sub)=Bdr,Ddr
13  Cc,Dc(mcp)=Cd,Dds
14 q:typ
15  lis
```

SRDC.SBR

```
 1  Ad,Bd,Cd,Dd,T,eps(SRdc,sbr)
    =Ac,Bcs,Bcr,Cc,Dcs,Dcr
 2  nli
 3  nty
 4  _(DT)==>(CT)_Transformation
    _into_(SI)_and_(RI)
 5  _(Ac,Bcs,Cc,Dcs)=(SI)
    _(DT)_model
 6  _(Ac,Bcr,Cc,Dcr)=(RI)
    _(DT)_model
 7  I,6(dzm)(tvc)=
    Ac,Bcs,Bcr,Cc,Dcs,Dcr
 8  T,Ad(lnm,t)=Ac
 9  T,Ac(eatf,t)=x,E,F
10  F,E,F(-)(-1)(*,t)=P
11  Ad,Bd,P,Cd,Dd,eps
    (R4R5,sub)=Bdo,Dcr
12   E,F(-)(-1),Bdo(*),T(s/,t)
    =Bcr
13  E(-1),Bd(*),T(s/,t)=Bcs
14  Cd,Dd(mcp)=Cc,Dcs
15 q:typ
16  lis
```

BLCD.SBR

```
 1  Ac,Bc,Cc,Dc,T,eps(BLcd,sbr)
    =Ad,Bd,Cd,Dd
 2  nli
 3  nty
 4  _(CT)==>(DT)_Bilinear
    _Transform_into_4_matrix_
 5  _State_space_models
 6  I,4(dzm)(tvc)=Ad,Bd,Cd,Dd
 7  Ac,Bc,T,1(Bcdc,sub)=
    Ad,Bo,B1,P
 8  Ad,Bo,P,Cc,Dc,eps
    (R5R4,sub)=Bd,Dd
 9  Cc(mcp)=Cd
10 q:typ
11  lis
```

BLDC.SBR

```
 1  Ad,Bd,Cd,Dd,T,eps
    (BLdc,sbr)=Ac,Bc,Cc,Dc
 2  nli
 3  nty
 4  _(DT)==>(CT)_Bilinear
    _Transformation:
 5  _4_term_State_Space_models
 6  I,4(dzm)(tvc)=Ac,Bc,Cc,Dc
 7  Ad,Bd,T,2(Bcdc,sub)=
    Ac,Bo,B1,P
 8  Ac,Bo,P,Cd,Dd,eps
    (R5R4,sub)=Bc,Dc
 9  Cd(mcp)=Cc
10 q:typ
11  lis
```

EAT.SBR

```
 1  T,Ac,Nrm,N(Eat,sbr)=Ad
 2  nli
 3  nty
 4  _Implemented_also_by_EATF
    _operator
 5  _Ad=exp(Ac*T)
 6  _Nrm_satisfies
    _||AcT||/Nrm_<_r;_r=2^(j)
 7  0(coin)=Ad
 8  Ac(nrr),T(*),Nrm(s/)(log),2
    (log)(s/)(int)(inc,t)=j
 9  j,2(log)(*)(exp,t)=r
10  T,r(s/,t)=T1
11  j,T1(out)=
12  N(fact,sub)=f
13  0(coin),1(coin),N(inc)
    =(cti)=vc
14  f,vc(gta)(t),T1(log)(s*)
    (exp)(f*)=C
15  C,Ac(polr)=Cr
16  Cr,Ac(pom)=Ad
17  j(ifj)=q,q,C
18 C:(nop)=
19  0(coin)=i
20 e:i(inc)=i
21  Ad,Ad(*)=Ad
22  i,j(ifj)=e,q,q
23 q:(nop)=
24  typ
25  lis
```

EATJ.SUB

```
 1  T,Ac,eps(Eatj,sub)=Ad
 2  (nli)=
 3  _Implemented_by_EATF
    _operator
 4  _Ad=exp(Ac*T)_only_for
```

```
  _Diagonalizable_Ac
 Ac(jfr,t)=Mc,Ac
 Ac(egv),T(s*)=egcT
 egcT,2,eps(efjf)=ExJf
 Mc,ExJf,Mc(-1)(*)(*,t)=Ad
 (lis)=
```

SICD.SBR

```
T,Ac,B,Nrm,N(SIcd,sbr)=Ad,Bd
  nli
  nty
   _Step_Invariant
   _Discretization
   _exp(Ac,T)==>Ad;
   _{Ad,Bd}_DT_pair
  I,2(dzm)(tvc)=Ad,Bd
  Ac(cdi)=n
  n,n(dim)=I
  Ac(nrr),T(*),Nrm(s/)(log),2
   (log)(s/)(int)(inc,t)=j
  j,2(log)(*)(exp,t)=r
  T,r(s/,t)=T1
  j,T1(out)=
  N(fact,sub)=f
   0(coin),1(coin),N(inc)
  (cti)=vc
   f,vc(gts)(t),T1(log)
  (s*)(exp)(f*)=C
  C,T1(s/)=C
  C(shl),Ac(polr)=Cr
  Cr,Ac(pom)=E
  j(ifj)=F,F,C
C:(nop)=
  0(coin)=i
e:i(inc)=i
   Ac,E(*),T1(s*),2(s/),I
   (+),E(*)=E
  T1,2(s*)=T1
  i,j(ifj)=e,F,F
F:(nop)=
  Ac,E(*),T1(s*),I(+)=Ad
  E,B(*),T1(s*)=Bd
q:typ
  lis
```

RICD.SBR

```
   T,Ac,B,Nrm,N(RIcd,sbr)=
  Ad,Bdo,Bd1
  nli
  nty
   _Ramp_Invariant
   _Discretization
   _exp(Ac*T)==>Ad;
   _{Ad,Bdo,Bd1}_DT
 _triple_in_five_matrix_model
  1,3(dzm)(tvc)=Ad,Bdo,Bd1
  Ac(cdi)=n
  n,n(dim)=I
  Ac(nrr),T(*),Nrm(s/)(log),2
   (log)(s/)(int)(inc,t)=j
  j,2(log)(*)(exp,t)=r
  T,r(s/,t)=T1
  j,T1(out)=
  N(fact,sub)=f
   0(coin),1(coin),N(inc)
  (cti)=vc
   f,vc(gts)(t),T1(log)(s*)
  (exp)(f*)=C
  C,T1(s/)=C
   C(shl)(shl),T1(s/),Ac
  (polr)=Crf
  Crf,Ac(pom)=F
  j(ifj)=F,F,C
C:(nop)=
  0(coin)=i
e:i(inc)=i
  Ac,F(*),T1(s*),I(+)=AFI
  AFI,2(s/)=AFI
  AFI,AFI(*),F,2(s/)(+)=F
  T1,2(s*)=T1
  i,j(ifj)=e,F,F
F:(nop)=
  Ac,F(*),T1(s*),I(+)=E
  Ac,E(*),T1(s*),I(+)=Ad
  E,B(*),T1(s*)=Bd
  F,B(*),T1(s*)=Bd1
  Bd,Bd1(-)=Bdo
q:typ
  lis
```

EATF.SBR

```
  T,Ac,Nrm,N(EATF,sbr)=Ad,E,F
  nli
  nty
   _Implemented_also_by_EATF
   _operator
 _Discretization_Ad=exp(Ac*T)
  _E_and_F_satisfy:
   _Ac*F*T=E_&_Ac*E*T=Ad
  I,3(dzm)(tvc)=Ad,E,F
  Ac(cdi)=n
  n,n(dim)=I
  Ac(nrr),T(*),Nrm(s/)(log),2
  (log)(s/)(int)(inc,t)=j
  j,2(log)(*)(exp,t)=r
  T,r(s/,t)=T1
  j,T1(out)=
  N(fact,sub)=f
   0(coin),1(coin),N(inc)
  (cti)=vc
   f,vc(gts)(t),T1(log)(s*)
  (exp)(f*)=C
  C,T1(s/)=C
   C(shl)(shl),T1(s/),Ac
  (polr)=Crf
  Crf,Ac(pom)=F
  j(ifj)=F,F,C
C:(nop)=
  0(coin)=i
e:i(inc)=i
  Ac,F(*),T1(s*),I(+)=AFI
```

```
26    AFI,2(s/)=AFI
27    AFI,AFI(*),F,2(s/)(*)=F
28    T1,2(s*)=T1
29    i,j(ifj)=e,F,F
30 F:(nop)=
31    Ac,F(*),T1(s*),I(+)=E
32    Ac,E(*),T1(s*),I(+)=Ad
33 q:typ
34    lis
```

LNM.SBR

```
 1    T,Ad,Egm,N,eps(Lnm,sbr)=Ac
 2    nli
 3    nty
 4    _Implemented_also_by_LNM
      _operator
 5    _Ac=Ln(Ad)/T
 6    _Egm_satisfies:
      _eg[I-Ad^(1/r)]_<_Egm
 7    O(coin)=Ac
 8    Ad(cdi)=n
 9    n,n(dim)=I
10    T,Ad(mcp)=Tr,Aj
11 j:I,Aj(-)=L
12    L(egv)(rpt),1,1,1,1
      (exm)=emax
13    emax,Egm(ifj)=z,z,p
14 p:Tr,2(s/)=Tr
15    Aj,eps(sqm,t)=Aj
16    (jmp)=j
17 z:(nop)=
18    T,Tr(s/)=r
19    r(out)=
20    N(fln,sub)=f
21    f,L(polr)=vr
22    vr,L(pom)=Ac
23    Ac,Tr(s/),-1(s*,t)=Ac
24 q:typ
25    lis
```

LNMJ.SUB

```
 1    T,Ad,eps(Lnmj,sub)=Ac
 2    (nli)=
 3   _Implemented_by_LNM_operator
 4    _Ac=Ln(Ad)/T_only_for
 5    _Diagonalizable_Ad
 6    Ad(jfr,t)=M
 7    Ad(egv,t)=egd
 8    egd,3,eps(efjf,t)=LnJf
 9     M,LnJf,M(-1)(*)(*),T
      (s/,t)=Ac
10    (lis)=
```

POM.SUB

```
 1    r,A(pom,sub)=R
 2    (nli)=
 3   _Implemented_by_POM_operator
 4    _R=r(A)_polynomial_of
      _(n_x_n)_A
 5    r(mcp)=p
 6    p(cdi)=k
 7    A(cdi)=n
 8    n,n(dim)=I
 9    n,n(dzm)=R
10 i:k(dec)=k
11    p,k(ctc)=p,pk
12    R,A(*),I,pk(s*)(+)=R
13    k(ifj)=j,j,i
14 j:(nop)=
15    (lis)=
```

POLR.SUB

```
 1    p,A(polr,sub)=r
 2    (nli)=
 3    _Implemented_by_POLR
      _operator
 4    _(n-1)-order_pol._r(x)_and_
 5    _N-order_pol._p(x)_satisfy
 6    _r(A)=p(A)
 7    _Algorithm_uses_C-H-Theorem
 8    A(sstf)=f
 9    p(mcp)=r
10    A(cdi)=n
11    f,n(ctc,t)=f,fn
12    r(cdi)=N
13 i:N(dec)=N
14    r,N(ctc,t)=r,rN
15    N,n(-)=m
16    1,m(dzm),f,rN(s*)(cti,t)=x
17    r,x(-,t)=r
18    m(ifj)=j,j,i
19 j:(nop)=
20    (lis)=
```

FACT.SUB

```
 1    n(fact,sub)=f
 2    (nli)=
 3    _Generates_f={fi},i=[o,n]
 4    _fi=1/i!
 5    I(coin)=one
 6    n(coin)=N
 7    one,one,N(cti)=v
 8    v(gtm)(t)=v
 9    v(mcp)=fa
10    1,n(step)=st
11    O(coin)=i
12 I:i(inc)=i
13    v,N,i(-)(ctc,t)=v1,x
14    st,i(ctc,t)=onev,x
15    fa,onev,v1(cti)(f*,t)=fa
16    i,n(ifj)=I,j,j
17 j:(nop)=
18    st,fa(f/,t)=ss
19    one,ss(cti,e)=f
20    (lis)=
```

FLN.SUB

```
 1    N(fln,sub)=f
```

```
   (nli)=
   0(coin)=z
   z,z(inc),N(inc)(cti)=v
    1,N(inc)(step),v(gte)(t)
   (f/,t)=f
   (lis)=
```

BCDC.SUB

```
    A4,B4,T,iodc(Bcdc,sub)=
   A5,B5o,B51,P
   (nli)=
   _Bilinear_Transformation:
    _For_iodc=1;_4_term_(CT)
   _(A4,B4)_==>_5_term_(DT)
   _(A5,B5o,B51,P)
      _For_iodc=2;_4_term_(DT)
     _(A4,B4)_==>_5_term_(CT)
     _(A5,B5o,B51,P)
   A4(cdi)=n
   n,n(dim)=I
   2(coin),T(s/,t)=a
   iodc,1(ifj)=c,c,d
c:(nop)=
   I,a(s*)=Ia
   Ia,A4(-)(-1)=IAin
    IAin,Ia,A4(+)(*),IAin,B4
   (*)(mcp,t)=A5,B5o
   B5o,I(mcp)=B51,P
   (jmp)=f
d:(nop)=
   A4,I(+)(-1)=AIin
   AIin,a(s*),A4,I(-)(*),AIin,
   B4(*),-1(s*)(mcp,t)=A5,B51
    B51,a(s*),-1(s*),I,a
   (s/),-1(s*)(mcp,t)=B5o,P
f:(lis)=
```

R5R4.SUB

```
    A,Bo,P,C,D,eps(r5r4,sub)
   =Be,De
   (nli)=
    _5_matrix_model_==>_4
   _matrix_model
    A(cdi),Bo(cdi),C(rdi)(mcp)
   =n,m,k
   n,n(dim)=I
   A,I,C(mtf,t)=f,V
   V(alt)=Vm,x
   Vm,eps(nrs)=N,R,r
   I(mcp)=InN
   r,n(ifj)=s,n,n
s:(nop)=
    I,N,N(t),N(*)(-1),N(t)
   (*)(*)(-)=InN
n:(nop)=
   InN,I,A,P(*)(+),Bo(*)(*)=Be
   C,P,Bo(*)(*),D(+)=De
   (lis)=
```

R4R5.SUB

```
    A,Be,P,C,De,eps(r4r5,sub)
   =Bon,Dn
   (nli)=
   _4_matrix_model_==>_5
   _matrix_model
    A(cdi),Be(cdi),C(rdi)(mcp)
   =n,m,k
   n,n(dim)=I
   A,I,C(mtf,t)=f,V
   V(alt)=Vm,x
   Vm,eps(nrs)=N,R,r
   I(mcp)=InN
   r,n(ifj)=s,n,n
s:(nop)=
    I,N,N(t),N(*)(-1),N(t)
   (*)(*)(-)=InN
n:(nop)=
    InN,I,A,P(*)(+)(-1)(*),Be
   (*)=Bon
   De,C,P,Bon(*)(*)(-)=Dn
   (lis)=
```

SQM.SUB

```
   A,eps(sqm,sub)=X
   (nli)=
 _Implemented_by_SQM_operator
   _X_satisfies_X*X=A
   A(cdi)=n
   n,n(dim)=I
   I(mcp)=X
   20(coin)=im
   0(coin)=i
i:i(inc)=i
   i,im(ifj)=c,j,j
c:(nop)=
   X,A,X(-1)(*)(+),2(s/,t)=Xn
   X,Xn(-)(nrr)=del
   Xn(mcp)=X
   del,eps(ifj)=j,j,i
j:(lis)=
```

CHAPTER 3

STR.SUB

```
   A,B,C,T(str,sub)=At,Bt,Ct
   (nli)=
    _Similarity_Transformation
  _Implemented_by_operator_STR
   T(-1),A,T(*)(*)=At
   T(-1),B(*)=Bt
   C,T(*)=Ct
   (lis)=
```

SMAT.SUB

```
 1    Ind(SMat,sub)=
      nx,Sa,Si,Sli,Sld
 2    (nli)=
 3     _PCI_or_POI_==>_Selector
      _Matrices
 4    Ind(poi)=n,nx,va,vi,vli,vld
 5    va(dsm),vi(dsm)(mcp)=Sa,Si
 6     vli(dsm),vld(dsm)(mcp)
      =Sli,Sld
 7    (lis)=
```

IND.SUB

```
 1    Q,m,cut,eps(Ind,sub)=Ind
 2    (nli)=
 3    _Q==>_Contr._Indices_Ind
 4    _m_#_of_inputs
 5    _cut_usually_=_0
 6    _Dets_indices_of_Ind._cols
 7    _in_Q_==>_vlit
 8     _Eliminates_the_first
      _"cut"_unities_==>_vli
 9    _Then_vli_==>_Ind
10    Q,eps(rkc)=x,y,vlit
11    _vlit_aux._sel._vector
12    _vli_sel._vector
13    vlit,cut(ctc)=x,vli
14    vli(cdi),m(s/,t)=k
15     1,k(step),vli,k(vtm)(t)
      (*,t)=Ind
16    (lis)=
```

C#.SUB

```
 1    A(C#,sub)=Adeg
 2    (nli)=
 3     _Adeg=Min(sing._val.)_/
      _Max(sing._val.)_of_A
 4     _Adeg_=_Admissibility
       _degree_=_1/Cond.#
 5    A(rdi),A(cdi)(mcp)=n,m
 6    n,m(ifj)=s,f,f
 7 s:A(t)(svd)=w
 8    (jmp)=F
 9 f:A(svd)=w
10 F:w(cdi)=nc
11     w,1,nc,1,1(exm),w,1(ctc)
      (s/,e)=Adeg
12    Adeg(out,e)=
13    (lis)=
```

CIND.SUB

```
 1    vli,m(cind,sub)=Ind
 2    (nli)=
 3    _Sel._vector_vli_==>_Ind
 4    _Ind_=_PCI_or_POI
 5    _m_=_#_of_Inputs/outputs
 6    vli(cdi),m(s/,t)=n
 7     1,n(step),vli,n(vtm)(t)
      (*,t)=Ind
 8    (lis)=
```

CFPP.SBR

```
 1    A,B,eps(CFpp,sbr)=Tc,Ind
 2    nli
 3    nty
 4     _Tc_=_Sim._Tr._into
      _Feedback_C_Form
 5    _Ind_=_Unique_CI
 6    I,2(dsm)(tvc)=Tc,Ind
 7    A(cdi),B(cdi)(mcp)=n,m
 8    A,B(qc)=Qc
 9    Qc,eps(rkc)=Qc1,x,vli
10     _By_substituting_CIND.SUB
      _with_its_code,_
11     _CFPP_may_be_converted
      _into_SUB
12    vli,m(cind,sub)=Ind
13   Ind(poi)=nn,nx,va,vi,vli,vld
14    A,B,nx(inc)(qc)=Qc
15     vli(dsm),vld(dsm)(mcp)=
      Sli,Sld
16    Qc,Sli(*)=qc1
17    Qc1(-1)=Qcri
18    Qcri,n,m(-)(ctr)=x,Pco
19    va(dsm)(t),Qcri(*,t)=Pc
20    A,Pc(Qo)=Tt
21    Tt,eps(rkr,t)=Tc,x,ro
22 q:typ
23    lis
```

CFNS.SBR

```
 1    A,B,eps(CFns,sbr)=Tc,Ind
 2    nli
 3    nty
 4   _Tc_=_Sim._Tr._into_Feedback
      _C_Form
 5    _Ind_=_Unique_CI
 6    _Using_Null_space_approach
 7    I,2(dsm)(tvc)=Tc,Ind
 8    A(cdi),B(cdi)(mcp)=n,m
 9    A,B(qc)=Qc
10    Qc,eps(rkc)=Qc1,y,vli
11    vli,m(cind,sub)=Ind
12   Ind(poi)=nn,nx,va,vi,vli,vld
13     0,n(dsm),1,n(step),va(dsm)
      (t),0(coin)(mcp)
      =rows,onv,Sat,i
14 i:i(inc)=i
15    Sat,i(ctr)=si,Sat
16    Qc1,onv,si(-)(dsm)(*,t)=Mi
17    Qc1,si(t)(*,t)=qi
18     Mi(t),eps(nrs)(t),qi(*,t)
      =ali
19    Mi(t),eps(nrs)(t,t)=ti
20    ti,ti,qi(*)(s/,t)=ti
21    rows,ti(rti,t)=rows
```

```
   i,m(ifj)=i,j,j
j:(nop)=
   A,rows(qo)=Qo
   Qo,eps(rkr)=Tc
q:typ
   lis
```

SECTION 4.1

SSRO.SUB

```
    A,B,C,D,no,eps(SSRo,sub)
    =Ao,Bo,Co,Do,Deg
    (nli)=
  _Genl_SS_==>_POF_based_on_no
   _D(mcp)=Do
    no(poi,t)
    =nn,nx,va,vi,vli,vld
    eps,eps,eps,eps(mcp)
    =Ao,Bo,Co,Deg
    no(mcp)=-
    A(cdi)=n
    n,nn(ifj)=n,e,n
n:n,-(out)=
     _POI_no_not_compatible
    _with_A
    _(jmp)=O
e:(nop)=
o:A,B,C(mcp)=Ao,Bo,Co
    Ao,Co,nx(inc)(qo)=Qo
    vli(dsm)(t),Qo(*,t)=T
    T(svd)=w
    w,n(dec)(ctc)=x,wn
    wn,w,l(ctc)(s/)=Deg
    Deg(out,e)=
    Deg,eps(ifj)=x,x,w
x:Deg(out,e)=
    -(out)=
    _POI_not_admissible
    _(jmp)=O
w:(nop)=
     Ao,Bo,Co,T(-1)(str,t)
    =Ao,Bo,Co
O:(nop)=
    (lis)=
```

SSRC.SUB

```
     A,B,C,D,nc,eps(SSRc,sub)
    =Ac,Bc,Cc,Dc,Deg
    (nli)=
  _Genl_SS_==>_PCF_based_on_nc
   _D(mcp)=Dc
    nc(poi,t)
    =nn,nx,va,vi,vli,vld
     eps,eps,eps,eps(mcp)
    =Ac,Bc,Cc,Deg
    nc(mcp)=-
    A(cdi)=n
    n,nn(ifj)=n,e,n
n:n,-(out)=
     _PCI_nc_not_compatible
     _with_A
    _(jmp)=O
e:(nop)=
c:A,B,C(t)=Ac,Cc,Bc
    Ac,Cc,nx(inc)(qo)=Qo
    vli(dsm)(t),Qo(*,t)=T
    T(svd)=w
    w,n(dec)(ctc)=x,wn
    wn,w,l(ctc)(s/)=Deg
    Deg(out,e)=
    Deg,eps(ifj)=x,x,w
x:Deg(out,e)=
    -(out)=
    _PCI_not_admissible
    _(jmp)=O
w:(nop)=
     Ac,Bc,Cc,T(-1)(str,t)
    =Ac,Bc,Cc
C:Ac,Bc,Cc(t)=Ac,Cc,Bc
O:(nop)=
     (lis)=
```

SSH.SUB

```
    A,B,C,D,M(SSH,sub)=H,hM
     (nli)=
    _General_SS_==>_Markov
    _parameters_H_in_PMF
    B(cdi)=m
  D,C,A,B,M(dec)(qc)(*)(cti)=H
    H,m,M(dec)(*)(ctc)=x,hM
    hM(nrr,e)=hM
    H,m(pmfr)=H
     (lis)=
```

RODN.SUB

```
     Ao,Bo,Co,Do,no(RoDN,sub)
    =Dl,Nl
    (nli)=
     _POF_==>_Left_coprime_MFD
    _(Dl,Nl)
  Ao(cdi),Bo(cdi),Co(rdi)(mcp)
    =n,m,p
    Ao,Bo,Co,Do(mtf)=do,Wo
    Wo(alt)=Wc,Wr
    no(poi)=nn,nx,va,vi,vli,vld
     va(dsm),vi(dsm),vli(dsm),
    vld(dsm)(mcp)=Sa,Si,Sli,Sld
    Sa(t),Ao(*,t)=Ar
    Sld(t),Ar,Sli(t)(*)(-,t)=Dr
    Dr,p(pmfr)=Dl
    Dl(p-1)=Dli,det
    det(elz)(pnr,t)=x,dnn
    det(elz)=det
    det(pnr)=x,dnn
    Dli,dnn(s/)=Dli
    Dli(alt)=Dic,x
     Dic,p,nx(inc),l(Toep),n
     (inc),p(*)(ctr,t)=Dmt
```

```
   Dmt,Wc(sle,t)=Ncc
   Ncc,p(gmfc)=Nl
   (lis)=
```

RCND.SUB

```
    Ac,Bc,Cc,Dc,nc(RcND,sub)
   =Nr,Dr
   (nli)=
    _PCF_==>_Right_coprime_MFD
   _(Nr,Dr)
   Bc(cdi),Cc(rdi)(mcp)=m,p
   Ac,Bc,Cc,Dc(mtf,t)=dc,Wc
   Wc(alt)=Wc1,Wr
   nc(poi)=n,nx,va,vi,vli,vld
    va(dsm),vi(dsm),vli(dsm),
   vld(dsm)(mcp)=Sa,Si,Sli,Sld
   Ac,Sa(*,t)=Ac1
   Sld,Sli,Ac1(*)(-,t)=Drc
   Drc,m(pmfc,t)=Dr
   Dr(p-1)=Dri,det
   det(elz)(pnr,t)=x,dnn
   det(elz)=det
   det(pnr)=x,dnn
   Dri,dnn(s/)=Dri
   Dri(alt)=x,Dri
    Dri(t),m,nx(inc),1(Toep)
   (t),n(inc),m(*)(ctc,t)=Dmt
   Dmt(t),Wr(t)(sle)(t,t)=Nr
   Nr,m(pmfr,t)=Nr
   (lis)=
```

SECTION 4.2

TFRO.SBR

```
    d,W,eps,nos(TFRo,sbr)
   =Ao,Bo,Co,Do,no,Cond
   nli
   nty
    _TF_(d,W)_==>_POF_Ro
   _based_on_no
   1,6(dzm)(tvc)
   =Ao,Bo,Co,Do,no,Cond
    nos,1(coin),0(coin)(mcp)
   =no,k,giv
   giv(mcp)=nx
   W(alt)=x,Nrr
   Nrr(rdi),x(cdi)(mcp)=p,m
   d,m(dpm,sub)=Dr
   Dr(alt)=x,Drr
  0(coin),0(coin)(mcp)=Ind,nol
   no(cdi),1(ifj)=N,N,G
G:1(dma)=giv
   no(poi)=nn,nx,va,vi,vli,vld
   nx(inc)=k
    Drr(t),m,k,1(Toep),Nrr(t),
   m,k,1(Toep),-1(s*)(cti)=DNrt
    (jmp)=x
N:(nop)=
k:k(inc)=k
   k,m(*)=km
    Drr(t),m,k,1(Toep),Nrr(t),
   m,k,1(Toep),-1(s*)(cti)=DNrt
   DNrt,eps(nrs)=w,x,n
   n,km(-,t)=n
   k(dec),n,nol(cti)=inno
   n,nol(-),n(mcp)=del,nol
   inno(out,t,0)=
   del(ifj)=X,K,k
K:(nop)=
   k,nx(inc)(ifj)=k,w,w
w:(nop)=
   n,p(cti)=np
   Ind,1(ifj)=a,d,x
a:(nop)=
    DNrt,p,k,m(*),eps(Ind,sub)
   =no
   1(coin)=Ind
   (jmp)=C
d:(nop)=
   np,no(out,t,0)=
   1,p(inpm)=no
C:(nop)=
   no(poi)=nn,nx,va,vi,vli,vld
   nn,n(ifj)=d,o,d
o:(nop)=
   Ind(inc)=Ind
   k,nx(inc)(ifj)=k,x,x
x:(nop)=
   1,k,p(*)(dzm)=zv
    zv,vli(p+),zv,vld(p+)(mcp)
   =vli,vld
   k,m(*)=km
   1,km(stmp),vli(cti)=vli
   1,km(dsm),vld(cti)=vld
    vli(dsm),vld(dsm)(mcp)
   =Sli,Sld
  DNrt,Sli(*),DNrt,Sld(*)(mcp)
   =H1,H2
   H1(svd)=w
   w,w(cdi)(dec)(ctc)=x,wn
   wn,w,1(ctc)(s/)=Cond
   no(out,t,0)=
   Cond(out,e)=
   giv(ifj)=t,t,J
t:(nop)=
    _For_different_POI;
    _Enter_j,d(dch)=ch1
  _Otherwise_Enter;_c(dch)=ch2
   ch1(tch)=
   ch2(tch)=
   (sto)=
J:(nop)=
   H1,H2(sle)(t),-1(s*,t)=ND1
   ND1,km(ctc,t)=N1r,D1r
   no(poi)=n,nx,va,vi,vli,vld
    va(dsm),vi(dsm),vli(dsm),
   vld(dsm)(mcp)=Sa,Si,Sli,Sld
   n,n(dim),p(ctr)=Co,A2
   Si,A2(*),Sa,D1r(*)(-,t)=Ao
   N1r,m(r2c)=Nc
```

```
    Ao,Sa,nx(inc)(Qo),Nc(*,t)=Bo
     d,p(dpm,sub)=Dr
      Nrr,m,d(cdi)(dec)(*)(ctc)
     =x,Do
  q:typ
     lis
```

TFRC.SBR

```
     d,W,eps,nca(TFRc,sbr)
     =Ac,Bc,Cc,Dc,nc,Cond
     nli
     nty
      _TF_(d,W)_==>_PCF_Rc
      _based_on_nc
     1,6(dzm)(tvc)
     =Ac,Bc,Cc,Dc,nc,Cond
      nca,1(coin),0(coin)(mcp)
     =nc,k,giv
     giv(mcp)=nx
     W(alt)=Nc,x
     Nc(cdi),x(rdi)(mcp)=m,p
     d,p(dpm,sub)=Dl
     Dl(alt)=Dc
     giv,giv(mcp)=Ind,nol
     nc(cdi),1(ifj)=k,k,G
  G:1(dma)=giv
     nc(poi)=nn,nx,va,vi,vli,vld
     nx(inc)=k
      Dc,p,k,1(Toep),Nc,p,k,1
      (Toep),-1(s*)(cti)=DN1
      (jmp)=x
  k:k(inc)=k
     k,p(*)=kp
      Dc,p,k,1(Toep),Nc,p,k,1
      (Toep),-1(s*)(cti)=DN1
     DN1,eps(nrs)=w,x,n
     n,kp(-,t)=n
     k(dec),n,nol(cti)=inno
     n,nol(-),n(mcp)=del,nol
     inno(out,t,0)=
     del(ifj)=K,K,k
  K:(nop)=
     k,nx(inc)(ifj)=k,w,w
  w:(nop)=
     n,m(cti)=nm
     Ind,1(ifj)=a,d,x
  a:(nop)=
   DN1,m,k,p(*),eps(Ind,sub)=nc
     1(coin)=Ind
     (jmp)=C
  d:(nop)=
     nm,nc(out,t,0)=
     1,m(inpm)=nc
  C:(nop)=
     nc(poi)=nn,nx,va,vi,vli,vld
     nn,n(ifj)=d,o,d
  o:(nop)=
     Ind(inc)=Ind
     k,nx(inc)(ifj)=k,x,x
  x:(nop)=
     k,p(*)=kp
     1,k,m(*)(dzm)=zv
      zv,vli(p+),zv,vld(p+)(mcp)
     =vli,vld
     1,kp(step),vli(cti)=vli
     1,kp(dzm),vld(cti)=vld
      vli(dzm),vld(dzm)(mcp)
     =Sli,Sld
      DN1,Sli(*),DN1,Sld(*)(mcp)
     =H1,H2
     H1(avd)=w
     w,w(cdi)(dec)(ctc)=x,wn
     wn,w,1(ctc)(s/)=Cond
     nc(out,t,0)=
     Cond(out,e)=
     giv(ifj)=t,t,J
  t:(nop)=
      _For_different_PCI_
      _Enter_j,d(dch)=ch1
    _Otherwise_Enter;_c(dch)=ch2
     ch1(tch)=
     ch2(tch)=
     (sto)=
  J:(nop)=
     H1,H2(sle),-1(s*,t)=NDr
     NDr,kp(ctr)=Ncc,Dcc
     nc(poi)=n,nx,va,vi,vli,vld
      va(dzm),vi(dzm),vli(dzm),
     vld(dzm)(mcp)=Sa,Si,Sli,Sld
     n,n(dim),m(ctc)=Bc,A2
      A2,Si(t)(*),Dcc,Sa(t)(*)
     (-,t)=Ac
     Ncc,p(c2r)=Nr
      Nr,Ac,Sa(t),nx(inc)(Qo)
     (*,t)=Cc
     d,m(dpm,sub)=Dl
      Nc,p,d(cdi)(dec)(*)(ctr)
     =x,Dc
  q:typ
     lis
```

TRON.SBR

```
     d,W(Tron,sbr)=Ao,Bo,Co
     nli
     nty
     _Strictly_proper_MIMO
         5
_Transfer_function_W(z)/d(z)_==>
         _Observable_uncontr._Ro
     1,3(dzm)(tvc)=Ao,Bo,Co
      d(cdi)(dec),W(ninp),W(rdi)
     (mcp)=n,m,pm
     pm,m(s/)=p
      1,p(step),m(dec),p(dzm)
     (rti)(mtv,t)=s
     m,p(*),0(dzm)=St
     0(coin)=i
  i:i(inc)=i
     St,s(dzm)(cti,t)=St
     s(shr)=s
     i,m(ifj)=i,j,j
  j:(nop)=
   St,W(*,t)=SW
   SW,m(c2r)(t,t)=Bo
   d,d(ccf,sub)=ac,bc,x,y
   n,m(*),n,p(*)(mcp)=nm,np
    nm,nm(dzm),np,np(dzm)(mcp)
   =Ac,Ao
    nm,m(dzm),p,np(dzm)(mcp)
   =Bc,Co
   1(coin)=io
   0(coin)=i
a:i(inc)=i
   Ao,ac(t),io,io(rmp,t)=Ao
   Co,bc(t),i,io(rmp,t)=Co
   io,n(+)=io
   i,p(ifj)=a,b,b
b:(nop)=
q:typ
   lis
```

TRCN.SBR

```
   d,W(TRcn,sbr)=Ac,Bc,Cc
   nli
   nty
   _Strictly_proper_MIMO
_Transfer_function_W(z)/d(z)_==>
     _Controllable_Unobs._Rc
   I,3(dzm)(tvc)=Ac,Bc,Cc
    d(cdi)(dec),W(ninp),W
   (rdi)(mcp)=n,m,pm
   pm,m(s/)=p
   d,d(ccf,sub)=ac,bc,x,y
   W,p(c2r,t)=Cc
   n,m(*),n,p(*)(mcp)=nm,np
    nm,nm(dzm),np,np(dzm)(mcp)
   =Ac,Ao
    nm,m(dzm),p,np(dzm)(mcp)
   =Bc,Co
   1(coin)=ic
   0(coin)=i
I:i(inc)=i
   Ac,ac,ic,ic(rmp,t)=Ac
   Bc,bc,ic,i(rmp,t)=Bc
   ic,n(+)=ic
   i,m(ifj)=I,J,J
J:(nop)=
q:typ
   lis
```

TFDN.SBR

```
    d,W,eps,nos(TFDN,sbr)
   =Dl,Nl,no,Cond
   nli
   nty
     _TF_(d,W)_==>_Left_coprime
    _MFD_(Dl,Nl)_based_on_no
   I,4(dzm)(tvc)=Dl,Nl,no,Cond
    nos,1(coin),0(coin)(mcp)
   =no,k,giv
   W(alt)=x,Nrr
   giv(mcp)=nx
   Nrr(rdi),x(cdi)(mcp)=p,m
   d,m(dpm,sub)=Dr
   Dr(alt)=x,Drr
   giv,giv(mcp)=Ind,nol
   no(cdi),1(ifj)=k,k,G
G:l(dma)=giv
   no(poi)=nn,nx,va,vi,vli,vld
   nx(inc)=k
    Drr(t),m,k,1(Toep),Nrr(t),
   m,k,1(Toep),-1(s*)(cti)=DNrt
   (jmp)=x
k:k(inc)=k
   k,m(*)=km
    Drr(t),m,k,1(Toep),Nrr(t),
   m,k,1(Toep),-1(s*)(cti)=DNrt
   DNrt,eps(nrs)=w,x,n
   n,km(-,t)=n
   k(dec),n,nol(cti)=inno
   n,nol(-),n(mcp)=del,nol
   inno(out,t,0)=
   del(ifj)=K,K,k
K:(nop)=
   k,nx(inc)(ifj)=k,w,w
w:(nop)=
   n,p(cti)=np
   Ind,1(ifj)=a,d,x
a:(nop)=
    DNrt,p,k,m(*),eps(Ind,sub)
   =no
   1(coin)=Ind
   (jmp)=C
d:(nop)=
   np,no(out,t,0)=
   1,p(inpm)=no
C:(nop)=
   no(poi)=nn,nx,va,vi,vli,vld
   nn,n(ifj)=d,o,d
o:(nop)=
   Ind(inc)=Ind
   k,nx(inc)(ifj)=k,x,x
x:(nop)=
   k,m(*)=km
   1,k,p(*)(dzm)=zv
    zv,vli(p+),zv,vld(p+)(mcp)
   =vli,vld
   1,km(step),vli(cti)=vli
   1,km(dzm),vld(cti)=vld
    vli(dsm),vld(dsm)(mcp)
   =Sli,Sld
 DNrt,Sli(*),DNrt,Sld(*)(mcp)
   =H1,H2
   H1(svd)=w
   w,w(cdi)(dec)(ctc)=x,wn
   wn,w,1(ctc)(s/)=Cond
   no(out,t,0)=
   Cond(out,e)=
   giv(ifj)=t,t,J
t:(nop)=
     _For_different_POI_
    _Enter_j,d(dch)=ch1
 _Otherwise_Enter;_c(dch)=ch2
```

```
   ch1(tch)=
   ch2(tch)=
   (sto)=
J:(nop)=
   H1,H2(sle)(t),-1(s*,t)=ND1
   ND1,km(ctc,t)=Nlr,Dlr
   vli,km(ctc)=x,vlii
   vld,km(ctc)=x,vldd
    Dlr,vlii(dsm)(t)(*),vldd
   (dsm)(t)(+,t)=Dlr
   Dlr,p(pmfr)=Dl
   Nlr,m(pmfr)=Nl
q:typ
   lis
```

TFND.SBR

```
   d,W,eps,ncs(TFND,sbr)
   =Nr,Dr,nc,Cond
   nli
   nty
    _TF_(d,W)_==>_Right_coprime
    _MFD_(Nr,Dr)_Based_on_nc
   I,4(dsm)(tvc)=Nr,Dr,nc,Cond
    ncs,1(coin),0(coin)(mcp)
   =nc,k,giv
   giv(mcp)=nx
   W(alt)=Nc,x
   Nc(cdi),x(rdi)(mcp)=m,p
   d,p(dpm,sub)=Dl
   Dl(alt)=Dc
   giv,giv(mcp)=Ind,nol
   nc(cdi),1(ifj)=k,k,G
G:1(dma)=giv
   nc(poi)=nn,nx,va,vi,vli,vld
   nx(inc)=k
    Dc,p,k,1(Toep),Nc,p,k,1
   (Toep),-1(s*)(cti)=DN1
   (jmp)=x
k:k(inc)=k
   k,p(*)=kp
    Dc,p,k,1(Toep),Nc,p,k,1
   (Toep),-1(s*)(cti)=DN1
   DN1,eps(nrs)=w,x,n
   n,kp(-,t)=n
   k(dec),n,nol(cti)=inno
   n,nol(-),n(mcp)=del,nol
   inno(out,t,0)=
   del(ifj)=K,K,k
K:(nop)=
   k,nx(inc)(ifj)=k,w,w
w:(nop)=
   n,m(cti)=nm
   Ind,1(ifj)=a,d,x
a:(nop)=
  DN1,m,k,p(*),eps(Ind,sub)=nc
   1(coin)=Ind
   (jmp)=C
d:(nop)=
   nm,nc(out,t,0)=
   1,m(inpm)=nc
C:(nop)=
   nc(poi)=nn,nx,va,vi,vli,vld
   nn,n(ifj)=d,o,d
o:(nop)=
   Ind(inc)=Ind
   k,nx(inc)(ifj)=k,x,x
x:(nop)=
   k,p(*)=kp
   1,k,m(*)(dsm)=zv
    zv,vli(p+),zv,vld(p+)(mcp)
   =vli,vld
   1,kp(step),vli(cti)=vli
   1,kp(dsm),vld(cti)=vld
    vli(dsm),vld(dsm)(mcp)
   =Sli,Sld
    DN1,Sli(*),DN1,Sld(*)(mcp)
   =H1,H2
   H1(svd)=w
   w,w(cdi)(dec)(ctc)=x,wn
   wn,w,1(ctc)(s/)=Cond
   nc(out,t,0)=
   Cond(out,e)=
   giv(ifj)=t,t,J
t:(nop)=
    _For_different_PCI_
    _Enter_j,d(dch)=ch1
  _Otherwise_Enter;_c(dch)=ch2
   ch1(tch)=
   ch2(tch)=
   (sto)=
J:(nop)=
   H1,H2(sle),-1(s*,t)=NDr
   NDr,kp(ctr,t)=Ncc,Dcc
   vli,kp(ctc)=x,vlii
   vld,kp(ctc)=x,vldd
    vlii(dsm),Dcc(*),vldd(dsm)
   (+,t)=Dcc
   Dcc,m(pmfc)=Dr
   Ncc,p(pmfc)=Nr
q:typ
   lis
```

CDTR.SBR

```
   d,W,u,T(cdtr,sbr)=y
   nli
   nty
    _(CD)_or_(DT)_TF_response
    _==>_y
    _For_D-T_response;_T=0
   0(coin)=y
   d,W(EXD,sub)=Wsp,D
   T(ifj)=d,d,c
d:(nop)=
    _D-T_response
   d,Wsp,u(t)(rdt)(t)=y
   y,D,u(*)(+,t)=y
   (jmp)=q
c:(nop)=
    _C-T_response
   _N=_#_of_points_in_u
    _should_satisfy_N_>
   _T_*_|pole-max|
   d,Wsp,u(t),T(rct)=y
   y(t),D,u(*)(+)=y
q:(nop)=
   typ
   lis
```

TFH.SBR

```
   d,W,L(TFH,sbr)=H,hL
   nli
   nty
   _TF_(d,W)_==>_Markov
   _parameters_H
   1,2(dzm)(tvc)=H,hL
   W(alt)=m,Wr
    d(cdi)(dec),m(cdi),Wr(rdi)
   (mcp)=n,m,p
   m,n(inc)(ImL,sub)=InL
   Wr,InL(*,t)=Wr
    d,n(ctc),-1(s*),1(coin)
   (cti,t)=dn
   dn,m(dpm,sub)=D
   D(alt),m,L,1(Toep)=D
   D,n,m(*)(ctr)=x,D
    p,L,m(*)(dzm),Wr,1,1
   (rmp,t)=H
   1(coin)=id
   0(coin)=i
i:i(inc)=i
   D,1,id,L,m(*),m(exm)=Di
   H,Di(*,t)=Hni
   H,Hni,1,id(rmp,t)=H
   id,m(+)=id
   i,L(ifj)=i,j,j
j:(nop)=
   H,L(dec),m(*)(ctc)=x,hL
   hL(nrr)=hL
   H,m(pmfr,t)=H
q:typ
   lis
```

GETD.SUB

```
   A,B,C(getd,sub)=n,m,p
   (nli)=
   _(A,B,C)_==>_(n,m,p)
   _Dimensions
    A(cdi),B(cdi),C(rdi)(mcp)
   =n,m,p
   (lis)=
```

DPM.SUB

```
   p,m(dpm,sub)=P
   (nli)=
   _P(z)=diag(p(z))
   _P_is_in_PMF
   p(t),m,m(dim)(mtv)(*,t)=P
   P(t),m(cmp)=P
   (lis)=
```

IML.SUB

```
   m,L(ImL,sub)=InL
   (nli)=
     _ImL="Inverted"_diag(_In_)
   _L-times
   m(coin),L(coin)(mcp)=M,d
    M,d(*),M(dim),M,d,-2
    (Toep),M,d(*)(ctr,t)=InL
    (lis)=
```

CCF.SUB

```
   den,num(ccf,sub)=A,b,c,d
   (nli)=
   _SISO_Transfer_Function
   _num(z)/den(z)
   _==>_State_space_model
    _(A,b,c,d)
   den(cdi)(dec)=n
   num(cdi)=nn
   1(dec)=d
   den,n(ctc)=dl
   num(mcp)=nl
   n,nn(ifj)=d,s,s
d:num,n(ctc,t)=nl,d
   nl,dl,d(s*),-1(s*)(p+,t)=nl
s:n(dec),n(dim)(shr),dl,-1
    (s*)(rti,t)=A
   n,1(dzm),0(inc),n,1(rmp)=b
   1,n(dzm),nl(rmp)=c
   (lis)=
```

EXD.SUB

```
   d,GD(exD,sub)=G,D
   (nli)=
   _GD(z)/d(z)_==>
   _[G(z)/d(z)+D]
   _Extracts_strictly_proper
   _part_of_G(z)
   _and_matrix_D
   GD(rdi)=pm
   pm,1(ifj)=a,a,c
a:GD,1(cmp)=GD
c:(nop)=
   GD(ninp,t)=m
   d(cdi)(dec,t)=nl
    GD(rdi),nl(inc)(dzm),m
   (cmp),GD(pma),nl(ctc,t)=G,D
    d(t),nl(ctr),m,m(dim)(mtv)
   (*,t)=dlp
   dlp(t),m(cmp)=dlp
   D,m(cmp),dlp(pmm),-1(s*),G,
   m(cmp)(pma,t)=G
   D(t),m(vtm,t)=D
   (lis)=
```

EXD.SBR

```
   d,GD(exD,sbr)=G,D
   nli
   nty
   _GD(z)/d(z)_==>
   _[G(z)/d(z)+D]
   _Extracts_strictly_proper
   _part_of_G(z)
   _and_matrix_D
   I,2(dzm)(tvc)=G,D
   GD(rdi)=pm
   pm,1(ifj)=a,a,c
a:GD,1(cmp)=GD
c:(nop)=
   GD(ninp,t)=m
   d(cdi)(dec,t)=n1
    GD(rdi),n1(inc)(dzm),m
    (cmp),GD(pma),n1(ctc,t)=G,D
   d,n1(ctc),m(dpm,sub)=dlp
    D,m(cmp),dlp(pmm),-1
    (s*),G,m(cmp)(pma,t)=G
   D(t),m(vtm,t)=D
q:typ
    lis
```

FGD.SBR

```
   d,G,D(fgd,sbr)=GD
   nli
   nty
   _G(z)/d(z)+D_==>_GD(z)/d(z)
   _Strictly_proper_part
   _G(z)_and_D
   0(coin)=GD
   G(ninp,t)=m
   d,m(dpm,sub)=dp
    G,D(mtv)(t),m(cmp),dp
    (pmm)(pma,t)=GD
q:typ
    lis
```

GEDO.SUB

```
   Ao,Bo,Co,D,N,eps
   (GeDo,sub)=Do
   (nli)=
  _Generate_Do=D(s)^(-1)*N(s)-
    _-Co*(Is-Ao)^(-1)*Bo
   _for_s=either
   _s=0_or_s=/=_system_pole
   Ao(rdi),Co(rdi),N(alt)(mcp)
   =n,p,Nc
   D(alt)=x,Dr
   Ao(egv)=eg
   eg,n,1,1,1(exm)(abs)=em
   em,eps(ifj)=s,s,g
s:eg(rpt),1,1,1,1(exm)
    (inc,t)=s
   s,s,s(-)(cti)=sc
   D,sc(gs)=Ds
   N,sc(gs)=Ns
  Ao,n,n(dim),s(s*)(-)(-1)=Aoi
    Co,Aoi,Bo(*)(*),Ds(-1),Ns
    (*)(+,t)=Do
    (jmp)=f
g:(nop)=
 Co,Ao(-1),Bo(*)(*),Dr,p(ctc)
  (-1),Nc,p(ctr)(*)(+,t)=Do
f:(nop)=
   (lis)=
```

GEDC.SUB

```
   Ac,Bc,Cc,N,D,eps
   (GeDc,sub)=Dc
   (nli)=
  _Generate_Dc=N(s)*D(s)^(-1)-
    _-Cc*(Is-Ac)^(-1)*Bc
   _for_s=either
   _s=0_or_s=/=_system_pole
   Ac(cdi),Bc(cdi),D(alt)
   (mcp)=n,m,Dc
   N(alt)=x,Nr
   Ac(egv)=eg
   eg,n,1,1,1(exm)(abs)=em
   em,eps(ifj)=s,s,g
s:eg(rpt),1,1,1,1(exm)
    (inc,t)=s
   s,s,s(-)(cti)=sc
   D,sc(gs)=Ds
   N,sc(gs)=Ns
  Ac,n,n(dim),s(s*)(-)(-1)=Aci
    Cc,Aci,Bc(*)(*),Ns,Ds(-1)
    (*)(+,t)=Dc
    (jmp)=f
g:(nop)=
    Cc,Ac(-1),Bc(*)(*),Nr,m
    (ctc),Dc,m(ctr)(-1)(*)
    (+,t)=Dc
f:(nop)=
    (lis)=
```

SECTION 4.3

HRO.SBR

```
   Hp,eps,nos(HRo,sbr)
   =Ao,Bo,Co,Do,no,Cond
   nli
   nty
   _Markov_parameters_==>
   _POF_Ro_based_on_no
   _Hp_Is_In_PMF
   1,6(dzm)(tvc)
   =Ao,Bo,Co,Do,no,Cond
    nos,1(coin),0(coin)(mcp)
   =no,k,giv
   giv(mcp)=nx
   Hp(alt)=Hc,p
   Hc(cdi),p(rdi)(mcp)=m,p
   giv,giv(mcp)=Ind,nol
   no(cdi),1(ifj)=k,k,G
G:1(dma)=giv
   no(poi)=nn,nx,va,vi,vli,vld
   nn(mcp)=n
   nx(inc)=k
```

```
  k,p(*)=kp
   Hc,kp,2(s*)(ctr),p,k,-2
  (Toep),kp(ctr,t)=x,H
  (jmp)=x
k:k(inc)=k
  k,p(*)=kp
   Hc,kp,2(s*)(ctr),p,k,-2
  (Toep),kp(ctr,t)=x,H
  H,eps(nrs)=w,x,n
  k(dec),n,nol(cti)=inno
  n,nol(-),n(mcp)=del,nol
  inno(out,t,0)=
  del(ifj)=K,K,k
K:(nop)=
  k,nx(inc)(ifj)=k,w,w
w:(nop)=
  n,p(cti)=np
  Ind,1(ifj)=a,d,x
a:(nop)=
  H(t),p,0,eps(Ind,sub)=no
  1(coin)=Ind
  (jmp)=C
d:(nop)=
  np,no(out,t,0)=
  1,p(inpm)=no
C:(nop)=
  no(poi)=nn,nx,va,vi,vli,vld
  nn,n(ifj)=d,o,d
o:(nop)=
  Ind(inc)=Ind
  k,nx(inc)(ifj)=k,x,x
x:(nop)=
  H,nx,p(*)(ctr)=H1
  vli,nx,p(*)(ctc)(dsn)=S
  S(t),H1(*)=H1
  H1(t)(svd)=w
  w,w(cdi)(dec)(ctc)=x,wn
  wn,w,1(ctc)(s/)=Cond
  no(out,t,0)=
  Cond(out,e)=
  giv(ifj)=t,t,J
t:(nop)=
   _For_different_POI_
   ‾Enter_j,d(dch)‾=ch1‾
 _Otherwise_Enter;_c(dch)=ch2
 ‾ch1(tch)=‾
  ch2(tch)=
  (sto)=
J:(nop)=
  H,p(ctr)=x,H2
  H2,nx,p(*)(ctr)=H2
   H1(t),S(t),H2(*)(t)(sle)
  (t,t)=Ao
 H1,m(ctc),p,n(dim),Hc,p(ctr)
  (mcp,t)=Bo,Co,Do
q:typ
  lis
```

HRC.SBR

```
  Hp,eps,ncs(HRC,sbr)
  =Ac,Bc,Cc,Dc,nc,Cond
  nli
  nty
   _Markov_parameters_==>
  ‾PCF_Rc‾based_on_nc‾
  ‾Hp_Is_In_PMF‾
  1,6(dzm)(tvc)=
  Ac,Bc,Cc,Dc,nc,Cond
   ncs,1(coin),0(coin)(mcp)
  =nc,k,giv
  giv(mcp)=nx
  Hp(alt)=Hc,p
  Hc(cdi),p(rdi)(mcp)=m,p
  giv,giv(mcp)=Ind,nol
  nc(cdi),1(ifj)=k,k,G
G:1(dma)=giv
  nc(poi)=nn,nx,va,vi,vli,vld
  nn(mcp)=n
  nx(inc)=k
  k,p(*)=kp
   Hc,kp,2(s*)(ctr),p,k,-2
  (Toep),kp(ctr,t)=x,H
  (jmp)=x
k:k(inc)=k
  k,p(*)=kp
   Hc,kp,2(s*)(ctr),p,k,-2
  (Toep),kp(ctr,t)=x,H
  H,eps(nrs)=w,x,n
  k(dec),n,nol(cti)=inno
  n,nol(-),n(mcp)=del,nol
  inno(out,t,0)=
  del(ifj)=K,K,k
K:(nop)=
  k,nx(inc)(ifj)=k,w,w
w:(nop)=
  n,m(cti)=nm
  Ind,1(ifj)=a,d,x
a:(nop)=
  H(t),m,0,eps(Ind,sub)=nc
  1(coin)=Ind
  (jmp)=C
d:(nop)=
  nm,nc(out,t,0)=
  1,m(inpm)=nc
C:(nop)=
  nc(poi)=nn,nx,va,vi,vli,vld
  nn,n(ifj)=d,o,d
o:(nop)=
  Ind(inc)=Ind
  k,nx(inc)(ifj)=k,x,x
x:(nop)=
  H,nx,m(*)(ctc)=H1
  vli,nx,m(*)(ctc)(dsm)=S
  H1,S(*)=H1
  H1(svd)=w
  w,w(cdi)(dec)(ctc)=x,wn
  wn,w,1(ctc)(s/)=Cond
  nc(out,t,0)=
  Cond(out,e)=
  giv(ifj)=t,t,J
t:(nop)=
   _For_different_PCI_
  ‾Enter_j,d(dch)‾=ch1‾
```

```
     _Otherwise_Enter;_c(dch)=ch2
     ch1(tch)=
     ch2(tch)=
     (sto)=
J:(nop)=
     H,m(ctc)=x,H2
     H2,nx,m(*)(ctc)=H2
     H1,H2,S(*)(sle,t)=Ac
      n,m(dim),H1,p(ctr),Bc,p
     (ctr)(mcp)=Bc,Cc,Dc
q:typ
     lis
```

HTF.SBR

```
     H,eps(HTF,sbr)=d,W
     nli
     nty
      _Markov_parameters_==>
     _W(z)/d(z)
     _Hp_is_in_PMF
      _Calls_either_HTFp.SBR
     _or_HTFm.SBR
     I,2(dzm)(tvc)=d,W
     H(rdi),H(ninp)(mcp,t)=pm,m
     pm,m(s/,t)=p
     p,m(ifj)=p,p,m
p:(nop)=
     H,eps(HTFp,sbr)=d,W
     nli
     nty
     (jmp)=q
m:(nop)=
     H,eps(HTFm,sbr)=d,W
     nli
     nty
q:(nop)=
     typ
     lis
```

HTFP.SBR

```
     Hp,eps(HTFp,sbr)=d,W
     nli
     nty
      _Markov_parameters_==>
     _W(z)/d(z)
     _Hp_is_in_PMF
      _Called_by_HTF.SBR
     I,2(dzm)(tvc)=d,W
     Hp(alt)=Hc,p
     Hc(cdi),p(rdi)(mcp)=n,p
     1(coin)=k
k:k(inc)=k
     k,p(*)=kp
      Hc,kp,2(s*)(ctr),p,k,2
     (Toep),kp(ctr,t)=T1,T2
     T2,eps(nrs)=w,x,n
     k(dec),n(cti)=in
     in(out,t,0)=
     k(dec),n(ifj)=k,K,K
K:(nop)=
     T2,p,n(*)(ctr)=H1,H2
     H1,H2(t)=H1,H2
     H1,eps(nrs)=Ns
     H1,H2(sle)=Ac
      1,n(step),p(dec),n(dzm)
     (rti)(mtv,t)=s
     0(coin)=i
i:i(inc)=i
     s,-1(s*)(inc,t)=sin
     s,s(shr)(mcp)=so,s
     sin(dsm)(t),Ns(*,t)=Nsi
     Nsi,eps(nrs)=x,y,r
     i,r,n,p(dec)(*)(cti)=irn
     irn(out,t,0)=
     r,n,p(dec)(*)(ifj)=s,e,e
e:(nop)=
     Ac,1,i,n,p(*),1(exm,t)=Aci
     sin(dsm)(t),Aci(*,t)=Acii
     Nsi(-1),Acii(*),-1(s*,t)=ti
     Ns,ti(*),Aci(+,t)=d
     so(dsm)(t),d(*),-1(s*)(t),1
     (coin)(cti,t)=d
     (jmp)=j
s:(nop)=
     i,p(ifj)=i,j,j
j:(nop)=
     d,p(dpm,sub)=D
     D(alt)=x,Dr
     Dr,T1(*),m(pmfr,t)=W
q:typ
     lis
```

HTFM.SBR

```
     Hp,eps(HTFm,sbr)=d,W
     nli
     nty
     Hp(alt)=m,Hr
      _Markov_parameters_==>
     _W(z)/d(z)
     _Hp_is_in_PMF
      _Called_by_HTF.SBR
     I,2(dzm)(tvc)=d,W
     m(cdi),Hr(rdi)(mcp)=n,p
     1(coin)=k
k:k(inc)=k
     k,m(*)=km
      Hr(t),km,2(s*)(ctr),m,k,2
     (Toep)(t),km(ctc,t)=T1,T2
     T2,eps(nrs)=w,x,n
     k(dec),n(cti)=in
     in(out,t,0)=
     k(dec),n(ifj)=k,K,K
K:(nop)=
     T2,m,n(*)(ctc)=H1,H2
     H1,eps(nrs)=Ns
     H1,H2(sle)=Ac
      1,n(step),m(dec),n(dzm)
     (rti)(mtv,t)=s
     0(coin)=i
i:i(inc)=i
     s,-1(s*)(inc,t)=sin
```

```
26    s,s(shr)(mcp)=so,s
27    sin(dsm)(t),Ns(*,t)=Nsi
28    Nsi,eps(nrs)=x,y,r
29    i,r,n,m(dec)(*)(cti)=irn
30    irn(out,t,0)=
31    r,n,m(dec)(*)(ifj)=s,e,e
32 e:(nop)=
33    Ac,1,i,n,m(*),1(exm,t)=Aci
34    sin(dsm)(t),Aci(*,t)=Acii
35    Nsi(-1),Acii(*),-1(s*,t)=ti
36    Ns,ti(*),Aci(*,t)=d
37     so(dsm)(t),d(*),-1(s*)
      (t),1(coin)(cti,t)=d
38    (jmp)=j
39 s:(nop)=
40    i,m(ifj)=i,j,j
41 j:(nop)=
42    d,m(dpm,sub)=D
43    T1,D(alt)(*),p(pmfc,t)=N
44 q:typ
45    lis
```

HDN.SBR

```
 1     Hp,eps,nos(HDN,sbr)
       =D,N,no,Cond
 2     nli
 3     nty
 4      _Markov_parameters_==>
       _Left_coprime_(D(z),N(z))
 5     _with_column_degrees_no
 6     _Hp_,_D_and_N_are_in_PMF
 7     I,4(dzm)(tvc)=D,N,no,Cond
 8      nos,1(coin),0(coin)(mcp)
       =no,k,giv
 9     giv(mcp)=nx
10     Hp(alt)=Hc,p
11     Hc(cdi),p(rdi)(mcp)=m,p
12     giv,giv(mcp)=Ind,nol
13     no(cdi),1(ifj)=k,k,G
14 G:1(dma)=giv
15     no(poi)=nn,nx,va,vi,vli,vld
16     nn(mcp)=n
17     nx(inc)=k
18     k,p(*)=kp
19      Hc,kp,2(s*)(ctr),p,k,2
       (Toep),kp(ctr,t)=T1,T2
20     (jmp)=x
21 k:k(inc)=k
22     k,p(*)=kp
23      Hc,kp,2(s*)(ctr),p,k,2
       (Toep),kp(ctr,t)=T1,T2
24     T2,eps(nrs)=w,x,n
25     k(dec),n,nol(cti)=inno
26     n,nol(-),n(mcp)=del,nol
27     inno(out,t,0)=
28     del(ifj)=K,K,k
29 K:(nop)=
30     k,nx(inc)(ifj)=k,w,w
31 w:(nop)=
32     n,p(cti)=np
33     Ind,1(ifj)=a,d,x
34 a:(nop)=
35     T2(t),p,0,eps(Ind,sub)=no
36     1(coin)=Ind
37     (jmp)=C
38 d:(nop)=
39     np,no(out,t,0)=
40     1,p(inpm)=no
41 C:(nop)=
42     no(poi)=nn,nx,va,vi,vli,vld
43     nn,n(ifj)=d,o,d
44 o:(nop)=
45     Ind(inc)=Ind
46     k,nx(inc)(ifj)=k,x,x
47 x:(nop)=
48     1,kp(dzm)=zv
49      zv,vli(p+),zv,vld(p+)
       (mcp)=vli,vld
50      vli(dsm),vld(dsm)(mcp)
       =Sli,Sld
51      Sli(t),T2(*),Sld(t),T2
       (*)(mcp,t)=H1,H2
52     H1(t)(svd)=w
53     w,w(cdi)(dec)(ctc)=x,wn
54     wn,w,1(ctc)(s/)=Cond
55     no(out,t,0)=
56     Cond(out,e)=
57     giv(ifj)=t,t,J
58 t:(nop)=
59      _For_different_POI_
       _Enter_j,d(dch)=ch1
60   _Otherwise_Enter;_c(dch)=ch2
61     ch1(tch)=
62     ch2(tch)=
63     (sto)=
64 J:(nop)=
65      H1(t),H2(t)(sle)(t),-1
       (s*,t)=Ar
66     Ar,Sli(t)(*),Sld(t)(+,t)=Dr
67     Dr,p(pmfr)=D
68     Dr,T1(*),m(pmfr,t)=N
69 q:typ
70     lis
```

HND.SBR

```
 1     Hp,eps,ncs(HND,sbr)
       =N,D,nc,Cond
 2     nli
 3     nty
 4      _Markov_parameters_==>
       _Right_coprime_(N(z),D(z))
 5     _with_row_degrees_nc
 6     _Hp_,_N_,_and_D_are_in_PMF
 7     I,4(dzm)(tvc)=N,D,nc,Cond
 8      ncs,1(coin),0(coin)(mcp)
       =nc,k,giv
 9     giv(mcp)=nx
10     Hp(alt)=m,Hr
11     m(cdi),Hr(rdi)(mcp)=m,p
12     giv,giv(mcp)=Ind,nol
13     nc(cdi),1(ifj)=k,k,G
14 G:1(dma)=giv
```

```
   nc(poi)=nn,nx,va,vi,vli,vld
   nn(mcp)=n
   nx(inc)=k
   k,m(*)=km
    Hr(t),km,2(s*)(ctr),m,k,2
   (Toep)(t),km(ctc,t)=T1,T2
   (jmp)=x
k:k(inc)=k
   k,m(*)=km
    Hr(t),km,2(s*)(ctr),m,k,2
   (Toep)(t),km(ctc,t)=T1,T2
   T2,eps(nrs)=w,x,n
   k(dec),n,nol(cti)=inno
   n,nol(-),n(mcp)=del,nol
   inno(out,t,0)=
   del(ifj)=K,K,k
K:(nop)=
   k,nx(inc)(ifj)=k,w,w
w:(nop)=
   n,m(cti)=nm
   Ind,1(ifj)=a,d,x
a:(nop)=
   T2,m,0,eps(Ind,sub)=nc
   1(coin)=Ind
   (jmp)=C
d:(nop)=
   nm,nc(out,t,0)=
   1,m(inpm)=nc
C:(nop)=
   nc(poi)=nn,nx,va,vi,vli,vld
   nn,n(ifj)=d,o,d
o:(nop)=
   Ind(inc)=Ind
   k,nx(inc)(ifj)=k,x,x
x:(nop)=
   1,km(dzm)=zv
    zv,vli(p+),zv,vld(p+)(mcp)
   =vli,vld
    vli(dsm),vld(dsm)(mcp)
   =Sli,Sld
    T2,Sli(*),T2,Sld(*)(mcp,t)
   =H1,H2
   H1(svd)=w
   w,w(cdi)(dec)(ctc)=x,wn
   wn,w,1(ctc)(s/)=Cond
   nc(out,t,0)=
   Cond(out,e)=
   giv(ifj)=t,t,J
t:(nop)=
    _For_different_PCI
    _Enter_j,d(dch)=ch1
  _Otherwise_Enter;_c(dch)=ch2
   ch1(tch)=
   ch2(tch)=
   (sto)=
J:(nop)=
   H1,H2(sle),-1(s*,t)=Ac
   Sli,Ac(*),Sld(*,t)=Dc
   Dc,m(pmfc)=D
   T1,Dc(*),p(pmfc,t)=N
q:typ
   lis
```

UHY.SUB

```
   u,H(uhy,sub)=y
   (nli)=
   _D-T_System_Response_to_u
   _u(m_x_N)_y_(p_x_L)
   _L=min(N,M)
   _H_is_in_PMF
   _M=#_of_Markov_param.
    u(rdi),u(cdi),H(cdi)(mcp)
   =m,N,M
   H(alt)=Hc,p
   p(rdi)=p
   M,N(ifj)=l,g,g
l:M(mcp)=N
g:(nop)=
   u,N(ctc)=uc
   uc,1(r2c)=uc
   Hc,p,N,2(toep)=Hc
   Hc,uc(*),p(c2r)=y
   y,N(ctc)=y
    (lis)=
```

HF.SUB

```
   H,f(Hf,sub)=Hf
   (nli)=
   _Time_scaling_"up"_of_H
   _with_f_<_1
   0(coin),1(coin),H(cdi)
   (cti,t)=v
   v(gts)(t),-1(s*)=v
   v,f(log)(s*)(exp)(dsm,t)=SS
   H,SS(*)=Hf
   Hf,H(ninp)(cmp)=Hf
   (lis)=
```

SECTION 4.4

DNRO.SUB

```
   D,N,eps(DNRo,sub)
   =Ao,Bo,Co,Do,no
   (nli)=
    _Left_coprime_(D(z),N(z))
    _==>_POF_Ro_based_on_no
   _no_is_equal_to_column
    _degrees_of_D(z)
   D(alt)=x,Dr
    N(alt),N(rdi),N(ninp)(mcp)
   =Nc,mp,m
   mp,m(s/,t)=p
   Dr,eps(D2nv,t)=no
   no(poi)=n,nx,va,vi,vli,vld
    va(dsm),vi(dsm),vli(dsm),
   vld(dsm)(mcp)=Sa,Si,Sli,Sld
   n,n(dim),p(ctr)=Co,A2
    Si,A2(*),Sa,Dr,Sli(*)(*)
   (-,t)=Ao
  Ao,Sa,nx(inc)(Qc),Nc(*,t)=Bo
   Ao(egv)=eg
   eg,n,1,1,1(exm)(abs)=em
   em,eps(lfj)=s,e,g
 s:eg(rpt),1,1,1,1(exm)
   (inc,t)=s
   s,s,s(-)(cti)=sc
   D,sc(gs)=Ds
   N,sc(gs)=Ns
  Ao,n,n(dim),s(s*)(-)(-1)=Aoi
   Co,Aoi,Bo(*)(*),Ds(-1),
   Ns(*)(+,t)=Do
   (jmp)=f
 g:(nop)=
   Co,Ao(-1),Bo(*)(*),Dr,p
   (ctc)(-1),Nc,p(ctr)(*)
   (+,t)=Do
 f:(nop)=
   (lis)=
```

NDRC.SUB

```
   N,D,eps(NDRc,sub)
   =Ac,Bc,Cc,Dc,nc
   (nli)=
    _Right_coprime_(N(z),D(z))
    ==> PCF_Rc_based_on_nc
   _nc_is_equal_to_row
    degrees_of_D(z)
   N(alt)=x,Nr
    D(alt),N(rdi),N(ninp)
   (mcp)=Drc,mp,m
   mp,m(s/,t)=1
   Drc(t),eps(D2nv,t)=nc
   nc(poi)=n,nx,va,vi,vli,vld
    va(dsm),vi(dsm),vli(dsm),
   vld(dsm)(mcp)=Sa,Si,Sli,Sld
   n,n(dim),m(ctc)=Bc,A2
    A2,Si(t)(*),Sli(t),Drc,
   Sa(t)(*)(*)(-,t)=Ac
    Nr,Ac,Sa(t),nx(inc)(Qo)
   (*,t)=Cc
   Ac(egv)=eg
   eg,n,1,1,1(exm)(abe)=em
   em,eps(lfj)=s,e,g
 s:eg(rpt),1,1,1,1(exm)
   (inc,t)=s
   s,s,s(-)(cti)=sc
   D,sc(gs)=Ds
   N,sc(gs)=Ns
  Ac,n,n(dim),s(s*)(-)(-1)=Aci
   Cc,Aci,Bc(*)(*),Ns,Ds(-1)
   (*)(+,t)=Dc
   (jmp)=f
 g:(nop)=
   Cc,Ac(-1),Bc(*)(*),Nr,m
   (ctc),Drc,m(ctr)(-1)(*)
   (+,t)=Dc
 f:(nop)=
   (lis)=
```

DNH.SUB

```
   D,N,L(DNH,sub)=H,nrm
   (nli)=
    _Left_coprime_(D(z),N(z))
    ==>
   _First_L_Markov_parameters
   _Applicable_only_for
    _DT_stable_systems
   _D,_N_and_H_are_in_PMF
   D(alt),N(alt)(mcp)=Dc,Nc
    Dc(cdi),Nc(cdi),Dc(rdi)
   (mcp)=p,m,kp
   Dc,p,L,-1(Toep,t)=Dm
    p,L(dec)(*),m(dzm),Nc
   (rti)=Nm
   Dm,Nm(sle)=H
   H,L(dec),p(*)(ctr)=x,y
   H,p(pmfc)=H
   y(nrr)=nrm
   (lis)=
```

NDH.SUB

```
   N,D,L(NDH,sub)=H,nrm
   (nli)=
    _Right_coprime_(N(z),D(z))
    ==>
   _First_L_Markov_parameters
   _Applicable_only_for
    _DT_stable_systems
   _N,_D_,_and_H_are_in_PMF
   D(alt)=x,Dr
   N(alt)=x,Nr
    Dr(rdi),Nr(rdi),Dr(cdi)
   (mcp)=m,p,km
   Dr(t),m,L,-1(Toep)(t)=Dm
    p,L(dec),m(*)(dzm),Nr
   (cti)=Nm
   Dm(t),Nm(t)(sle)(t)=H
   H,L(dec),m(*)(ctc)=x,y
   H,m(pmfr)=H
   y(nrr)=nrm
   (lis)=
```

DNTF.SBR

```
   D,N,eps(DNTf,sbr)=d,W
   nli
   nty
    _Left_coprime_(D(z),N(z))
    ==>
    _Transfer_function
    _W(z)/d(z)
   I,2(dzm)(tvc)=d,W
   D(p-1,t)=Dlad,d
   Dlad,N(pmm,t)=W
   W,eps(elzc,sub)=W
   d(elz)=d
   d(pnr)=x,dn
   d,dn(e/),W,dn(e/)(mcp)=d,W
 q:typ
   lis
```

NDTF.SBR

```
   N,D,eps(NDTf,sbr)=d,W
   nli
   nty
    _Right_coprime_{N(z),D(z)}
     ==>
   _Transfer_function
    W(z)/d(z)
   I,2(dzm)(tvc)=d,W
   D(p-1,t)=Drad,d
   N,Drad(pmm,t)=W
   W,eps(elzc,sub)=W
   d(elz)=d
   d(pnr)=x,dn
   d,dn(s/),W,dn(s/)(mcp)=d,W
q:typ
    lis
```

DNTS.SBR

```
   Dl,Nl,f(DNts,sbr)=Dlf,Nlf
   nli
   nty
    _Time_scaling_of_Left
    _coprime
    _{Dl(z),Nl(z)}_"down"_f_<_1
   I,2(dzm)(tvc)=Dlf,Nlf
   Dl(ninp)=p
   Dl(alt)=x,Dr
   1e-5(dma)=eps
   Dr,eps(d2nv)=no
   no,Dl(cdi),f(tscl,sub)=So
   So,Dl(alt)(*),p(pmfc)=Dlf
   So,Nl(alt)(*),p(pmfc)=Nlf
q:typ
    lis
```

NDTS.SBR

```
   Nr,Dr,f(NDts,sbr)=Nrf,Drf
   nli
   nty
    _Time_scaling_of_Right
    _coprime
    _{Nr(z),Dr(z)}_"down"_f_<_1
   I,2(dzm)(tvc)=Nrf,Drf
   Dr(ninp)=m
   Nr(alt)=x,Nrr
   Dr(alt)=x,Drr
   1e-5(dma)=eps
   x(t),eps(d2nv)=nc
   nc,Dr(cdi),f(tscl,sub)=Sc
   Nrr,Sc(*),m(pmfr)=Nrf
   Drr,Sc(*),m(pmfr)=Drf
q:typ
    lis
```

TSCL.SUB

```
   no,n,f(tscl,sub)=S
   (nli)=
    _S_=_"Time_scaling"_diag
    _matrix
   _f_<_1
   _Called_by_DNTS.SBR_&
    NDTS.SBR
   0,n(dzm)=s
   no(cdi)=p
   no(ord),p,p(diim)(*)=noo
   0(coin)=i
I:i(inc)=i
   noo,1,i(exm,t)=noi
   noi,-1(coin),n(cti)=x
    x(gts)(t),f(log)(s*)
    (exp,t)=vec
   s,vec(rti,t)=s
   i,p(ifj)=I,J,J
J:(nop)=
   s(mtv)(ddm,t)=S
   (lis)=
```

DNRC.SBR

```
   D,N,eps,ncs(DNRc,sbr)
   =Ac,Bc,Cc,Dc,nc,Cond
   nli
   nty
    _Left_coprime_{D(z),N(z)}
     ==>
   _PCF_Rc_based_on_nc
   I,6(dzm)(tvc)=
   Ac,Bc,Cc,Dc,nc,Cond
    ncs,1(coin),0(coin)(mcp)
   =nc,k,giv
   giv(mcp)=nx
   D(alt),N(alt)(mcp)=Dc,Nc
   Dc(cdi),Nc(cdi)(mcp,t)=p,m
   giv,giv(mcp)=Ind,nol
   nc(cdi),1(ifj)=k,k,G
G:1(dma)=giv
   nc(poi)=nn,nx,va,vi,vli,vld
   nx(inc)=k
   k,p(*)=kp
    Dc,p,k,1(Toep),Nc,p,k,1
    (Toep),-1(s*)(cti)=DN
    (jmp)=x
k:k(inc)=k
   k,p(*)=kp
    Dc,p,k,1(Toep),Nc,p,k,1
    (Toep),-1(s*)(cti)=DN
   DN,eps(nrs)=w,x,n
   n,kp(-,t)=n
   k(dec),n,nol(cti)=inno
   n,nol(-),n(mcp)=del,nol
   inno(out,t,0)=
   del(ifj)=K,K,k
K:(nop)=
   k,nx(inc)(ifj)=k,w,w
w:(nop)=
   n,m(cti)=nm
   Ind,1(ifj)=a,d,x
a:(nop)=
   DN,m,k,p(*),eps(Ind,sub)=nc
   1(coin)=Ind
```

```
36     (jmp)=C
37  d:(nop)=
38     nm,nc(out,t,0)=
39     1,m(inpm)=nc
40  C:(nop)=
41     nc(poi)=nn,nx,va,vi,vli,vld
42     nn,n(ifj)=d,o,d
43  o:(nop)=
44     Ind(inc)=Ind
45     k,nx(inc)(ifj)=k,x,x
46  x:(nop)=
47     k,p(*)=kp
48     1,k,m(*)(dzm)=zv
49      zv,vli(p+),zv,vld(p+)
       (mcp)=vli,vld
50     1,kp(step),vli(cti)=vli
51     1,kp(dzm),vld(cti)=vld
52      vli(dsm),vld(dsm)(mcp)
       =Sli,Sld
53      DN,Sli(*),DN,Sld(*)(mcp)
       =H1,H2
54     H1(svd)=w
55     w,w(cdi)(dec)(ctc)=x,wn
56     wn,w,1(ctc)(s/)=Cond
57     nc(out,t,0)=
58     Cond(out,e)=
59     giv(ifj)=t,t,J
60  t:(nop)=
61       _For_different_PCI_
       ‾Enter_j,d(dch)=ch1
62   _Otherwise_Enter;_c(dch)=ch2
63     ch1(tch)=
64     ch2(tch)=
65     (sto)=
66  J:(nop)=
67     N1,N2(ele),-1(s*,t)=NDr
68     NDr,kp(ctr)=Ncc,Dcc
69     nc(poi)=n,nx,va,vi,vli,vld
70      va(dsm),vi(dsm),vli(dsm),
       vld(dsm)(mcp)=Sa,Si,Sli,Sld
71     n,n(dim),m(ctc)=Bc,A2
72      A2,Si(t)(*),Dcc,Sa(t)(*)
       (-,t)=Ac
73     Ncc,p(c2r)=Nr
74      Nr,Ac,Sa(t),nx(inc)(Qo)
       (*,t)=Cc
75      vli(dsm),Dcc(*),vld(dsm)
       (+,t)=Dcc
76     Dcc,m(pmfc)=D1
77     Ncc,p(pmfc)=N1
78      Ac,Bc,Cc,N1,D1,eps
       (GeDc,sub)=Dc
79  q:typ
80     lis
```

NDRO.SBR

```
 1     N,D,eps,nos(NDRo,sbr)
       =Ao,Bo,Co,Do,no,Cond
 2     nli
 3     nty
 4      _Right_coprime_{N(z),D(z)}
        ==>
 5     ‾POF_Ro_based_on_no
 6     1,6(dzm)(tvc)=
       Ao,Bo,Co,Do,no,Cond
 7      nos,1(coin),0(coin)(mcp)
       =no,k,giv
 8     giv(mcp)=nx
 9     N(alt)=x,Nrr
10     D(alt)=x,Drr
11     Nrr(rdi),Drr(rdi)(mcp)=p,m
12     giv,giv(mcp)=Ind,nol
13     no(cdi),1(ifj)=k,k,G
14  G:1(dma)=giv
15     no(poi)=nn,nx,va,vi,vli,vld
16     nx(inc)=k
17     k,m(*)=km
18      Drr(t),m,k,1(Toep),Nrr(t),
       m,k,1(Toep),-1(s*)(cti)=DN
19     (jmp)=x
20  k:k(inc)=k
21     k,m(*)=km
22      Drr(t),m,k,1(Toep),Nrr(t),
       m,k,1(Toep),-1(s*)(cti)=DN
23     DN,eps(nrs)=w,x,n
24     n,km(-,t)=n
25     k(dec),n,nol(cti)=inno
26     n,nol(-),n(mcp)=del,nol
27     inno(out,t,0)=
28     del(ifj)=K,K,k
29  K:(nop)=
30     k,nx(inc)(ifj)=k,w,w
31  w:(nop)=
32     n,p(cti)=np
33     Ind,1(ifj)=a,d,x
34  a:(nop)=
35     DN,p,k,m(*),eps(Ind,sub)=no
36     1(coin)=Ind
37     (jmp)=C
38  d:(nop)=
39     np,no(out,t,0)=
40     1,p(inpm)=no
41  C:(nop)=
42     no(poi)=nn,nx,va,vi,vli,vld
43     nn,n(ifj)=d,o,d
44  o:(nop)=
45     Ind(inc)=Ind
46     k,nx(inc)(ifj)=k,x,x
47  x:(nop)=
48     k,m(*)=km
49     1,k,p(*)(dzm)=zv
50      zv,vli(p+),zv,vld(p+)(mcp)
       =vli,vld
51     1,km(step),vli(cti)=vli
52     1,km(dzm),vld(cti)=vld
53      vli(dsm),vld(dsm)(mcp)
       =Sli,Sld
54      DN,Sli(*),DN,Sld(*)(mcp)
       =H1,H2
55     H1(svd)=w
56     w,w(cdi)(dec)(ctc)=x,wn
57     wn,w,1(ctc)(s/)=Cond
58     no(out,t,0)=
   Cond(out,e)=
   giv(ifj)=t,t,J
t:(nop)=
    _For_different_PCI_
   _Enter_j,d(dch)=ch1
  _Otherwise_Enter;_c(dch)=ch2
   ch1(tch)=
   ch2(tch)=
   (sto)=
J:(nop)=
   H1,H2(sle)(t),-1(s*,t)=ND
   ND,km(ctc,t)=Nlr,Dlr
   nc(poi)=n,nx,va,vi,vli,vld
    va(dsm),vi(dsm),vli(dsm),
   vld(dsm)(mcp)=Sa,Si,Sli,Sld
   n,n(dim),p(ctr)=Co,A2
   Si,A2(*),Sa,Dlr(*)(-,t)=Ao
   Nlr,m(r2c)=Nc
  Ao,Sa,nx(inc)(Qc),Nc(*,t)=Bo
   Dlr,vli(dsm)(t)(*),vld(dsm)
   (t)(+,t)=Dlr
   Dlr,p(pmfr)=Dl
   Nlr,m(pmfr)=Nl
    Ao,Bo,Co,Dl,Nl,eps
   (GeDo,sub)=Do
q:typ
   lis
```

DNND.SBR

```
   Dl,Nl,eps,ncs(DNND,sbr)
   =Nr,Dr,nc,Cond
   nli
   nty
   _Left_MFD_{Dl(z),Nl(z)}_==>
   _Right_coprime
   _{Nr(z),Dr(z)}
   _with_row_degrees_nc
   1,4(dzm)(tvc)=Nr,Dr,nc,Cond
    ncs,1(coin),0(coin)(mcp)
   =nc,k,giv
   giv(mcp)=nx
   Dl(alt),Nl(alt)(mcp)=Dc,Nc
   Dc(cdi),Nc(cdi)(mcp,t)=p,m
   giv,giv(mcp)=Ind,nol
   nc(cdi),1(ifj)=k,k,G
G:1(dma)=giv
   nc(poi)=nn,nx,va,vi,vli,vld
   nx(inc)=k
   k,p(*)=kp
    Dc,p,k,1(Toep),Nc,p,k,1
   (Toep),-1(s*)(cti)=DNl
   (jmp)=x
k:k(inc)=k
   k,p(*)=kp
    Dc,p,k,1(Toep),Nc,p,k,1
   (Toep),-1(s*)(cti)=DNl
   DNl,eps(nrs)=w,x,n
   n,kp(-,t)=n
   k(dec),n,nol(cti)=inno
   n,nol(-),n(mcp)=del,nol
   inno(out,t,0)=
   del(ifj)=K,K,k
K:(nop)=
   k,nx(inc)(ifj)=k,w,w
w:(nop)=
   n,m(cti)=nm
   Ind,1(ifj)=a,d,x
a:(nop)=
 DNl,m,k,p(*),eps(Ind,sub)=nc
   1(coin)=Ind
   (jmp)=C
d:(nop)=
   nm,nc(out,t,0)=
   1,m(inpm)=nc
C:(nop)=
   nc(poi)=nn,nx,va,vi,vli,vld
   nn,n(ifj)=d,o,d
o:(nop)=
   Ind(inc)=Ind
   k,nx(inc)(ifj)=k,x,x
x:(nop)=
   k,p(*)=kp
   1,k,m(*)(dzm)=zv
    zv,vli(p+),zv,vld(p+)(mcp)
   =vli,vld
   1,kp(step),vli(cti)=vli
   1,kp(dsm),vld(cti)=vld
    vli(dsm),vld(dsm)(mcp)
   =Sli,Sld
    DNl,Sli(*),DNl,Sld(*)(mcp)
   =H1,H2
   H1(svd)=w
   w,w(cdi)(dec)(ctc)=x,wn
   wn,w,1(ctc)(s/)=Cond
   nc(out,t,0)=
   Cond(out,e)=
   giv(ifj)=t,t,J
t:(nop)=
    _For_different_PCI_
   _Enter_j,d(dch)=ch1
  _Otherwise_Enter;_c(dch)=ch2
   ch1(tch)=
   ch2(tch)=
   (sto)=
J:(nop)=
   H1,H2(sle),-1(s*,t)=NDr
   NDr,kp(ctr,t)=Ncc,Dcc
   vli,kp(ctc)=x,vlii
   vld,kp(ctc)=x,vldd
    vlii(dsm),Dcc(*),vldd
   (dsm)(+,t)=Dcc
   Dcc,m(pmfc)=Dr
   Ncc,p(pmfc)=Nr
q:typ
   lis
```

NDDN.SBR

```
   Nr,Dr,eps,ncs(NDDN,sbr)
   =Dl,Nl,nc,Cond
   nli
   nty
   _Right_MFD_{Nr(z),Dr(z)}_==>
```

```
     _Left_coprime_(Dl(z),Nl(z))
     _with_column_degrees_no
     I,4(dzm)(tvc)=Dl,Nl,no,Cond
      nos,1(coin),0(coin)(mcp)
     =no,k,giv
     Nr(alt)=x,Nrr
     giv(mcp)=nx
     Dr(alt)=x,Drr
     Nrr(rdi),Drr(rdi)(mcp)=p,m
     giv,giv(mcp)=Ind,nol
     no(cdi),1(ifj)=k,k,G
G:1(dma)=giv
     no(poi)=nn,nx,va,vi,vli,vld
     nx(inc)=k
     k,m(*)=km
      Drr(t),m,k,1(Toep),Nrr(t),
     m,k,1(Toep),-1(s*)(cti)=DNrt
     (jmp)=x
k:k(inc)=k
     k,m(*)=km
      Drr(t),m,k,1(Toep),Nrr(t),
     m,k,1(Toep),-1(s*)(cti)=DNrt
     DNrt,eps(nrs)=w,x,n
     n,km(-,t)=n
     k(dec),n,nol(cti)=inno
     n,nol(-),n(mcp)=del,nol
     inno(out,t,0)=
     del(ifj)=K,K,k
K:(nop)=
     k,nx(inc)(ifj)=k,w,w
w:(nop)=
     n,p(cti)=np
     Ind,1(ifj)=a,d,x
a:(nop)=
     DNrt,p,k,m(*),eps
     (Ind,sub)=no
     1(coin)=Ind
     (jmp)=C
d:(nop)=
     np,no(out,t,0)=
     1,p(ingm)=no
C:(nop)=
     no(poi)=nn,nx,va,vi,vli,vld
     nn,n(ifj)=d,o,d
o:(nop)=
     Ind(inc)=Ind
     k,nx(inc)(ifj)=k,x,x
x:(nop)=
     k,m(*)=km
     1,k,p(*)(dzm)=zv
      zv,vli(p+),zv,vld(p*)(mcp)
     =vli,vld
     1,km(step),vli(cti)=vli
     1,km(dzm),vld(cti)=vld
      vli(dsm),vld(dsm)(mcp)
     =Sli,Sld
    DNrt,Sli(*),DNrt,Sld(*)(mcp)
     =H1,H2
     H1(svd)=w
     w,w(cdi)(dec)(ctc)=x,wn
     wn,w,1(ctc)(s/)=Cond
     no(out,t,0)=
     Cond(out,e)=
     giv(ifj)=t,t,J
t:(nop)=
      _For_different_POI_
      _Enter_j,d(dch)=ch1
    _Otherwise_Enter;_c(dch)=ch2
     ch1(tch)=
     ch2(tch)=
     (sto)=
J:(nop)=
     H1,H2(sle)(t),-1(s*,t)=ND1
     ND1,km(ctc,t)=Nlr,Dlr
     vli,km(ctc)=x,vlii
     vld,km(ctc)=x,vldd
      Dlr,vlii(dsm)(t)(*),vldd
     (dsm)(t)(+,t)=Dlr
     Dlr,p(pmfr)=Dl
     Nlr,m(pmfr)=Nl
q:typ
     lis
```

CHAPTER 5

UYRO.SBR

```
     u,y,eps,nos(uyRo,sbr)
     =Ao,Bo,Co,Do,no,xo,Cond
     nli
     nty
      _Input/Output_Data_==>
      _POF_Ro_based_on_no
       _Deterministic_MIMO_system
      _Identification
     I,7(dzm)(tvc)=
     Ao,Bo,Co,Do,no,xo,Cond
     u,y(t)=u,y
      nos,0(coin),0(coin)(mcp)
     =no,i,giv
     giv(mcp)=nx
     giv(mcp)=Ind
      0(coin),0(coin),u(rdi),u
     (cdi),y(cdi)(mcp)=
     nol,i,N,m,p
     u,y(mcp)=U,Y
     no(cdi),1(ifj)=i,i,G
G:1(dma)=giv
     no(poi)=nn,nx,va,vi,vli,vld
     nx(inc),nn(mcp)=i,n
     N,i(dec)(-)=Ni
      u,1,i,-2(toep),y,1,i,-2
     (toep)(mcp)=U,Y
     U,i(dec)(ctr)=x,U
     Y,i(dec)(ctr)=x,Y
     (jmp)=x
i:i(inc)=i
     u(shu),y(shu)(mcp)=u,y
     U,u(cti),Y,y(cti)(mcp)=U,Y
      U,Y(cti),N,i(-)(ctr),eps
     (nrs)=xx,yy,r
     r,i(inc),m(*)(-,t)=n
```

```
  i,n,nol(cti)=inno
  n,nol(-),n(mcp,t)=del,nol
  inno(out,t,0)=
  del(ifj)=c,c,i
c:(nop)=
  i,nx(ifj)=i,w,w
w:(nop)=
  n,p(cti)=np
  N,i(-)=Ni
  Ind,l(ifj)=a,d,x
a:(nop)=
  U,Y(cti),Ni(ctr)=Z
   Z,p,i(inc),m(*),eps
  (Ind,sub)=no
  l(coin)=Ind
  (jmp)=C
d:(nop)=
  no,np(out,t,0)=
  l,p(inpm)=no
C:(nop)=
  no(poi)=nn,nx,va,vi,vli,vld
  n,nn(ifj)=d,o,d
o:(nop)=
  Ind(inc)=Ind
  i,nx(ifj)=i,x,x
x:(nop)=
   va(dsm),vi(dsm),vli(dsm),
  vld(dsm)(mcp)=Sa,Si,Sli,Sld
   U,Ni(ctr),Y,Ni(ctr)(mcp)
  =Uc,Yc
   Uc,nx(inc),m(*)(ctc),Yc,nx
  (inc),p(*)(ctc)(mcp)=Uc,Yc
   Yc,Sli(*),Yc,Sld(*)(mcp)
  =Y1,Y2
  Uc,Y1(cti)=Z
  Z(svd)=w
  w,w(cdi)(dec)(ctc)=x,wn
  wn,w,l(ctc)(s/)=Cond
  no(out,t,0)=
  Cond(out,e)=
  giv(ifj)=t,t,J
t:(nop)=
   _For_different_POI
   _Enter_j,d(dch)=ch1
 _Otherwise_Enter;_c(dch)=ch2
  ch1(tch)=
  ch2(tch)=
  (sto)=
J:(nop)=
   Z,Y2(sle)(t),nx(inc),m(*)
  (ctc,t)=Bt,At
  n,n(dim),p(ctr)=Co,A2
  Sa,At(*),Si,A2(*)(+,t)=Ao
  Ao,Sa,nx(inc)(qc)=Qc
  Bt,m(r2c)=Btt
  Qc,Btt(*,t)=Bo
  Bt,m(ctc)=Bt1
  Bt,m(pmfr)=Np
   Sld(t),At,Sli(t)(*)(-),p
  (pmfr)=Dp
   Ao,Bo,Co,Dp,Np,eps
  (GeDo,sub)=Do
   Y1(t),l(ctc),Uc(t),l(ctc)
  (mcp)=Y11,U1
  Do,Ao,Co,nx(Qo),Bo(*)
  (rti),p,nx,2(Toep)=SS
   Y11,Sli(t),SS(*),U1,nx,m
  (*)(ctr)(*)(-,t)=xo
q:(nop)=
  typ
  lis
```

UYDN.SBR

```
  u,y,eps,nos(uyDN,sbr)
  =Dp,Np,no,Cond
  nli
  nty
   _Input/Output_pair_==>
  _Left_Coprime_(D(z),N(z))
    _Deterministic_MIMO_system
   _Identification
  I,4(dsm)(tvc)=Dp,Np,no,Cond
  u,y(t)=u,y
   nos,0(coin),0(coin)(mcp)
  =no,i,giv
  giv(mcp)=nx
  giv(mcp)=Ind
   0(coin),0(coin),u(rdi),u
  (cdi),y(cdi)(mcp)
  =nol,i,N,m,p
  u,y(mcp)=U,Y
  no(cdi),l(ifj)=i,i,G
G:l(dms)=giv
  no(poi)=nn,nx,va,vi,vli,vld
  nx(inc),nn(mcp)=i,n
  N,i(dec)(-)=Ni
   u,1,i,-2(toep),y,1,i,-2
  (toep)(mcp)=U,Y
  U,i(dec)(ctr)=x,U
  Y,i(dec)(ctr)=x,Y
  (jmp)=x
i:(nop)=
   _here_no(cdi)=1
   _for_p=1_go_to_N
   _for_no_=_or_>_1_go_to_C
  p,i(ifj)=N,N,I
N:no,l(ifj)=I,G,G
I:i(inc)=i
  u(shu),y(shu)(mcp)=u,y
  U,u(cti),Y,y(cti)(mcp)=U,Y
   U,Y(cti),N,i(-)(ctr),eps
  (nrs)=xx,yy,r
  r,i(inc),m(*)(-,t)=n
  i,n,nol(cti)=inno
  n,nol(-),n(mcp,t)=del,nol
  inno(out,t,0)=
  del(ifj)=c,c,i
c:(nop)=
  i,nx(ifj)=I,w,w
w:(nop)=
  n,p(cti)=np
  N,i(-)=Ni
  Ind,l(ifj)=a,d,x
```

```
43 a:(nop)=
44    U,Y(cti),Ni(ctr)=Z
45    Z,p,i(inc),m(*),eps
      (Ind,sub)=no
46    i(coin)=Ind
47    (jmp)=C
48 d:(nop)=
49    no,np(out,t,0)=
50    1,p(inpm)=no
51 C:(nop)=
52    no(poi)=nn,nx,va,vi,vli,vld
53    n,nn(ifj)=d,o,d
54 o:(nop)=
55    Ind(inc)=Ind
56    i,nx(ifj)=i,x,x
57 x:(nop)=
58     va(dsm),vi(dsm),vli(dsm),
      vld(dsm)(mcp)=Sa,Si,Sli,Sld
59     U,Ni(ctr),Y,Ni(ctr)(mcp)
      =Uc,Yc
60     Uc,nx(inc),m(*)(ctc),Yc,
      nx(inc),p(*)(ctc)(mcp)=Uc,Yc
61     Yc,Sli(*),Yc,Sld(*)(mcp)
      =Y1,Y2
62    Uc,Y1(cti)=Z
63    Z(svd)=w
64    w,w(cdi)(dec)(ctc)=x,wn
65    wn,w,1(ctc)(s/)=Cond
66    no(out,t,0)=
67    Cond(out,e)=
68    giv(ifj)=t,t,J
69 t:(nop)=
70     _For_different_POI
      _Enter_j,d(dch)=ch1
71   _Otherwise_Enter;_o(dch)=ch2
72    ch1(tch)=
73    ch2(tch)=
74    (sto)=
75 J:(nop)=
76     Z,Y2(sle)(t),nx(inc),
      m(*)(ctc,t)=Bt,At
77    Bt,m(pmfr,t)=Np
78     Sld(t),At,Sli(t)(*)(-),p
      (pmfr,t)=Dp
79 q:(nop)=
80    typ
81    lis
```

UYTF.SBR

```
 1    U,Y,epsl,nos(uyTF,sbr)
      =dtt,Wt,no,C#
 2    nli
 3    nty
 4     _Input/Output_pair_==>
      _Transfer_Function
 5    _Deterministic_MIMO
      _Identification
 6    _using_one_output_at_a_time
 7    I,4(dzm)(tvc)=dtt,Wt,no,C#
 8   1,0(dzm),1.0(dzm)(mcp)=C#,no
 9    nos(cdi)=dim
10    Y(mcp)=YY
11    U(rdi),Y(rdi)(mcp)=m,p
12    0,10(dzm)=Wt
13    Wt(mcp)=dt
14    0(coin)=i
15 i:i(inc)=i
16    YY,1(ctr)=y1,YY
17    epsl(mcp)=nosi
18    dim,i(ifj)=o,o,g
19 g:nos,1(ctc)=nosi,nos
20 o:(nop)=
21     U,y1,epsl,nosi(uydn,sbr)
      =d1,W1,noi,C#i
22    nli
23    nty
24     dt,1,10(dzm),d1,1,i(rmp)
      (rti,t)=dt
25     Wt,m,10(dzm),W1,1,i(rmp)
      (rti,t)=Wt
26     C#,C#i(cti),no,noi(cti)
      (mcp)=C#,no
27    i,p(ifj)=i,j,j
28 j:(nop)=
29  Wt,p(cmp),epsl(elzc,sub)=Wtt
30   Wtt,p(cmp)(pmt,sub)=Wt
31  dt,1(cmp),epsl(elzc,sub)=dtt
32 q:typ
33    lis
```

PMT.SUB

```
 1    N(pmt,sub)=Nt
 2    (nli)=
 3    _Polynomial_matrix
      _Transposition
 4    N(alt)(t),N(rdi),N(ninp)
      (s/)(pmfr,t)=Nt
 5    (lis)=
```

CTCP.SUB

```
 1    G,m1(CTCp,sub)=G1,G2
 2    (nli)=
 3     _Cut_by_columns_Polynomial
      _matrix
 4    _G(s)==>|G1(s)|G2)s)|
 5    _G1_has_m1_columns
 6    m1(coin)=mm1
 7    G(rdi),G(ninp)(mcp)=pm,m
 8    pm,m(s/)=p
 9    G,mm1,p(*)(ctr)=G1,G2
10     G1,mm1(cmp),G2,m,mm1(-)
      (cmp)(mcp,t)=G1,G2
11    (lis)=
```

CTRP.SUB

```
 1    G,p1(CTRp,sub)=G1,G2
 2    (nli)=
 3     _Cut_by_rows_Polynomial
      _matrix
 4    __________|G1(s)|
    _G(s)==>|-----|
    ________|G2(s)|
    _G1_has_p1_rows
    G(rdi),G(ninp)(mcp)=pm,m
     pm,m(s/),G(cdi),p1(coin)(mcp)=p,n,pp1
     p,pp1(-),0,n(dzm),G(mcp)=p2,G1,X
    G1,0(coin)(mcp)=G2,i
i:i(inc)=i
    X,pp1(ctr)=x,X
    x(cpm)=x
    G1,x(rti)=G1
    X,p2(ctr)=x,X
    x(cpm)=x
    G2,x(rti)=G2
    i,m(ifj)=i,j,j
j:(nop)=
     G1,m(cmp),G2,m(cmp)(mcp,t)=G1,G2
    (lis)=
```

RTPM.SUB

```
    A,B(RTpm,sub)=C
    (nli)=
    _Row_tie_Polynomial_matrices
    _|A(s)|
    _|----|==>C(s)
    _|B(s)|
     A(rdi),A(cdi),A(ninp)(mcp)=pm1,n1,m1
     B(rdi),B(cdi),B(ninp)(mcp)=pm2,n2,m2
     pm1,m1(s/),pm2,m2(s/),n1(mcp)=p1,p2,n12
    m1,m2(ifj)=e,a,e
e:(nop)=
     _A_&_B_should_have_same_#_of_columns
    m1,m2(cti)=-
    -(out)=
    0(coin)=C
    (jmp)=q
a:n2,n12(ifj)=s,s,g
g:n2(mcp)=n12
s:pm1,pm2(+),n12(dzm)=C
     A(cpm),B(cpm),0(coin),1(coin)(mcp)=Ax,Bx,i,l
i:i(inc)=i
    Ax,p1(ctr,t)=x,Ax
    C,x,l,1(rmp,t)=C
    l,p1,p2(+)(+)=l
    i,m1(ifj)=i,j,j
j:0(coin),p1(inc)(mcp)=i,l
I:i(inc)=i
    Bx,p2(ctr,t)=x,Bx
    C,x,l,1(rmp,t)=C
    l,p1,p2(+)(+)=l
    i,m2(ifj)=I,J,J
J:C,m1(cmp,t)=C
q:(lis)=
```

CTPM.SUB

```
    A,B(CTpm,sub)=C
    (nli)=
    _Column_tie_Polynomial_matrices
    _|A(s)|B(s)|==>C(s)
     A(rdi),A(cdi),A(ninp)(mcp)=pm1,n1,m1
     B(rdi),B(cdi),B(ninp)(mcp)=pm2,n2,m2
     pm1,m1(s/),pm2,m2(s/),n1(mcp)=p1,p2,n12
    p1,p2(ifj)=e,a,e
e:(nop)=
     _A_&_B_should_have_same_#_of_rows
    p1,p2(cti)=-
    -(out)=
    0(coin)=C
    (jmp)=q
a:n2,n12(ifj)=s,s,g
g:n2(mcp)=n12
s:pm1,pm2(+),n12(dzm)=C
     C,A(cpm),1,1(rmp),B(cpm),pm1(inc),1(rmp,t)=C
    C,m1,m2(+)(cmp,t)=C
q:(lis)=
```

ELZC.SUB

```
    G,eps(Elzc,sub)=Gr
    (nli)=
    G(mcp)=Gi
    0(coin)=Gr
     _Elimination_of_last_zero_columns
     _from_G(z)_in_PMF
    Gi(cdi)(inc),Gi(rdi)(mcp)=i,pm
i:i(dec)=i
    Gi,1,i,pm,1(exm)(t)=x
     x,x(t)(*)(sqr),eps(ifj)=c,c,f
c:i,1(ifj)=f,f,i
f:Gi,i(ctc,t)=Gr
    Gr,Gi(ninp)(cmp)=Gr
q:(lis)=
```

UYH.SUB

```
    u,y,L(uyh,sub)=H,nrm
    (nli)=
     _Input/Output_data_==>_Markov
    _parameters_H(z)_in_PMF
     _Applicable_only_to_DT_stable_systems
    u(rdi)=m
    m,m(mcp)=H,nrm
    u(t),1,L,2(Toep)=U
    U(rdi),U(cdi)(cti)=-
    -(out)=
    U(rdi),U(cdi)(ifj)=e,e,o
o:(nop)=
    U,y(t)(ale)(t)=H
    H,m,L(dec)(*)(ctc)=x,U
    U(nrr)=nrm
e:(nop)=
    H,m(pmfr)=H
    (lis)=
```

COMD.SBR

```
    Do,eps(ComD,sbr)=comd,F
    nli
    nty
    _Column_Do(z)_==>
    _Comm._Den_d(z)=comd
    I,2(dzm)(tvc)=comd,F
    Do(rdi)=p
    Do,1(coin)(mcp)=D,i
    D,1(ctr)=di,D
    di(elz)=comd
i:i(inc)=i
    D,1(ctr)=di,D
    di(elz)=di
    comd,di,eps(prd,t)
    =cdr,dir,com
    cdr,dir,com(p*)(p*,t)=comd
    i,p(ifj)=i,j,j
j:(nop)=
    comd(cdi)=dim
    D,comd(cdi)(dzm)=fpol
    Do(mcp)=D
    D(coin)=i
I:i(inc)=i
    D,1(ctr)=di,D
    di(elz)=di
    comd,di,eps(prd,t)=fi,x,y
     fpol,p(inc),dim(dzm),
    fi,1,1(rmp)(rti,t)=fpol
    i,p(ifj)=I,J,J
J:(nop)=
    fpol,p,p(*)(ctr,t)=F
    F,p(cmp),eps(elzc,sub)=F
q:typ
    lis
```

APPENDIX B

MIN.SUB

```
    A,B,C,eps(Min,sub)
    =Ao,Bo,Co,Tt
    (nli)=
    _Elimination_of
    _unobservable_modes
    _using_Hessenberg
    _Transformation
    _Called_twice_by_MIN.SBR
    A(rdi),C(rdi)(mcp)=n,p
    n,n(dim)=I
    C(svd)=w,u,v
    A,B,C,v(str)=A1,B1,C1
    v(mcp)=Tt
    n,0(inc),p(inc)(mcp)=np,i,j
    np,p(-)=np
i:(nop)=
    A1,i,j,p,np(exm,t)=Aij
    Aij(svd)=w,u,v
    I,v,j,j(rmp,t)=T
    Tt,T(*)=Tt
    A1,B1,C1,T(str)=A1,B1,C1
     i(inc),j(inc),np(dec)(mcp)
    =i,j,np
    j,n(ifj)=i,i,G
G:(nop)=
    A,B,C,Tt(str)=At,Bt,Ct
    n(mcp)=deg
    1(dec)=i
I:i(inc)=i
    n,i(-)=ni
    ni(inc)=ni1
    At,1,ni1,ni,i(exm)=z
    z(nrr,t)=z
    z,eps(ifj)=a,a,g
a:i(mcp,t)=deg
    ni(mcp)=deg
g:(nop)=
    i,n,p(-)(ifj)=I,K,K
K:(nop)=
    At,1,1,deg,deg(exm)=Ao
    B(cdi)=m
    Ct,1,1,p,deg(exm)=Co
    Bt,1,1,deg,m(exm)=Bo
    (lis)=
```

MIN.SBR

```
    A,B,C,eps(min,sbr)=Am,Bm,Cm
    nli
    nty
    _Minimal_Realization_using
    _Hessenberg_transformation
    _Calls_twice_MIN.SUB
    A,B,C,eps(min,sub)
    =Ao,Bo,Co,T1
    Ao(t),Co(t),Bo(t),eps
    (min,sub)=Amd,Bmd,Cmd,T2
    Amd,Cmd,Bmd(t,t)=Am,Bm,Cm
    typ
    lis
```

KALD.SBR

```
    A,B,C,eps(Kald,sbr)
    =Ad,Bd,Cd,T,dim
    nli
    nty
    _Kalman_decomposition
    _(A,B,C)_is_nC_&_nO
    _==>_Min_(Ad,Bd,Cd)
    _T_=_similarity_transform
    _dim_(1_x_4)=_Subspace
    _dimensions
 1,5(dzm)(tvc)=Ad,Bd,Cd,T,dim
  B(cdi),C(rdi)(mcp)=m,p
  A,B(qc)=Qc
  A,C(qo)=Qo
  Qc,eps(nrs)=x,Tc,X
  Qo(t),eps(nrs)=x,Rqot,X
  Rqot(t),eps(nrs)=Tob
  Tc,Tob,eps(INOU,sub)
  =COb,CbOb
  Tob,Tc,eps(INOU,sub)=COb,CO
  COb,CO,CbOb(cti,t)=T1
  T1(t),eps(nrs)=CbO
   COb(cdi),CO(cdi),CbOb
  (cdi),CbO(cdi)(cti)=dim
  T1,CbO(cti)=T
  dim(tvc)=cbo,co,bcbo,bco
  cbo(inc,t)=i
  A,B,C,T(str)=Ad,Bd,Cd
  Ad,i,i,co,co(exm)=Ad
  Bd,i,1,co,m(exm,t)=Bd
  Cd,1,i,p,co(exm,t)=Cd
q:typ
  lis
```

INOU.SUB

```
  R,Q,eps(InOu,sub)=Qr,Qou
  (nli)=
   _Q_is_decomposed_into_Qr
  _&_Qou
  _Qr_is_in_range_of_R
  _Qou_is_out_of_range_of_R
  Q(cdi)=k
  Q,R(cti),eps(nrs)=Nqr
  Nqr,k(ctr)=Nq
  Nq(t),eps(nrs)=Nqn
  Q,Nq(*)=Qr
  Q,Nqn(*)=Qou
  (lis)=
```

MODM.SBR

```
  A,Egv,eps(ModM,sbr)=P
  nli
  nty
  _Calculates_Modal_matrix_of
  _non-diagonalizable_A
  _Satisfies_A*P=P*Aj
  _Aj_=_Block_diagonal_Jordan
  _Form
  eps(mcp)=P
  A(rdi)=n
  Egv(rdi)=m
  O(dma)=z
  n,O(dzm)=Zm
  Zm(mcp)=P
  z(mcp)=j
j:j(inc)=j
  Egv,j,1,1,2(exm)=egj
  j(out)=
  egj,1(ctc)=sj,oj
  oj(ifj)=C,R,C
C:oj(abs)=oj
  A,sj,oj,eps(ChaC,sbr)=Pj
  nli
  nty
  (jmp)=y
R:(nop)=
  A,sj,eps(ChaR,sbr)=Pj
  nli
  nty
y:(nop)=
  P,Pj(cti,t)=P
  P(out)=
  j,m(ifj)=j,f,f
f:(nop)=
q:typ
  lis
```

CHAC.SBR

```
  A,sj,oj,eps(ChaC,sbr)=Pj
  nli
  nty
   _Calculates_eigenvectors
  _in_Pj
   _associated_with_complex
  _conj._pair
   _wj=sj+/-j*oj.Satisfies
   _A*Pj=Pj*Aj
  _Aj=Jordan_block_associated
  _with_wj
  eps(mcp)=Pj
  A(rdi)=n
  n,n(*)=n2
  O(dma)=z
  n,n(dim)=I
  n2,n2(dim)=I2
  n,O(dzm)=Zm
  Zm(mcp)=Pj
  A,I,sj(s*)(-)=Bjr
  I,oj(s*)=Bji
   Bjr,Bji(cti),Bji,-1(s*),
  Bjr(cti)(rti)=Bj
  Bj,eps(nrs)=N,R,r
  N(cdi)=vj
  I2,z,z(mcp)=Bk,k,r
k:k(inc)=k
  Bk,Bj(*)=Bk
  Bk,eps(nrs)=Nk
  R,Nk,eps(InOu,sub)=Y,M
  M(cdi)=q
  M,q,2(s/)(ctc)=Mt
  q(ifj)=k,k,a
a:(nop)=
  Mt,1(ctc)=m,Mt
  z,Zm(mcp)=i,Pi
i:i(inc)=i
  m,n(ctr)=mr,mi
  mr,mi,Pi(cti,t)=Pi
  Bj,m(*)=m
  i,k(ifj)=i,z,z
z:(nop)=
    Pj,Pi(cti,t)=Pj
    Mt(cdi)(ifj)=x,x,a
x:(nop)=
    R,M(cti,t)=R
    r,q(+,t)=r
    r,vj(ifj,t)=k,q,q
q:(nop)=
    typ
    lis
```

CHAR.SBR

```
    A,sig,eps(ChaR,sbr)=Pj
    nli
    nty
     _Calculates_Eigenvectors
    _in_Pj
    _associated_with_real
    _eigenvalue_sig
    _Satisfies_A*Pj=Pj*Aj
    _Aj_=_Jordan_block
    _associated _with_real_sig
    eps(mcp)=Pj
    A(rdi)=n
    O(dma)=z
    n,n(dim)=I
    n,O(dzm)=Zm
    Zm(mcp)=Pj
    A,I,sig(s*)(-)=Bj
    Bj,eps(nrs)=N,R,x
    N(cdi)=vj
    I,z,z(mcp)=Bk,k,r
k:k(inc)=k
    Bk,Bj(*)=Bk
    Bk,eps(nrs)=Nk
    R,Nk,eps(InOu,sub)=Y,M
    M(cdi)=q
    M(mcp)=Mt
    q(ifj)=k,k,a
a:(nop)=
    Mt,l(ctc)=m,Mt
    z,Zm(mcp)=i,Pi
i:i(inc)=i
    m,Pi(cti,t)=Pi
    Bj,m(*)=m
    i,k(ifj)=i,z,z
z:(nop)=
    Pj,Pi(cti,t)=Pj
    Mt(cdi)(ifj)=x,x,a
x:(nop)=
    R,M(cti,t)=R
    r,q(+,t)=r
    r,vj(ifj,t)=k,q,q
q:(nop)=
    typ
    lis
```

COTS.SBR

```
    A,B,C,im,eps(COts,sbr)
    =Resc,Reso,xxc,xxo
    nli
    nty
    _Degrees_of_Controllability
    _or_Observability
     _Resc_&_Reso_contain_n_
    _eigenvalues_of_A
    _and_im*(n-m)/im*(n-p)
    _eigenvalues_of
    _auxiliary_matrices_Acc/Aoo
     _Arrays_xxc/xxo_are_to_be
    _used_for_plotting
    _with_xx<c/o>(NIK)=
    _outside_subroutine
    _if desired!
    1,4(dzm)(tvc)=Resc,Reso,
    xxc,xxo
    A,B,C(getd,sub)=n,m,p
    n,m(-),n,p(-)(+),0(dzm)=Res
    A,B,C(mcp)=Ao,Bo,Co
    0(coin)=i
b:(nop)=
    i(inc)=i
    B(t),eps(nrs,t)=Nr
    C,eps(nrs,t)=Nc
    Nc(t),A,Nc(*)(*,t)=Aoo
    Nr(t),A,Nr(*)(*,t)=Acc
    Acc(egv,t)=egc
    Aoo(egv,t)=ego
    egc,ego(rti)=egco
    n,n,1,1(dpm)=T
    Ao,Bo,Co,T(str)=A,B,C
    Res,egco(cti,t)=Res
    i,im(ifj)=b,f,f
f:(nop)=
    A(egv)=egg
    egg(out)=
    Res,n,m(-)(ctr)=Resc,Reso
    Reso,2(r2c,t)=Reso
    egg,Reso(rti)=Reso
    Reso(polp,sub)=xxo
    Resc,2(r2c,t)=Resc
    egg,Resc(rti)=Resc
    Resc(polp,sub)=xxc
q:typ
    lis
```

C.7 References

The reference list for this appendix includes several articles which present the research which contributed toward the development of the *L-A-S* software. Interested readers are directed to the proceedings of the IFAC CAD Symposia, particularly those of 1982, 1985, 1988 and 1991, representing the 2nd, 3rd, 4th and 5th IFAC Symposia held in Lafayette, Indiana; Lyngby, Denmark; Beijing, People's Republic of China; and Swansea, United Kingdom, respectively. Also included as useful references are M. Jamshidi's texts on computer-aided design (CAD) and a manual for *L-A-S* which has been in use for some time.

Bingulac, S. (1983), "Recent modification in the *L-A-S* language and its use in CAD of control systems," *Proceedings of the 21st Allerton Conference*, University of Illinois, pp. 393-401.

Bingulac, S. (1988), "CAD package *L-A-S*: a research and educational tool in systems and control," *Proceedings of the 20th Southeastern Symposium on System Theory*, Charlotte, NC, pp. 44-49.

Bingulac, S., and D.L. Cooper (1990), *L-A-S User's Manual*, Virginia Polytechnic Institute and State University, Department of Electrical Engineering Report, 235 pages.

Bingulac, S., and M. Farias (1979), "*L-A-S* (linear algebra and systems) language and its use in control system education and research," *Journal of Computing*, Springer Verlag, **23**, pp. 1-23.

Bingulac, S., and N. Gluhajic (1982), "Computer-aided design of control systems on mini-computers using the *L-A-S* language," *Proceedings of the 2nd IFAC Symposium on CAD in Control*, Lafayette, IN, pp. 277-284.

Bingulac, S., P.J. West, and W.R. Perkins (1985), "Recent advances in the *L-A-S* software used in CAD of control systems," *Proceedings of the 3rd IFAC Symposium on CAD in Control*, Lyngby, Denmark, pp. 134-139.

Jamshidi, M., and C.J. Hegert (1985), *Computer-Aided Control System Engineering*, Elsevier Science Publishing Co. (North Holland), Amsterdam.

Jamshidi, M., M. Tarokh, and B. Shafai (1992), *Computer-Aided Analysis and Design of Linear Control Systems*, Prentice-Hall, Inc., Englewood Cliffs, NJ.

West, P.J., S. Bingulac, and W.R. Perkins (1985), "*L-A-S*: a computer-aided control system design language," *Computer-Aided Control System Engineering* (M. Jamshidi and C.J. Hegert, editors), Elsevier Science Publishers B.V. (North-Holland), Amsterdam, pp. 243-261.

Index